

Session 3

7.8 Advanced Merging

Merge Conflicts

브랜치 병합(merging)은 GUI 툴을 사용하는게 더 좋은 것 같다.

브랜치를 병합하기 전에 항상 Working tree가 깨끗한 상태인지 확인하고 병합하는 게 좋다. 그래야 Merge 중에 문제가 생겼을 때 Merge를 취소할 수 있다.

Merge를 취소할 때는 `$ git merge --abort`를 사용하면 된다.

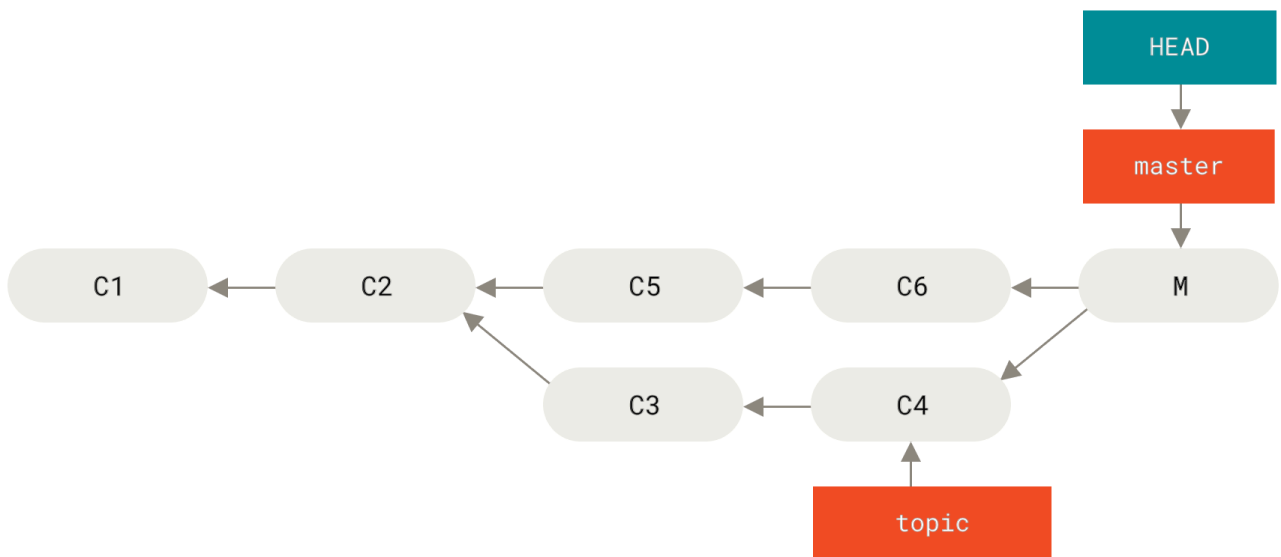
`$ git merge -Xignore-space-change` 명령어를 이용하면 빈칸에 의한 충돌을 무시할 수 있다.

Merge를 진행할 때는 **Stages**라는 개념이 있는데 Stage 1은 공통 조상 커밋, Stage 2는 merge target 브랜치, Stage 3는 merge source 브랜치를 나타낸다.

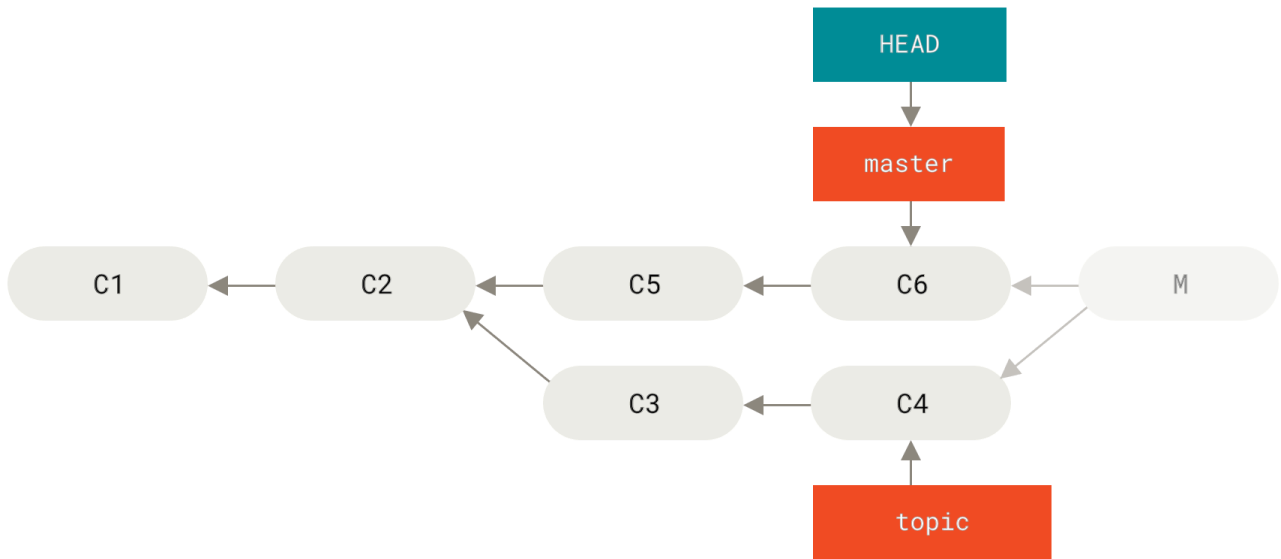
이후 `$ git merge-file` 명령어를 사용해 충돌을 resolve한 파일을 병합할 수 있다.

Undoing Merges

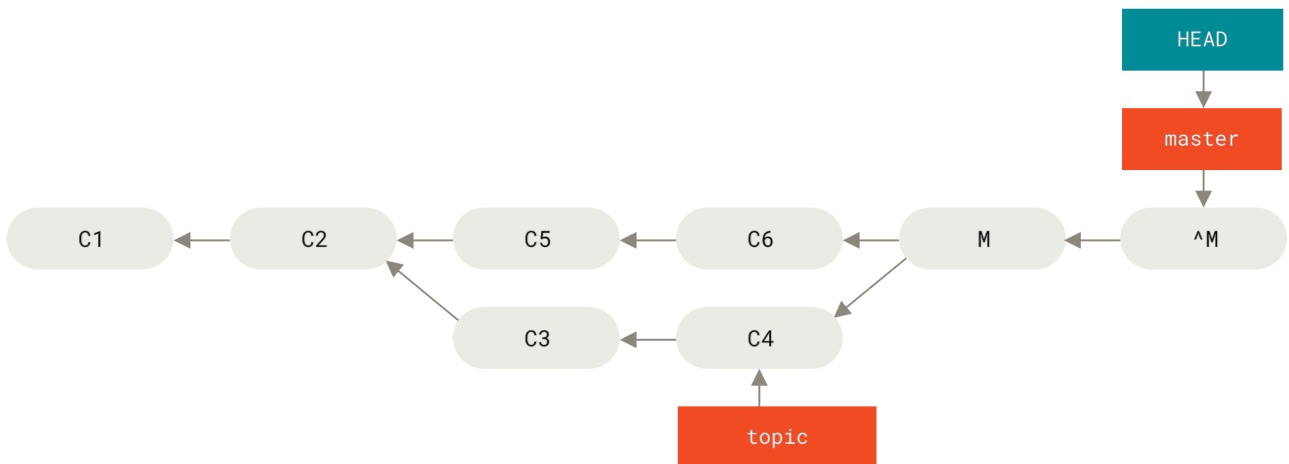
`master` 브랜치에서 잘못 병합했을 경우에는



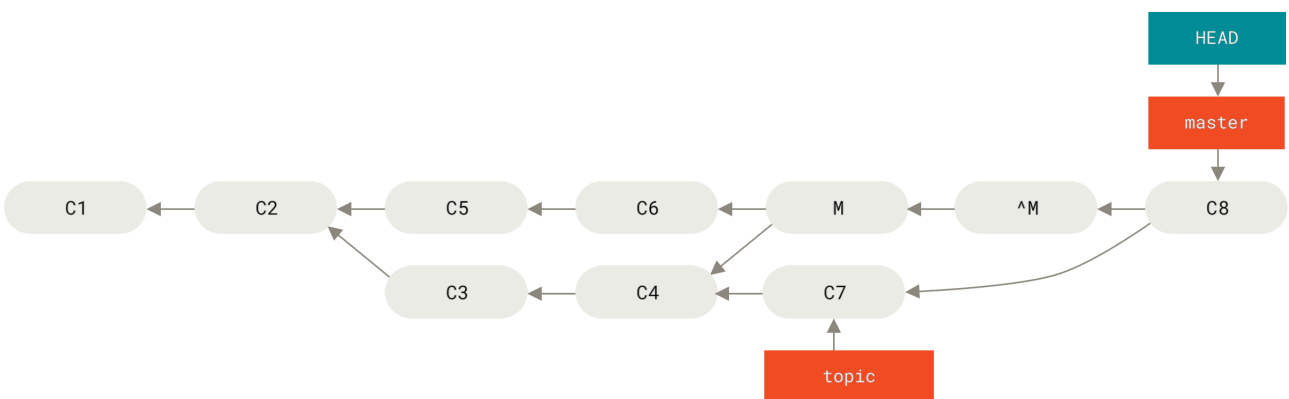
앞에서 배웠던 `$ git reset` 명령어를 이용해 손쉽게 merge를 취소할 수 있다. 하지만 `reset`은 publish되지 않은 브랜치에만 사용해야 하기 때문에 이미 publish를 한 상태라면 `$ git revert`를 사용해서 merge를 취소해야 한다.



하지만 아래와 같이 `$ git revert`를 이용해 **merge commit**을 취소하고 나면 이후에 `topic` 브랜치를 병합할 때 문제가 발생한다.

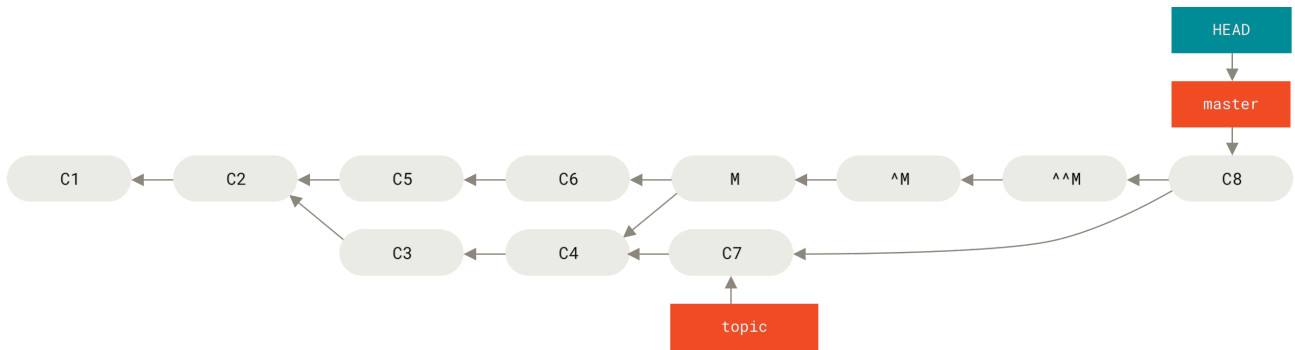


`C7` 커밋을 생성한 후 `$ git merge topic`을 하게 되면 **revert commit** `^M`으로 인해 `C3`, `C4`는 `master`에 이미 병합되었다 삭제된 내용이기 때문에 병합되지 않는다.



따라서 이런 상황이 발생할 경우 `^M`을 다시 revert한 후에 `merge`를 진행해야 `C3`, `C4`에 대한

내용을 `master`에 합칠 수 있다.



Other Types of Merges

`$ git merge`, `$git merge-file`을 할 때 `-Xours`나 `-Xtheirs` 옵션을 이용해 충돌이 발생할 경우에 자동으로 어느 쪽을 선택할지 지정할 수도 있다.

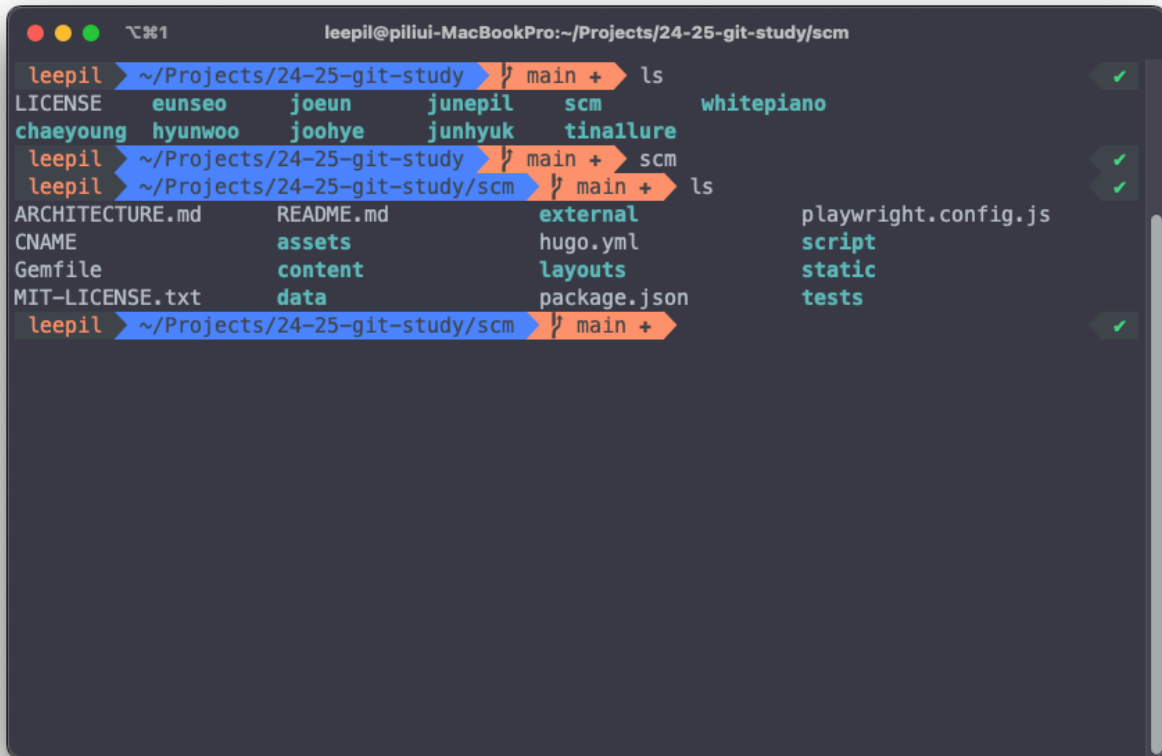
subtree merging을 이용하면 git repository 내부의 디렉토리에 다른 git repository를 넣어서 관리할 수 있다. `$ git read-tree` 명령어를 이용해 등록할 수 있다.

remote를 등록한 다음에 checkout을 하면 같은 디렉토리에 전혀 다른 파일들이 있는 걸 볼 수 있다.

```
leepil@piliui-MacBookPro:~/Projects/24-25-git-study
leepil > ~/Projects/24-25-git-study > main > ls
LICENSE      eunseo      joeun       junepil     tinallure
chaeyoung    hyunwoo     joohye      junhyuk     whitepiano
leepil > ~/Projects/24-25-git-study > main > git co -
Updating files: 100% (35013/35013), done.
Switched to branch 'scm'
Your branch is up to date with 'scm/main'.
leepil > ~/Projects/24-25-git-study > scm->scm/main ? > ls
ARCHITECTURE.md  README.md      external      playwright.config.js
CNAME            assets         hugo.yml      script
Gemfile          content        layouts       static
MIT-LICENSE.txt  data          package.json  tests
leepil > ~/Projects/24-25-git-study > scm->scm/main ? > _
```

`git read-tree --prefix=scm/ -u scm` 명령어를 입력하면 하위 디렉토리 내부에 다른 git repository의 내용이 들어가 있는 걸 볼 수 있다. 이제 `scm` 브랜치와 `main` 브랜치에서 서로 변

경사향을 공유할 수 있게 된다.



```
leepil ~/Projects/24-25-git-study 1 main + ls
LICENSE eunseo joeun junepil scm whitepiano
chaeyoung hyunwoo joohye junhyuk tinaallure
leepil ~/Projects/24-25-git-study 1 main + scm
leepil ~/Projects/24-25-git-study/scm 1 main + ls
ARCHITECTURE.md README.md external playwright.config.js
CNAME assets hugo.yml script
Gemfile content layouts static
MIT-LICENSE.txt data package.json tests
leepil ~/Projects/24-25-git-study/scm 1 main +
```

사실 이 방법보다는 **submodule**이 더 유용한 것 같다.

7.9 Rerere

이것만 알면 당신도 git 고수!

Reuse Recorded Resolution의 약자로 merge시 충돌이 발생하면 이전에 동일한 충돌을 해결했을 경우 해결 방법을 기억했다가 자동으로 resolve 시켜주는 기능이다.

예를 들어 merge나 rebase를 진행했다 `$ git reset`을 했을 때 이전의 작업을 git이 알아서 해준다.

`$ git config --global rerere.enabled true`로 활성화해놓으면 언젠가 도움이 될 것이다.

7.11 Submodules

Starting with Submodules

git repository 안에 다른 git repository를 넣어서 관리할 수 있게 해준다. `$ git submodule add` 명령어를 이용해 하위디렉토리에 repository를 추가할 수 있다.

git-study 레포지토리에 js-deep-dive 레포지토리를 서브모듈로 추가한 모습은 아래와 같다. `.gitmodules`라는 파일이 새로 생성됐고, 디렉토리도 만들어진 걸 볼 수 있다.

. `gitmodules` 는 서브모듈을 관리하는 파일이며 서브모듈에 대응하는 디렉토리와 사용하는 브랜치를 저장하고 있다.

```
leepil@piliui-MacBookPro:~/Projects/24-25-git-study
tudy-js-deep-dive
leepil ~/Projects/24-25-git-study submodule ls -al
total 40
drwxr-xr-x 17 leepil staff 544 Nov 10 17:09 .
drwxr-xr-x 13 leepil staff 416 Nov 9 12:08 ..
-rw-r--r--@ 1 leepil staff 6148 Nov 4 17:54 .DS_Store
drwxr-xr-x 18 leepil staff 576 Nov 10 17:10 .git
-rw-r--r-- 1 leepil staff 2060 Nov 10 16:18 .gitignore
-rw-r--r-- 1 leepil staff 135 Nov 10 17:09 .gitmodules
drwxr-xr-x 14 leepil staff 448 Nov 10 17:09 24-25-study-js-deep-dive
-rw-r--r-- 1 leepil staff 1088 Nov 10 16:18 LICENSE
drwxr-xr-x 3 leepil staff 96 Nov 10 16:18 chaeyoung
drwxr-xr-x 3 leepil staff 96 Nov 10 16:18 eunseo
drwxr-xr-x 3 leepil staff 96 Nov 10 16:18 hyunwoo
drwxr-xr-x 4 leepil staff 128 Nov 10 16:18 joeun
drwxr-xr-x 3 leepil staff 96 Nov 10 16:18 joohye
drwxr-xr-x 4 leepil staff 128 Nov 10 16:39 junepil
drwxr-xr-x 4 leepil staff 128 Nov 10 16:18 junhyuk
drwxr-xr-x 3 leepil staff 96 Nov 10 16:18 tinallure
drwxr-xr-x 3 leepil staff 96 Nov 10 16:18 whitepiano
leepil ~/Projects/24-25-git-study submodule cat .gitmodules
[submodule "24-25-study-js-deep-dive"]
    path = 24-25-study-js-deep-dive
    url = git@github.com:gdsc-konkuk/24-25-study-js-deep-dive.git
leepil ~/Projects/24-25-git-study submodule _
```

Cloning a Project with Submodules

서브모듈을 포함하는 레포지토리를 clone할 때는 서브모듈에 해당하는 디렉토리가 빈 디렉토리인 상태이기 때문에 아래의 명령어들을 순차적으로 실행해 줘야 한다.

1. `$ git submodule init`
2. `$ git submodule update`

또는 `$ git clone --recurse-submodules` 를 이용해 중첩된 서브모듈들을 자동으로 복제할 수도 있다. 중첩된 서브모듈들을 전부 사용하려면 `$ git submodule update --init --recursive` 명령어를 입력하면 된다.

Working on a Project with Submodules

Pulling in Upstream Changes from the Submodule Remote

서브모듈에 있는 내용을 단순히 사용하기만 할 때에는 서브모듈의 디렉토리로 이동해 `$ git fetch` 와 `$ git merge` 를 사용하거나 `$ git merge` 를 사용하면 된다.

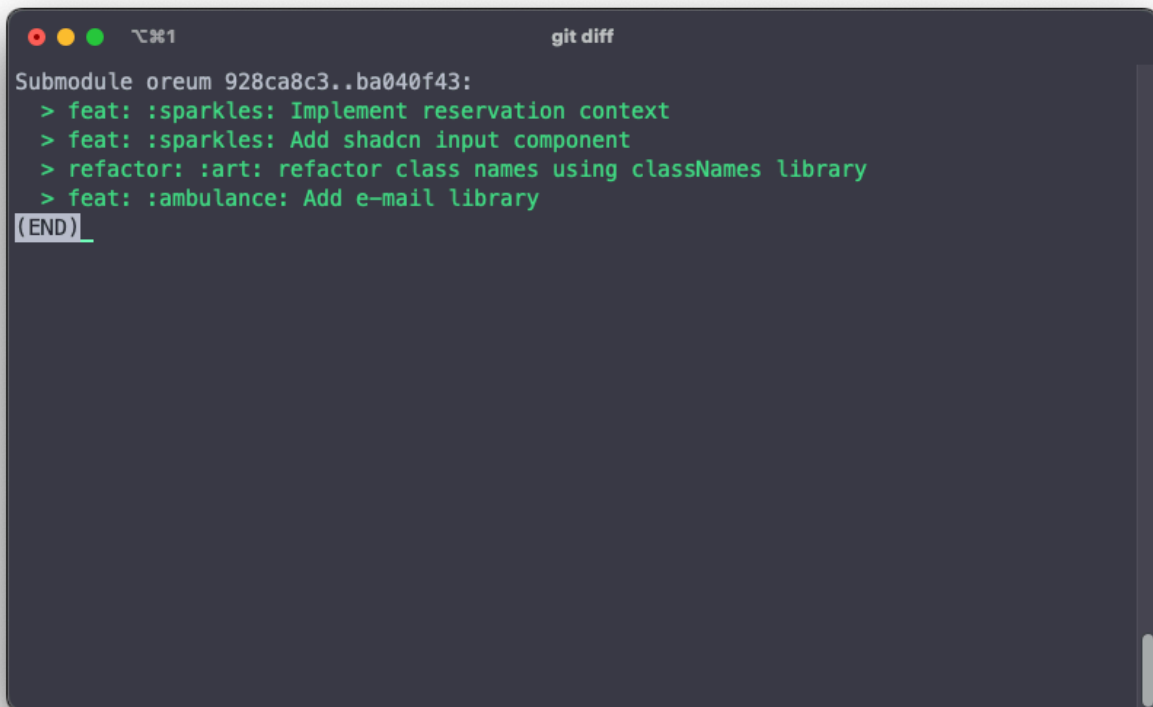
이렇게 서브모듈을 업데이트하면 아래와 같이 `oreum` 에 새로운 커밋이 있다는 내용을 볼 수 있고

```
leepil@piliui-MacBookPro:~/Projects/24-25-git-study
leepil ~/Projects/24-25-git-study » submodule clear
leepil ~/Projects/24-25-git-study » submodule oreum
leepil ~/Projects/24-25-git-study/oreum » main git fetch
remote: Enumerating objects: 28, done.
remote: Counting objects: 100% (28/28), done.
remote: Compressing objects: 100% (11/11), done.
remote: Total 19 (delta 8), reused 19 (delta 8), pack-reused 0 (from 0)
Unpacking objects: 100% (19/19), 2.96 KiB | 151.00 KiB/s, done.
From github.com:hanu9257/oreum
 928ca8c..ba040f4  main      -> origin/main
leepil ~/Projects/24-25-git-study/oreum » main ↵ git pull
Updating 928ca8c..ba040f4
Fast-forward
 app/reservation/context.ts | 16 ++++++++
 app/reservation/layout.tsx | 15 ++++++++
 components/calender.tsx    | 37 ++++++++
 components/ui/input.tsx    | 25 ++++++++
 package-lock.json          | 10 ++++++++
 package.json               |  1 +
 6 files changed, 85 insertions(+), 19 deletions(-)
 create mode 100644 app/reservation/context.ts
 create mode 100644 components/ui/input.tsx
leepil ~/Projects/24-25-git-study/oreum » main gloga
leepil ~/Projects/24-25-git-study/oreum » main ..
leepil ~/Projects/24-25-git-study » submodule git status
On branch submodule
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   oreum (new commits)

no changes added to commit (use "git add" and/or "git commit -a")
leepil ~/Projects/24-25-git-study » submodule
```

`$ git diff` 를 통해 서브모듈 `oreum` 이라는 서브모듈에 4개의 커밋이 추가됐다는 기록 또한 볼 수 있다.

`git config --global diff.submodule log` 를 적용해야 아래와 같은 출력을 볼 수 있다.



```
git diff
Submodule oreum 928ca8c3..ba040f43:
> feat: :sparkles: Implement reservation context
> feat: :sparkles: Add shadcn input component
> refactor: :art: refactor class names using classNames library
> feat: :ambulance: Add e-mail library
(END)
```

프로젝트의 루트 디렉토리에서 `$ git pull` 을 하는 경우 서브모듈들에 변경사항이 적용되지는 않기 때문에 `$ git submodule update` 를 이용해 직접 업데이트를 해줘야 한다.

`$ git submodule update --init --recursive` 를 기본적으로 사용해야 **pull**을 했을 때 새로운 서브모듈에 존재할 경우까지 한번에 처리할 수 있다.

Working on a Submodule

`$ git submodule update` 를 이용할 경우에 내려받은 변경사항은 로컬 브랜치에 속하지 않는 커밋의 상태로 남게 된다. 따라서 각 서브모듈들의 브랜치에 어떤 방식으로 변경사항들을 병합할지 선택해야 한다. 예를 들어 `$ git submodule update --remote --merge|--rebase` 를 입력하게 되면 추가된 변경사항들을 내 로컬 브랜치들에 자동으로 병합해준다.

만약 로컬에 있는 변경사항들을 커밋하지 않았다면 git이 Merge나 Rebase를 수행하지 않고 변경사항들을 불러오기만 한다.

Publishing Submodule Changes

로컬에서 서브모듈에 변경사항을 만들었을 때는 서브모듈에 만든 변경사항들까지 전부 push하지 않을 경우 다른 사람들이 레포지토리를 pull했을 때 문제가 발생할 수 있다.

이런 상황을 막기 위해 `git push --recurse-submodules=check` 옵션을 사용할 수 있다. 서브모듈에 변경사항이 있고 publish되지 않았을 경우 push를 취소하는 옵션이다. 만약 `--recurse-submodules=on-demand` 로 설정하면, 서브모듈들에 존재하는 변경사항들을 모두 publish 해준다.

Merging Submodule Changes

\$ `git pull` 을 이용해 변경사항을 가져오면 서브모듈의 경우 자동으로 변경사항을 병합해주지 않고 그냥 내버려 둔다. \$ `git diff` 를 통해 충돌하는 변경사항을 확인할 수 있어 **SHA-1**를 확인하고 아래와 같이 브랜치를 생성해 병합하면 된다.

```
$ cd DbConnector

$ git rev-parse HEAD
eb41d764bccf88be77aced643c13a7fa86714135

$ git branch try-merge c771610

$ git merge try-merge
Auto-merging src/main.c
CONFLICT (content): Merge conflict in src/main.c
Recorded preimage for 'src/main.c'
Automatic merge failed; fix conflicts and then commit the result.
```

이미 기존에 **merge commit**이 존재할 경우에는 해당 커밋을 사용할지 git이 물어보기도 한다.

Submodules Tips

\$ `git submodule foreach` 를 이용해 모든 서브모듈을 순회하며 동일한 명령어를 적용할 수 있다.

디렉토리를 서브모듈로 교체하는 경우에 \$ `git rm` 명령어를 이용해 git이 추적하지 않도록 디렉토리를 지우고 \$ `git submodule add` 를 해야 한다. 그리고 서브모듈이 적용된 브랜치와 그렇지 않은 브랜치 사이를 이동할 때 디렉토리 내의 파일이 사라질 수 있기 때문에 주의해서 이동해야 한다.

7.12 Bundling

네트워크를 사용할 수 없는 환경에서 git repository를 압축해서 하나의 파일로 공유할 수 있게 해주는 방법이다. \$ `git bundle create <output> <commit> <branch>` 명령어를 이용해 레포지토리를 압축할 수 있다.

이렇게 압축된 번들 파일을 이용해서 레포지토리를 생성할 때는 \$ `git clone <file> <path>` 를 이용하면 된다.

commit ranges를 이용해 레포지토리에서의 변경 사항만 번들로 만들고 공유할 수도 있는데 \$ `git bundle create commits.bundle HEAD ^HEAD~3` 과 같이 3개의 커밋만을 담은 번들 파일을 만들어 사용하면 된다.

이렇게 생성된 번들 파일은 \$ `git bundle verify` 를 사용하면 특정 번들 파일이 현재 레포지토리에 합쳐질 수 있는지 아닌지 여부를 판단할 수 있다. `list-heads` 를 입력하면 어떤 브랜치가 해당 변경사항에 들어있는지 보여준다.

```
$ git bundle list-heads ../commits.bundle
71b84daaf49abed142a373b6e5c59a22dc6560dc refs/heads/master
```

이렇게 `master` 브랜치에 해당하는 변경사항이 있다는 걸 확인했다면

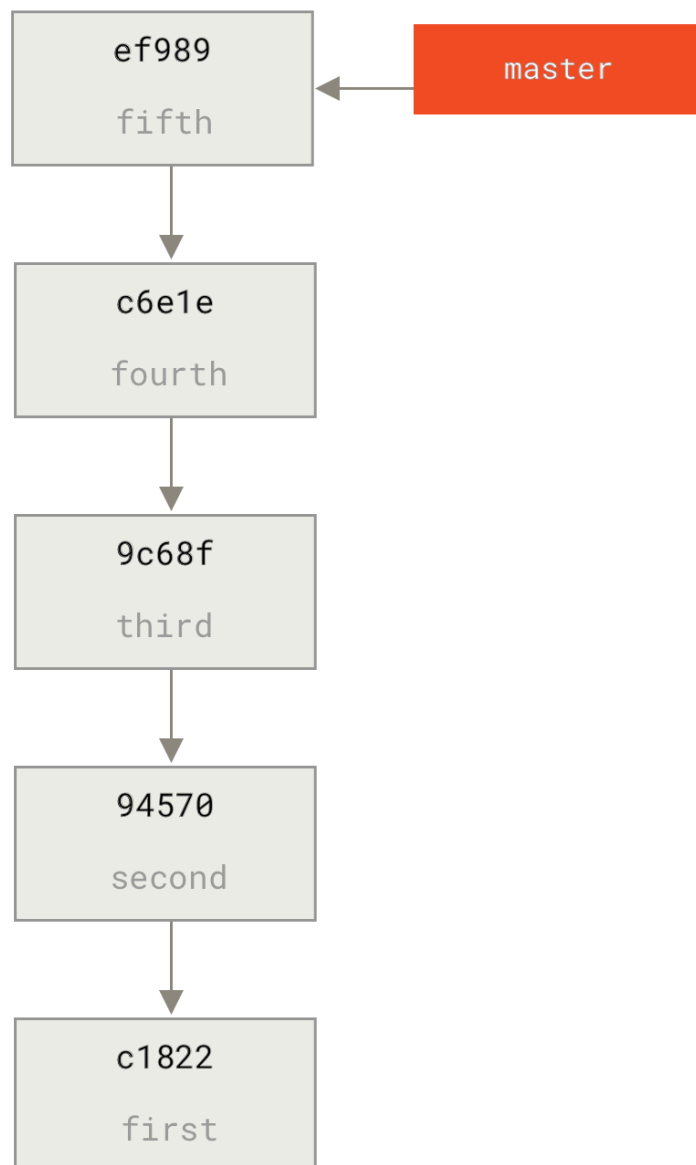

```
$ git fetch ../commits.bundle master:other-master
From ../commits.bundle
* [new branch]      master    → other-master
```

`fetch` 명령어를 이용해 변경사항을 번들로부터 불러올 수 있다. 이 예시는 번들에 있는 `master`의 내용을 로컬 레포지토리의 `other-master`에 추가해준다.

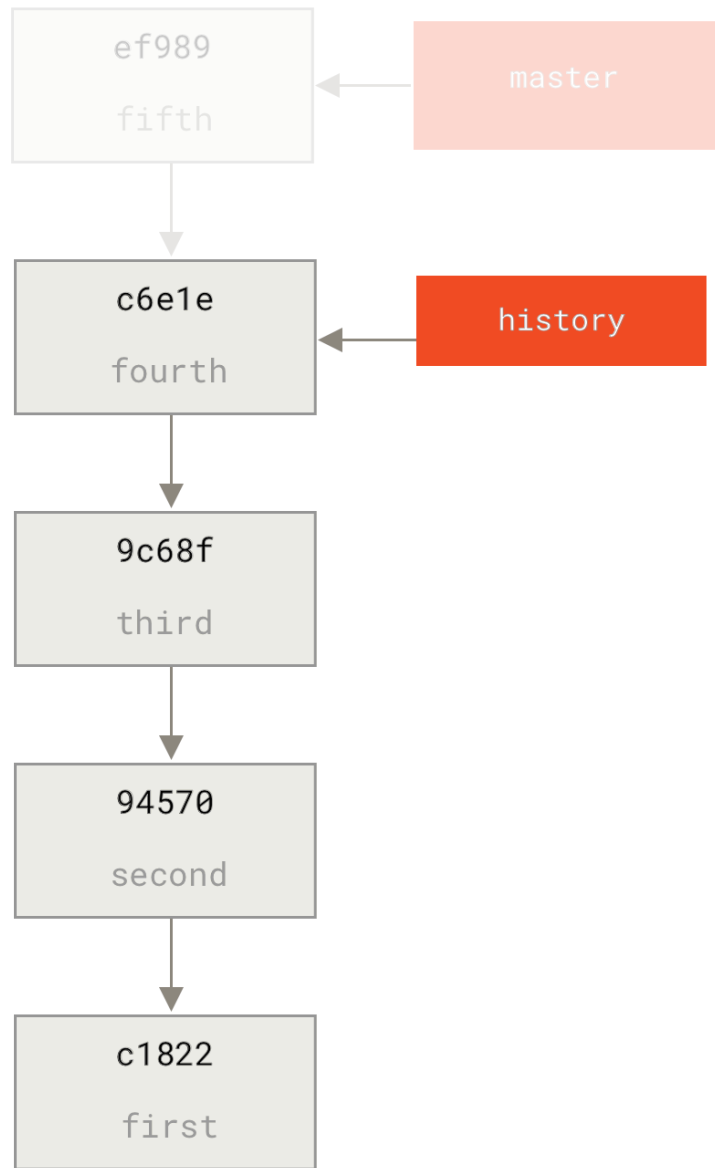
7.13 Replace

특정 git object를 다른 git object로 대체시켜준다. 어떻게 사용할 수 있는지 아래의 예시를 통해 알아보자.

우리는 `master` 브랜치를 최신 변경사항을 담은 레포지토리와 과거의 기록을 모두 저장하고 있는 레포지토리 2개로 분리하고 싶다. `master`는 아래와 같은 상황이다.



우선 `history`라는 브랜치를 `HEAD~`를 가르키게 생성하고, 이 브랜치를 새로운 레포지토리의 `main` 브랜치로 만든다.

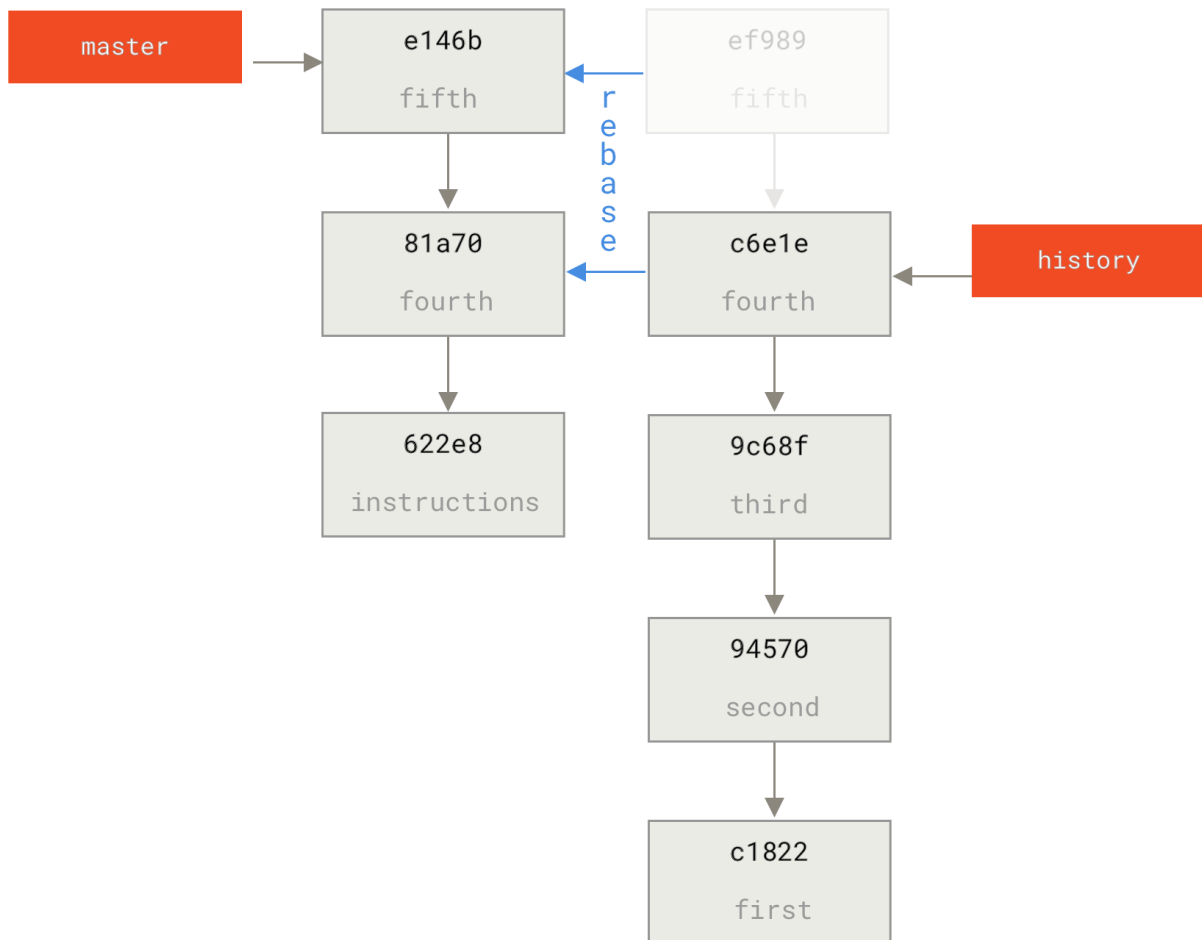


`$ git commit-tree 9c68fdc^{tree}` 명령어를 이용해 최신의 내역만을 담은 레포지토리의 **base commit**으로 사용할 커밋을 생성했다.

`9c68fdc^{tree}` 는 해당 커밋 오브젝트의 tree의 내용을 담는다는 뜻이다.



이제 `git rebase --onto 622e88 9c68fdc` 를 이용해 `master` 의 내용을 `622e8` 에서부터 시작하도록 만들 수 있다. 이렇게 과거의 기록을 담은 레포지토리, 최신 변경사항들을 남은 레포지토리 두 개로 분리가 가능하다.



이제 각각의 레포지토리에 대한 **remote** 설정을 하고 각 브랜치에 대한 정보를 불러오면 다음과 같은 기록이 보일 것이다.

```

$ git log --oneline master
e146b5f Fifth commit
81a708d Fourth commit
622e88e Get history from blah blah blah

$ git log --oneline project-history/master
c6e1e95 Fourth commit
9c68fdc Third commit
945704c Second commit
c1822cf First commit

```

이제 여기서 `$ git replace 81a708d c6e1e95` 를 하게 되면 **replacing**된 것이다. **replace**를 하더라도 여전히 `81a708d` 라는 SHA-1 값으로 커밋을 지칭할 수 있다.

하지만 내용은 `c6e1e95` 의 내용이 나온다.

그리고 **replace**한 기록은 **ref**에 남아있는다.

```

$ git for-each-ref
e146b5f14e79d4935160c0e83fb9ebe526b8da0d commit refs/heads/master

```

```
c6e1e95051d41771a649f3145423f8809d1a74d4 commit refs/remotes/history/master
e146b5f14e79d4935160c0e83fb9ebe526b8da0d commit refs/remotes/origin/HEAD
e146b5f14e79d4935160c0e83fb9ebe526b8da0d commit refs/remotes/origin/master
c6e1e95051d41771a649f3145423f8809d1a74d4 commit
refs/replace/81a708dd0e167a3f691541c7a6463343bc457040
```

8.2 Git Attributes

git repository에 적용할 속성들을 정의하는 `.gitattributes` 파일을 이용해 여러 속성을 지정할 수 있다.

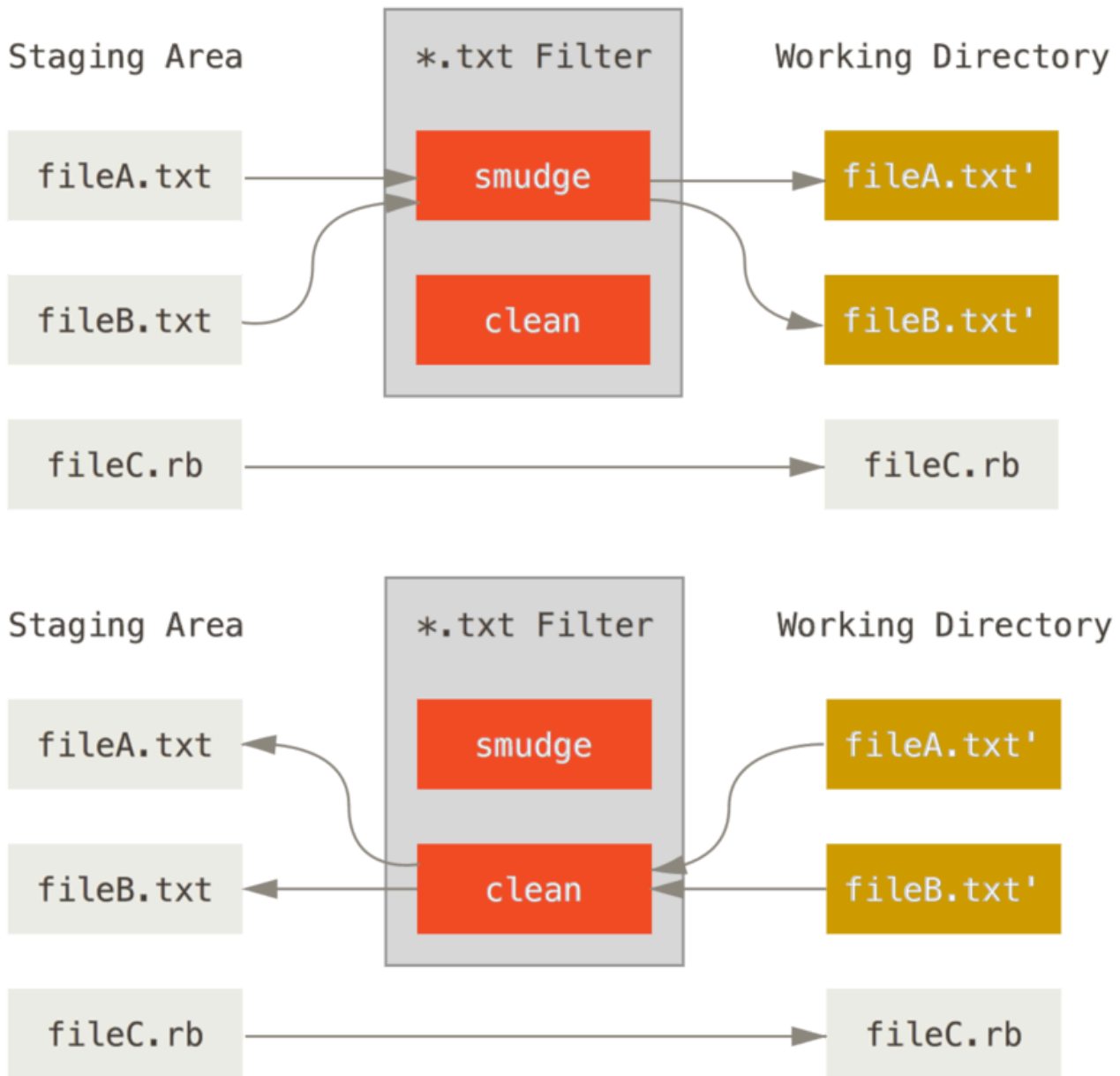
Binary Files

binary 파일등을 지정해 `show` 나 `diff` 명령어의 대상에서 제외할 수 있다. 그리고 binary, 이미지, docx 파일등이 git diff에서 어떻게 해석할지도 등록할 수 있다.

```
*.docx diff=word
```

위와 같이 설정하면 `word` 라는 필터를 `.docx` 확장자를 열 때 사용하도록 등록할 수 있는데, **docx2txt**와 같은 프로그램을 사용하면 워드 파일의 변경사항을 git에서 볼 수 있게 된다.

commit/checkout 시에 적용되는 필터를 등록할 수도 있는데 각각 `smudge`, `clean` 필터라고 한다.



파일마다 다른 병합 전략을 지정할 수도 있다.

8.4 An Example Git-Enforced Policy

`hooks/update` 파일을 활용해 Server-side Hook을 사용할 수 있다. 예를 들어 커밋 메시지에 특정 형식의 문자열이 있는지 확인하고 없을 경우 **push**를 취소시킬 수 있다.

ACL (Access Control List)를 함께 사용하면 git repository의 디렉토리마다 사용자의 접근 권한을 부여할 수도 있다.

client-side hook을 사용해 local 단계에서도 개인마다 정책을 구현할 수 있는데 client-side hook은 git으로 공유되지 않는다.