3주차

- 7.8 Advanced Merging
- 7.9 Rerere
- 7.11 Submodules
- 7.12 Bundling
- 7.13 Replace
- 8.2 Git Attributes
- 8.4 An Example Git-Enforced Policy

7.8

merge 충돌

워킹 디렉토리에 작업하던 게 있다면 임시 브랜치에 커밋하거나 Stash 해둠. 그래야 어떤 일이 일어나도 다시 되돌릴 수 있다. 작업 중인 파일을 저장하지 않은 채로 Merge 하면 작업했던 일부를 잃을 수도 있음.

merge 취소

Merge 중에 발생한 충돌을 해결하는 방법

1. `git merge --abort` : merge 하기 전으로 되돌림

   완전히 뒤로 되돌리지 못하는 경우 = Merge 전에 워킹 디렉토리에서 StashX or 커밋하지 않은 파일이 존재하고 있었을 때

2. `git reset --hard HEAD` : 워킹 디렉토리를 그 시점으로 완전히 되돌림(저장하지 않은 것은 사라짐)

공백 무시하기

공백이 충돌의 원인이 될 때가 있음→기본 Merge 전략은 공백의 변화는 무시하도록 하는 옵션을 줌

Merge 할 때 무수한 공백 때문에 문제가 생기면 그냥 Merge를 취소한 다음

`-Xignore-all-space` : 모든 공백을 무시

`-Xignore-space-change` : 뭉쳐 있는 공백을 하나로 취급

`$ git merge -Xignore-space-change whitespace` → 모든 공백 변경 사항을 무시하면 실제 파일은 충돌 나지 않고 모든 Merge가 잘 실행됨

수동으로 merge하기

파일을 `dos2unix` 로 변환하고 Merge 하기

충돌 파일 checkout

충돌 해결 이후의 결과가

merge 로그

`git log` : 로그에는 충돌을 해결할 때 도움이 될만한 정보가 있을 수 있음

`$ git log --oneline --left-right HEAD...MERGE_HEAD` : Merge 에 사용한 양 브랜치의 모든 커밋의 목록을 얻을 수 있음

`$ git log --oneline --left-right --merge` : 충돌이 발생한 파일이 속한 커밋만 보여줌

`-p` (merge 대신) : 충돌 난 파일의 변경사항만 볼 수 있음.

combined diff 형식

“Combined Diff” 형식

각 라인은 두 개의 컬럼으로 구분 가능. 첫 번째 컬럼은 “ours” 브랜치와 워킹 디렉토리의 차이(추가 또는 삭제), 두 번째 컬럼은 “theirs” 와 워킹 디렉토리사이의 차이

`<<<<<<<` & `>>>>>>>` → 어떤 쪽에도 존재하지 않고 추가된 코드

```
$ git diff
diff --cc hello.rb
index 0399cd5,59727f0..0000000
--- a/hello.rb
+++ b/hello.rb
@@@ -1,7 -1,7 +1,11 @@@
  #! /usr/bin/env ruby

  def hello
++<<<<<<< HEAD
 +  puts 'hola world'
```

```
++=======
+   puts 'hello mundo'
++>>>>>>> mundo
  end

  hello()
```

```
$ vim hello.rb
$ git diff
diff --cc hello.rb
index 0399cd5,59727f0..0000000
--- a/hello.rb
+++ b/hello.rb
@@@ -1,7 -1,7 +1,7 @@@
  #! /usr/bin/env ruby

  def hello
-   puts 'hola world'
 -  puts 'hello mundo'
++  puts 'hola mundo'
  end

  hello()
```

“hola world” 는 Our 브랜치에 있었지만 워킹 디렉토리에는 없다. “hello mundo” 는 Their 브랜치에 있었지만 워킹 디렉토리에는 없다. “hola mundo” 는 어느 쪽 브랜치에도 없고 워킹 디렉토리에는 있다.

merge 되돌리기

- refs 수정

  실수로 생긴 Merge 커밋이 로컬 저장소에만 있을 때→브랜치를 원하는 커밋을 가리키도록 옮기는 것이 쉽고 빠름

  `git reset --hard HEAD~` 명령으로 브랜치를 되돌리기

  단점:히스토리를 다시 쓴다는 것

- 커밋 되돌리기
  브랜치를 옮기는 것을 할 수 없는 경우→모든 변경사항을 취소하는 새로운 커밋을 만들

기(=revert)

다른 방식의 merge

- our/their 선택하기

  아무 옵션도 지정하지 않고 두 브랜치를 Merge 하면 Git은 코드에 충돌 난 곳을 표시하고 해당 파일을 충돌 난 파일로 표시해줌. 충돌을 직접 해결하는 게 아니라 미리 Git에게 충돌이 났을 때 두 브랜치 중 한쪽을 선택하라고 알려줄 수 있음. `merge` 명령을 사용할 때 `-Xours` 나 `Xtheirs` 옵션을 추가하면 됨

- 서브트리 merge

  프로젝트 두 개가 있을 때 한 프로젝트를 다른 프로젝트의 하위 디렉토리로 매핑하여 사용하는 것
  =메인 프로젝트로 서브프로젝트의 내용을 Merge

7.9

`git rerere` : Git에서 발생하는 충돌을 기록, 이후 동일한 충돌이 발생할 때 자동으로 해결하는 기능

`$ git config --global rerere.enabled true` : rerere 기능 활성화

`git rerere status` : 충돌 난 파일을 확인

`git rerere diff` : 해결 중인 상태를 확인

7.11

서브모듈= Git 저장소 안에 다른 Git 저장소를 디렉토리로 분리해 넣는 것

다른 독립된 Git 저장소를 Clone 해서 내 Git 저장소 안에 포함할 수 있음&각 저장소의 커밋은 독립적으로 관리

서브모듈 시작하기

`git submodule add URL주소` : 서브모듈을 추가하는 명령
기본적으로 서브모듈은 프로젝트 저장소의 이름으로 디렉토리를 만든다

```
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   .gitmodules #gitmodules 파일이 생성.서브디렉토리와 하위 프로젝트 URL의
                                                                                매
핑 정보를 담은 설정파일
    new file:   DbConnector
```

```
$ git diff --cached DbConnector
diff --git a/DbConnector b/DbConnector
new file mode 160000
index 0000000..c3f01dc
--- /dev/null
+++ b/DbConnector
@@ -0,0 +1 @@
+Subproject commit c3f01dc8862123d317dd46284b05b6892c7b29bc
#Git은 DbConnector 디렉토리를 서브모듈로 취급하기 때문에 해당 디렉토리 아래의 파일
수정사항을 직접 추적하지 않음.서브모듈 디렉토리를 통째로 특별한 커밋으로 취급
```

7.12

`git bundle` : 데이터를 한 파일에 몰아넣는 것

`git bundle create` : Bundle 생성

`$ git bundle create repo.bundle HEAD master` = `repo.bundle` 이라는 이름의 파일을 생성

먼저 Bundle 파일에 추가시킬 커밋의 범위를 정해야 함(차이점만 Bundle로 묶는 게 좋음)

`$ git log --oneline master ^origin/master` : 로컬에서 만든 세 개의 커밋 얻기

`$ git bundle create commits.bundle master ^9a466c5` : 로컬에서 만든 세 개의 커밋만 묶기

`bundle verify` : 파일이 올바른 Git Bundle인가, 제대로 적용하는 데 필요한 모든 히스토리가 현재 저장소에 있는가 확인

7.13

`replace` : 간단해진 히스토리를 전체 히스토리의 마지막 부분에 연결해서 사용할 수 있음. 히스토리를 변경하는 데도 커밋을 새로 쓰지x

히스토리 두개로 분리

1. 원래의 히스토리를 유지한 채, 두 개의 저장소로 분리

   네 번째 커밋을 기준으로 `history`라는 새로운 브랜치를 만들어 원래 히스토리의 일부를 별도로 유지

   ```
   $ git branch history c6e1e95
   $ git log --oneline --decorate
   ef989d8 (HEAD, master) fifth commit
   c6e1e95 (history) fourth commit
   9c68fdc third commit
   945704c second commit
   c1822cf first commit
   ```

   `history` 브랜치를 새 리모트 저장소에 push

   ```
   $ git remote add project-history https://github.com/schaco
   $ git push project-history history:master
   ```

2. 새 히스토리는 최신 커밋만 포함

   새로운 히스토리는 네 번째 커밋 이후의 커밋만 포함

   `git commit-tree` → 커밋을 추가 & `git rebase` → 조정

   세 번째 커밋을 기준으로 새로운 커밋을 만들기

   ```
   $ echo 'get history from blah blah blah' | git commit-tree
   622e88e9cbfbacfb75b5279245b9fb38dfea10cf
   ```

   새로 만든 커밋을 기준으로 네 번째 이후의 커밋을 rebase → 최신 커밋만 유지하는 새로운 히스토리 생성됨

```
$ git rebase --onto 622e88 9c68fdc
```

3. `replace` 사용하여 두 히스토리를 연결

   `git replace` : 새 히스토리의 커밋이 원래 히스토리에 속한 커밋을 가리키도록 함

8.2

`.gitattributes` 파일을 사용하면, Git이 특정 파일을 어떻게 처리할지 설정할 수 있음

`.git/info/attributes` 파일을 커밋하고 싶지X 경우

- 바이너리 파일

  `*.pbxproj binary` : pbxproj파일을 바이너리 파일로 설정

  Git Attribute를 통해 Git이 바이너리 파일을 텍스트 포맷으로 변환하고 그 결과를 `diff` 명령으로 비교하도록 함

- 키워드 치환

  *.txt ident
  → 이 설정을
  `.gitattributes`에 추가하면, Git은 `.txt` 파일에서 `$Id$`라는 키워드를 자동으로 SHA-1 체크섬 값으로 치환함

- 저장소 익스포트

  `export-ignore` : 아카이브를 만들 때 제외할 파일이나 디렉토리가 무엇인지 설정

  `export-subst` : Attribute로 설정한 파일들의 키워드가 치환됨

8.4

서버 훅

서버 정책은 전부 `update` 훅으로 만든다

- 커밋 메세지 규칙

`git rev-list` & `git cat-file` : Push된 커밋의 메시지를 검사하고, 규칙을 어긴 커밋은 Push할 수 없도록 함

- 파일 접근 권한

특정 사용자가 특정 디렉토리만 Push 할 수 있도록 ACL 파일을 사용해 권한을 관리함

`avail`로 지정된 사용자만 지정된 디렉토리나 파일에 대해 Push할 수 있도록 설정

`check_directory_perms` : 각 커밋에 수정된 파일을 확인, 사용자가 해당 파일을 Push할 권한이 있는지 확인

클라이언트 훅

서버 훅의 단점은 Push 할 때까지 Push 할 수 있는지 없는지 알 수 없다는 것

- 커밋 메세지 검증

`commit-msg` : 커밋 메시지가 규칙을 준수하는지 확인. 메시지에 패턴이 없으면 커밋을 거부

- 파일 접근 권한 확인

`pre-commit` : 로컬에서 커밋할 때 수정한 파일에 대한 권한을 확인. 사용자가 권한이 없는 파일을 수정했다면 커밋을 거부

- fast-forward push만 허용

`pre-rebase` : Rebase 중인 커밋이 이미 리모트 저장소에 존재하면 Rebase를 거부