Git 기초 - 실습 정리

date	@2023년 3월 1일
author	June
tags	Git Github

💡 다음 출처에서 실습용 코드와 더 상세한 설명을 제공하니 정확한 정보는 꼭 사이트에서 확인.

제대로 파는 Git & GitHub (무료 파트)

어려운 프로그래밍 개념들을 쉽게 설명해주는 유튜브 채널 '얄팍한 코딩사전'. 영상에서 다 알려주지 못한 정보들이나 자주 묻는 질문들의 답변들, 예제 코드들을 얄코에서 확인하세요!

https://www.yalco.kr/lectures/git-github/

제대로 파는 Git & GitHub (대학생 전체강의 반값할인)

#git #github #깃

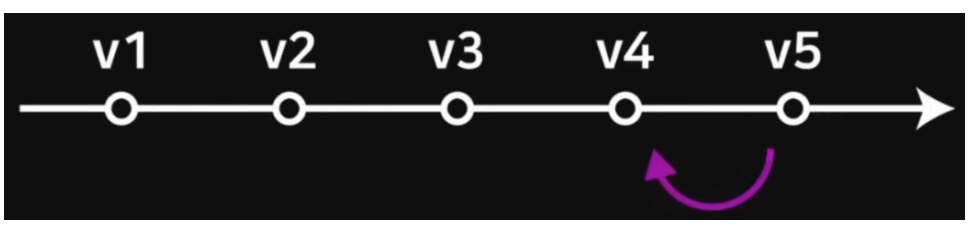
🎁 전체강의 30% 할인쿠폰 (모든분께!)

https://youtu.be/1l3hMwQU6GU?si=bYYXd_ROVnrOwiDd

Git의 기초

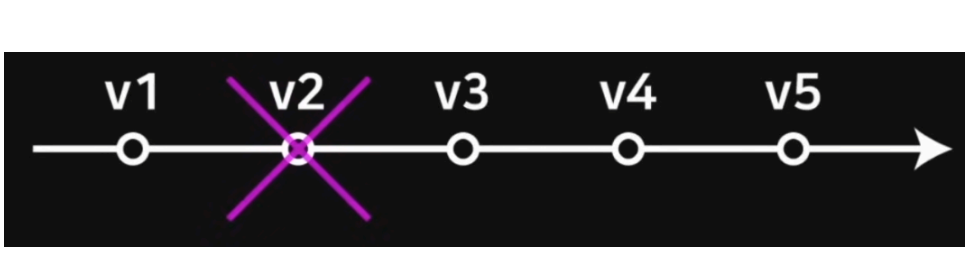
Git의 목표

1. 버전 관리



출처: 제대로 파는 Git & GitHub (대학생 전체강의 반값할인)

2. 일부 버전 파일 삭제



출처: 제대로 파는 Git & GitHub (대학생 전체강의 반값할인)

3. 여러 폴더에서 진행된 작업의 반영 결정

출처: 제대로 파는 Git & GitHub (대학생 전체강의 반값할인)

Windows Git 설치

(참고: https://www.yalco.kr/@git-github/1-2/)

- 설치 https://git-scm.com/

- Git client https://www.sourcetreeapp.com/
  - Registration : 건너뛰기
  - 도구 설치 없음 (체크 해제)
- VScode
  - CTRL + SHIFT + P
  - Select Default Profile
  - **Git Bash** 선택

CLI v.s. GUI

- CLI : terminal에서 사용 - 작업 중 사용
- GUI: sourcetree 사용 - 중간/최종 검토에 사용

Git의 최초 설정

- Git bash에서 사용자 등록 확인
  - `git config --global user.name`
  - `git config --global user.email`
- 등록된 사용자가 없는 경우 사용자 등
  - Git bash에서 사용자 등록 확인
    - `git config --global user.name "(본인 이름)"`
    - `git config --global user.email "(본인 이메일)”`
  - 이후 사용자 등록 확인
- 기본 브랜치명 변경
  - git config --global init.defaultBranch main

작업 폴더의 생성과 설정

- 작업 위치에서 폴더 생성 후 VScode로 open
- 터미널에서 `git init` 입력

  → 숨김 폴더 .git가 생성됨
  - 해당 폴더 내에 git 관리 내역이 저장되므로 폴더 삭제 시 프로젝트 과거 내역, branch 관리 불가
- 파일 생성 후 `git status` 입력

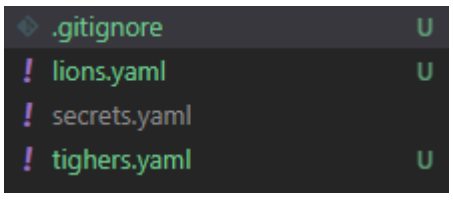
  → 붉은 글씨로 새로 생성한 파일이 표시됨

  = 명령어를 입력한 폴더에 존재하는 새로운 파일을 관리할지 보여줌

  ※ .git 폴더를 지우고 `git status`를 입력하면

  `fatal: not a git repository (or any of the parent directories): .git` 오류 발생

  이 경우, sourcetree에서 생성(Create)으로 연결해주면 .git 폴더가 생성됨 → cli에서 `git init` 입력하는 게 훨씬 편함
- Git으로 관리하지 않을 폴더/파일을 선택하는 방법
  - 자동 생성 파일들, 민감한 파일들은 제외해야 함
  - `.gitignore` 파일을 사용하여 배제할 요소 지정 가능
    - “.gitignore”이 파일명
    - 파일 명 옆에 U는 Untracked를 의미하며, Git이 추적 관리할 수 있는 대상 중 관리되지 않고 있는 파일을 의미함



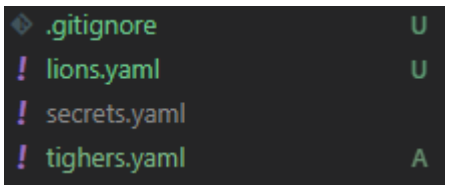
- 이후 `git status`를 입력하면 `.gitignore`에 추가한 파일은 나타나지 않음
- .gitignore 형식 https://git-scm.com/docs/gitignore 참조

```
#를 사용해서 주석

# 모든 file.c
file.c

# 최상위 폴더의 file.c
/file.c

# 모든 .c 확장자 파일
*.c

# .c 확장자지만 무시하지 않을 파일
!not_ignore_this.c

# logs란 이름의 파일 또는 폴더와 그 내용들
logs

# logs란 이름의 폴더와 그 내용들
logs/

# logs 폴더 바로 안의 debug.log와 .c 파일들
logs/debug.log
logs/*.c

# logs 폴더 바로 안, 또는 그 안의 다른 폴더(들) 안의 debug.log
logs/**/debug.log
```

Git의 사용

Commit

- commit = version
  - `git status`를 입력했을 때 아직 커밋이 없으면
    On branch main // No commits yet 이라고 나옴
  - Untracked file은 어떠한 버전에도 포함되지 않은
    git이 관리한 적이 없는 파일을 의미함
- `git add "파일명"`으로 git에 파일을 담을 수 있음
  - `git add .`를 사용하면 모든 파일이 담김
    - 일반적으로는 `add .`을 사용함
  - 파일이 git의 관리가 되기 시작하면 U→A()로 변경됨



  - 모든 파일을 git에 담고 `git status`를 확인하면
    Changes to be committed: new file: 의 목록이 나타남
- `git commit`은 별개의 지점으로서 새로운 version을 생성한다는 의미로 commit에 포함되는 파일들이 바르게 들어갔는지 등을 확인하기 위해 해당 명령어를 사용함
  - commit은 일부 파일을 복사해서 백업 버전을 관리하는 폴더 역할과 유사함

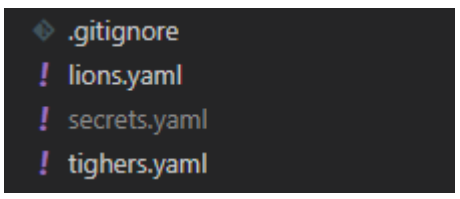
    - `git commit`을 입력하면 vi 편집기 화면으로 넘어감

      첫 번째 line 위치에서

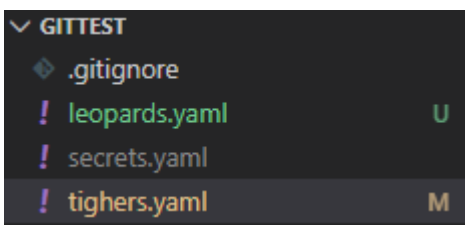
        - `i` : *입력*
        - `FIRST COMMIT` (또는 상황에 맞춰 commit 수정 설명 작성)
        - `[ESC]` : *입력 종료*
        - `:wq` : *저장하고 vi 종료* (상황에 따라 저장 없이 종료는 `:q` 또는 강제 종료 `:q!`)
- Commit을 완료하면 main 커밋이 생성되었다는 안내가 나오며,
  tag(해시값 상위 값), 입력한 commit명과 함께 어떤 파일들이 포함되었는지 함께 나타남

```
$ git commit
[main (root-commit) e53bb5f] FISRT COMMIT
 3 files changed, 17 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 lions.yaml
 create mode 100644 tighers.yaml
```

    - git status를 입력하면 main branch에 commit할 것이 없고
      작업 트리(working tree)가 clean하다고 나옴
    - 또한 commit을 완료하면 해당 시점을 기준으로 변경된 파일이 없으므로 vscode 파일 상태 A가 사라짐



- `git log`를 입력하면 commit의 해시값이 나옴
    - 해시 옆 (HEAD → main)은 해당 commit이 main branch에 접속해있고 가장 최근 commit임을 의미
        - 다른 branch로 이동하면 (HEAD → branchA, main)처럼 표시되고
        - 이 상태로 새로운 작업이 더 진행되면 (HEAD → branchA)라고 표시됨
        - 특정 과거 commit으로 이동하면 (HEAD)라고 나옴 ; Detached HEAD
        - `git log --oneline --graph --all`을 입력하면
          브랜치와 HEAD의 관계를 직관적으로 파악할 수 있음
    - 해시값과 함께 commit을 생성한 사람, 생성 일자과 커밋 명이 나타남
- 일반적으로 `git commit -m “commit 생성/수정/삭제 설명”`을 입력하여 한 번에 commit을 생성함 (설명은 작은 따옴표로 작성해도 됨)
    - 이후 `git log`를 확인하면 수정 내역이 나타남
    - `git commit -am “commit 생성/수정/삭제 설명”`을 이용하면 **add와 commit을 한번**에 수행할 수 있음
        - 단, untracked 파일이 없을 때만 가능함
        - 새로운 파일을 추가한 경우에는
          `git add .` 과 `git commit -m “*”`를 순서대로 수행해주어야 함
- 파일을 수정한 후 commit을 생성할 때
    - 파일을 수정하면 vscode 파일 상태가 M(modified)으로 변경됨



    - `git status`를 확인하면 삭제된 파일, 수정된 파일, git에 포함되지 않은 파일 목록이 나타남
    - `git diff`를 입력하면 기존 git이 관리하던 파일들의 변경 내역을 확인할 수 있음 (`:q`로 탈출)

과거 commit 확인

- Reset: 시간을 과거로 되돌림 = 되돌리기 위해 선택한 commit 이후 생성한 commit을 모두 삭제함
- Revert : inverse commit을 생성함
  - 커밋이 A → B → C 순으로 생성되었을 때, B로 돌아가고 싶어서 revert를 하면 A → B → C → -C (= B) 과정을 통해 B로 돌아옴
  - 이 방식은 법적 대응을 위한 수정 기록 관리 또는 팀의 협업(팀 협업 중 한 사용자가 reset하면 다른 사용자들의 작업이 충돌하는 문제가 생김)을 위해 수정 기록을 모두 남겨야 하는 경우와
  - 중간 commit 하나에서 수정한 내역만 삭제하길 바라고 이후 생성한 변경 내역은 유지하고 싶은 경우 사용함

Reset을 이용한 과거 commit으로의 이동

- `git reset --hard (commit hash)` 를 사용하여 (hash) 값을 가지는 커밋으로 이동
  - .gitignore에 추가해둔 파일은 영향을 받지 않음
- .git 폴더를 백업해두면 해당 폴더를 덮어씌움으로써 reset하기 전 시점으로 이동 가능
  - `git reset --hard` 로 해시를 입력하지 않으면 .git 폴더의 가장 최신 커밋으로 이동함
  - 이때 reset 전 존재했던 파일이 이동한 커밋에는 존재하지 않는 경우, 이 파일은 직접 삭제해야 함

Revert를 이용한 과거 commit으로의 이동

- `git revert (commit hash)` 를 사용하여 (hash) 값을 가지는 커밋으로 이동
  - 이를 입력하면 `git commit` 을 했을 때와 같은 화면으로 이동함
  - 단, commit message가 “Revert ~”로 입력되어 있음 (일반적으로 수정할 필요 없음)
  - :`wq`를 입력하고 commit log를 확인하면 새로운 commit이 생성된 것을 확인할 수 있음
    - Sourcetree로 확인하면 역 과정을 수행하고 있음을 확인 가능
- 한 커밋이 아닌 여러 커밋을 넘어가려는 경우, error가 발생할 수 있음 (git이 결정하지 못하는 case에 대한 오류 메시지 반환)
  - 직접 삭제해야 하는 파일이 존재할 경우

    `git rm (파일명)` 을 사용하여 해당 파일 삭제 후, `git revert --continue` 입력
- Revert한 커밋을 revert로 되돌리는 경우 (revert 전 시점의 commit으로 이동하려는 경우) commit 내역이 너무 많아진다는 문제가 있으므로, 이때는 reset을 이용하여 commit을 되돌림
- `git revert --no-commit (commit hash)` 를 사용하여 commit하지 않고 revert할 수 있음
  - commit 전인 `git add` 까지만 수행한 상태가 됨
  - 이 명령어는 다른 코드 수정까지 한 번에 커밋하려는 경우 사용함
  - commit하지 않고, `git reset --hard`를 입력하면 revert 및 add하려고 했던 내용들을 취소할 수 있음

Branch

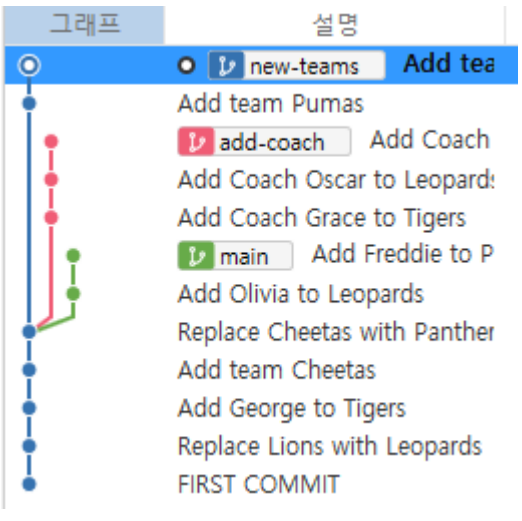
Branch 생성 (Branch, Switch)

- Branch를 사용하는 이유
  - main branch의 프로젝트를 배포용으로 관리할 때, 새로운 기능을 각 branch로 개발하고 완성되는 기능들은 빠르게 통합하여 update version을 제공함
  - main branch로 오픈소스 프로젝트를 개발할 때 공개 버전은 유지하면서 branch에서 새로운 기능을 작업하여 업데이트 프로젝트를 제공함
- Branch는 생성 후 이동하는 작업이 필요함
  - `git branch (branch name)` 을 사용하여 새로운 branch 생성
  - `git branch` 를 입력하면 생성되어 있는 branch 목록을 확인할 수 있음
    - 현재 작업 위치는 글씨가 다른 색으로 표시되고 branch 이름 앞에 * 가 표시됨
  - `git switch (branch name)` 을 입력하면 이동함
- `git switch -c (branch name)`을 사용하면 branch 생성과 switch가 한 번에 이뤄짐
- `git branch -d (삭제할 브랜치명)` 을 사용하면 branch를 삭제할 수 있음

    - 지우려는 branch에 아직 다른 branch로 적용하지 않고 남아있는 작업 커밋이 있는 경우 `git branch -D (삭제할 브랜치명)`를 이용하면 강제 삭제할 수 있음
- `git branch -m (변경 전 branch name) (변경 후 branch name)` 을 사용하면 branch name을 수정할 수 있음
    - m은 move를 의미함
    - `git log --all --decorate --oneline --graph` 를 이용하면 여러 브랜치 내역을 한 번에 볼 수 있음

```
$ git log --all --decorate --oneline --graph
* 8a5c6b5 (new-teams) Add team Jaguars
* 899df84 Add team Pumas
| * bd27da5 (add-coach) Add Coach Teddy to Panthers
| * 09a2cf0 Add Coach Oscar to Leopards
| * f35991e Add Coach Grace to Tigers
|/
| * 4c199b8 (HEAD -> main) Add Freddie to Panthers
| * e3c82ba Add Olivia to Leopards
|/
* 5c7dc2a Replace Cheetas with Panthers
* 3a82dcc Add team Cheetas
* f6f00bb Add George to Tigers
* d7856a5 Replace Lions with Leopards
* a294563 FIRST COMMIT
```

Sourcetree에서 refresh하여 확인해도 됨 (매번 확인할 때 더 편함)



Branch 병합 (Merge/Rebase)

- Merge는 서로 다른 두 branch를 붙이면서 새로운 commit을 하나 더 생성함
    - 두 branch에서 각각 작업한 내용이 모두 반영된 새로운 commit임
        - merge는 새로운 커밋을 생성하므로 reset으로 삭제하여 merge 전 시점으로 되돌릴 수 있음

$$A_1 - A_2 - A_3 - A_4 \qquad\qquad A_1 - A_2 - A_3 - A_4 - C_1$$
$$\qquad\qquad\quad \leftharpoondown B_1 - B_2 \quad \rightarrow \quad\qquad\quad \leftharpoondown B_1 - B_2 \rightharpoonup$$

    - `git switch A` 병합될(주로 main) branch에서 수행해야 함; *B를 끌어오는 방식*
    - `git merge B` 병합할 branch 이름을 입력
    - 병합한 branch는 `git branch -d B`로 삭제하여 branch 커밋 내역을 관리할 수 있음
        - 삭제된 branch는 graph에서 붉은 색으로 표시됨

```
$ git log --all --decorate --oneline --graph
*   122432c (HEAD -> main) Merge branch 'add-coach'
|\
| * bd27da5 Add Coach Teddy to Panthers
| * 09a2cf0 Add Coach Oscar to Leopards
| * f35991e Add Coach Grace to Tigers
* | 4c199b8 Add Freddie to Panthers
* | e3c82ba Add Olivia to Leopards
|/
| * 8a5c6b5 (new-teams) Add team Jaguars
| * 899df84 Add team Pumas
|/
* 5c7dc2a Replace Cheetas with Panthers
* 3a82dcc Add team Cheetas
* f6f00bb Add George to Tigers
* d7856a5 Replace Lions with Leopards
* a294563 FIRST COMMIT
```

- Rebase는 어떤 branch를 분기 시작점에서 잘라내어 다른 하나의 branch 끝에 이어 붙이는 방식임
  - branch를 날리는 것과 다름없으며, 다른 사람이 작업하던 branch에 덮어 씌우는 것이기 때문에 협업 중 오류가 발생할 가능성이 있음

$$\begin{matrix} A_1 - A_2 - A_3 - A_4 \\ \models B_1 - B_2 \end{matrix} \quad \rightarrow \quad A_1 - A_2 - A_3 - A_4 - B_1 - B_2$$

  - `git rebase B` 병합할 branch에서 수행해야 함; *B를 잘라 옮기는 방식*
  - `git rebase A` 병합될(주로 main) branch 이름을 입력
  - 병합한 branch는 잘려서 끝으로 이동함
    - ‘Add team Pumas’와 ‘Add team Jaguars’가 (main) 위로 붙음

```
$ git log --all --decorate --oneline --graph
* 9df829b (HEAD -> new-teams) Add team Jaguars
* 27fab68 Add team Pumas
*   122432c (main) Merge branch 'add-coach'
|\
| * bd27da5 Add Coach Teddy to Panthers
| * 09a2cf0 Add Coach Oscar to Leopards
| * f35991e Add Coach Grace to Tigers
* | 4c199b8 Add Freddie to Panthers
* | e3c82ba Add Olivia to Leopards
|/
* 5c7dc2a Replace Cheetas with Panthers
* 3a82dcc Add team Cheetas
* f6f00bb Add George to Tigers
* d7856a5 Replace Lions with Leopards
* a294563 FIRST COMMIT
```

  - A branch가 commit 최상단에 위치해야 함
    - `git switch A`로 이동 후
    - `git merge B`로 뒤쳐진 A branch를 올려줌

```
* 9df829b (HEAD -> main, new-teams) Add team Jaguars
* 27fab68 Add team Pumas
*   122432c Merge branch 'add-coach'
|\
| * bd27da5 Add Coach Teddy to Panthers
| * 09a2cf0 Add Coach Oscar to Leopards
| * f35991e Add Coach Grace to Tigers
* | 4c199b8 Add Freddie to Panthers
* | e3c82ba Add Olivia to Leopards
|/
* 5c7dc2a Replace Cheetas with Panthers
* 3a82dcc Add team Cheetas
* f6f00bb Add George to Tigers
* d7856a5 Replace Lions with Leopards
* a294563 FIRST COMMIT
```

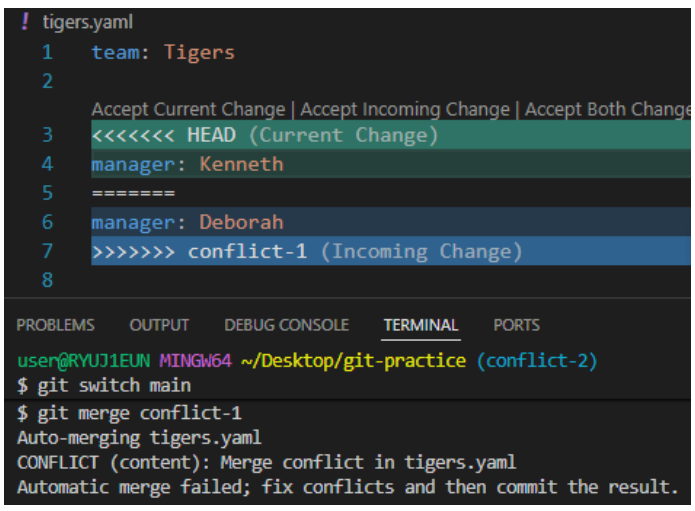
  - 병합한 branch는 `git branch -d B`로 삭제함
    - branch를 삭제하면 graph 최상단 HEAD가 남은 branch만을 가리키게 됨
      → merge commit을 생성하지 않고 HEAD pointer만 이동시킴 = Fast-Forward (branch가 하나이면 merge해도 새로운 commit을 생성하지 않음)

```
$ git log --all --decorate --oneline --graph
* 9df829b (HEAD -> main) Add team Jaguars
* 27fab68 Add team Pumas
*   122432c Merge branch 'add-coach'
|\
| * bd27da5 Add Coach Teddy to Panthers
| * 09a2cf0 Add Coach Oscar to Leopards
| * f35991e Add Coach Grace to Tigers
* | 4c199b8 Add Freddie to Panthers
* | e3c82ba Add Olivia to Leopards
|/
* 5c7dc2a Replace Cheetas with Panthers
* 3a82dcc Add team Cheetas
* f6f00bb Add George to Tigers
* d7856a5 Replace Lions with Leopards
* a294563 FIRST COMMIT
```

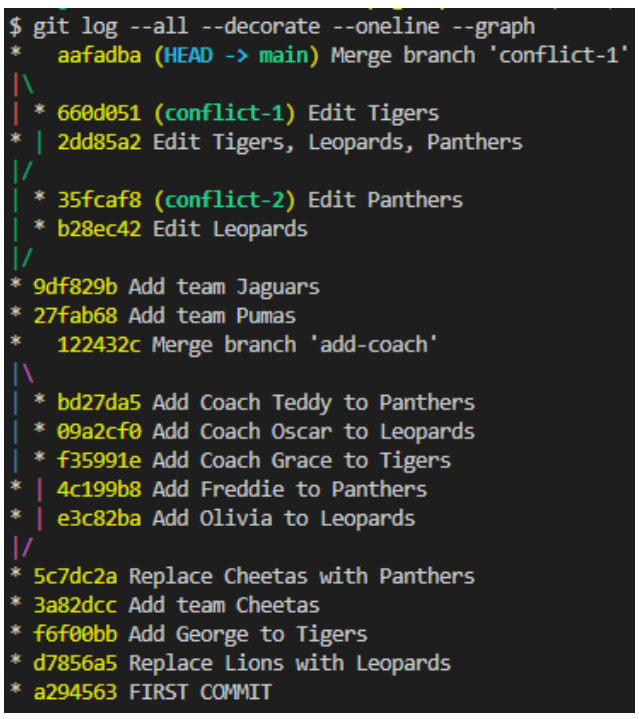
- 두 방식의 최종 commit에서는 결과물이 같지만, commit 내역이 다름
  - commit log 남겨야 하는 경우 merge를 사용
  - branch가 매우 많아 commit 내역을 줄여야 하는 경우 rebase를 사용

Branch 충돌 해결

- 같은 파일 같은 line에 서로 다른 작업을 수행하면 병합 과정에서 충돌이 발생함
  - `git status`로 문제가 발생한 파일을 확인할 수 있음
- Merge 과정에 오류가 발생한 경우 vscode에서 현재 상태와 merge하려는 커밋의 상태를 색을 칠해서 보여줌 (다른 편집기에도 충돌 메시지는 표시됨)
  - 어떤 작업을 반영할지는 두 작업 내역 위에 표시된 VScode에서 제공하는 옵션 중 하나를 선택하면 자동으로 반영됨



  - 작업을 선택하면 이후 `git add .`와 `git commit`으로 병합을 완료함
    - 이때 commit의 이름은 자동으로 설정되어 있으므로, `:wq`로 탈출



  - 충돌이 너무 많아서 특정 작업을 선택할 수 없을 때는 `git merge --abort`를 입력하면 merge를 취소하고 이전 상태로 돌아감
- Merge는 새로운 commit을 생성하는 방식이기 때문에 충돌이 commit을 생성하는 방식으로 끝남
- Rebase는 작업을 옮기는 과정이므로 각 충돌을 해결하고 옮기는 작업을 수행해야 함
  - Rebase를 취소하려는 경우 `git rebase --abort`를 입력함
  - 충돌을 해결 가능한 경우, 수정 후 `git add .`와 `git rebase --continue` 입력
    - `git rebase --continue`를 입력하면 나타나는 창에서는 `:wq` 입력
    - 충돌이 모두 수정될 때까지 진행
  - 수정 작업 중 incoming change를 무시하고 current change를 선택하면 branch commit이 추가되지 않고 사라짐
    - 아래 예시에서는 merge branch ‘conflict-1’ 이후 ‘conflict-2’의 작업이 2개 붙어야 했지만, 하나는 반영하지 않아서 commit이 하나만 옮겨짐

```
$ git log --all --decorate --oneline --graph
* e2ca109 (HEAD -> main) Edit Leopards
*   aafadba Merge branch 'conflict-1'
|\
| * 660d051 Edit Tigers
* | 2dd85a2 Edit Tigers, Leopards, Panthers
|/
* 9df829b Add team Jaguars
* 27fab68 Add team Pumas
*   122432c Merge branch 'add-coach'
|\
| * bd27da5 Add Coach Teddy to Panthers
| * 09a2cf0 Add Coach Oscar to Leopards
| * f35991e Add Coach Grace to Tigers
* | 4c199b8 Add Freddie to Panthers
* | e3c82ba Add Olivia to Leopards
|/
* 5c7dc2a Replace Cheetas with Panthers
* 3a82dcc Add team Cheetas
* f6f00bb Add George to Tigers
* d7856a5 Replace Lions with Leopards
* a294563 FIRST COMMIT
```

GitHub

Personal access token

- GitHub에 '프로젝트'를 업로드하기 위해 필요한 token
  - 프로필을 누르면 나오는 Settings 페이지 좌측 최하단의 Developer Settings 선택
  - Personal access tokens를 클릭한 후 Tokens (classic)을 선택
  - Generate new token으로 새로운 토큰 생성
    - 토큰 이름
    - 만료기한
    - 허용 권한 범위
  - 생성한 토큰은 페이지를 나가면 다시 볼 수 없으므로 메모장에 복사해둘 것
    - 자격 증명 관리에 git:https://github.com를 추가하면 토큰을 매번 입력할 필요 없음
    - 사용자 이름은 프로필에 옅은 회식으로 뜨는 이름
- GitHub에서 새로운 repository 생성
  - Settings에서 Collaborators를 추가할 수 있음

프로젝트 최초 업로드

- Code 페이지에서 새로운 프로젝트를 생성할 것인지 기존 프로젝트를 업로드할 것인지 선택하여 코드 내용 복사
  - VScode에서 프로젝트 폴더에 들어가 복사한 코드를 붙여넣음

```
// orign은 업로드한 프로젝트가 저장될 원격 저장소 이름을 의미함
// 즉, origin as github 원격 저장소
// 저장소 이름은 다르게 설정해도 됨
git remote add origin (github 원격 저장소 주소)

// 기본 branch name을 main으로 수정
git branch -M main

// 로컬 저장소에 있는 commit 중
// 업로드되지 않은 commit을 origin의 main branch에 업로드
git push -u origin main
```

  - GitHub Code 페이지를 새로고침하면 프로젝트가 업로드된 것을 확인할 수 있음
  - VScode에서 `git remote`를 입력하면 현재 프로젝트와 연결된 원격 저장소가 나음
    - `git remote -v`를 입력하면 더 자세한 내용을 볼 수 있음

프로젝트 다운로드

- GitHub에서 찾은 코드의 [<>Code] 버튼에서 Clone을 위한 HTTPS 주소를 복사함
- 프로젝트를 다운받길 원하는 폴더 위치에서 git bash를 엶
- `git clone (복사한 주소)`를 입력함
- Cloning이 완료되면 VScode에서 코드를 사용하거나 `git log`로 commit 내역을 확인할 수 있음

Push & Pull

- Push: 최초 업로드 후 수정한 파일이 있을 때 새롭게 생성한 commit을 GitHub에 업로드하는 방법
  - Push하려는 commit 위치에서 `git push` 입력
    - 처음 프로젝트를 생성할 때, push할 기본 branch를 미리 `git push -u origin main`으로 지정했기 때문에 별도 위치 지정 없이 push 가능
- Pull: 로컬 저장소에는 없는데, GitHub에는 있는 코드를 다운받는 방법
  - 변경된 파일이 있을 경우 `git pull` 입력
  - Pull할 파일이 있는데, Push하는 경우
    - `git push`를 하면 오류 발생 : push를 하려면 원격 저장소 최신 내역이 반영되어 있어야 함
    - 이에 따라 pull을 하려고 할 때, 만약 로컬 저장소에 생성해둔 commit이 있을 경우
      1. pull의 옵션을 merge로 선택

         `git pull --no-rebase` : 로컬의 main branch와 원격의 main branch를 별개로 보고 새로운 commit으로 합침
      2. pull의 옵션을 rebase로 선택 (협업 중 push는 rebase로 하면 안되지만, pull은 괜찮음)

         `git pull --rebase` : 원격 branch와 history를 맞추기 위해 github 커밋을 먼저 맞추고 로컬 커밋을 뒤로 추가함
         - 같은 줄에서 충돌하는 경우 오류 위치에서 current/incoming change 선택은 코드 결과 뿐 아니라 커밋 수에 영향을 줌
           - current change를 선택하면, `git add .`, `git rebase --continue`를 입력했을 때 추가 확인 없이 병합이 종료됨
           - incomming change를 선택하면, 새로운 commit을 생성하고 pull data가 추가되며, 이전 로컬 수정 내역은 commit으로 남게 됨
      3. (협업 시 합의 없이 절대 X) 로컬 내역이 원격보다 뒤쳐졌지만 강제 push하고 싶은 경우

         `git push --force`
  - Pull이 완료된 후 Push 진행

원격의 branch 다루기

- main이 아닌 branch를 생성하고 push를 하면 오류 메시지가 나타남
  - 새롭게 생성한 branch를 어디로 push해야 하는지 모르는 상태
  - `git push -u origin (branch name)` 명령 또는 `git push --set-upstream origin (branch name)`으로 원격 branch 명시 및 기본 설정
  - `git branch --all` 또는 `git branch --a` 명령으로 원격에 존재하는 branch를 확인할 수 있음
    - 원격의 branch 목록은 “remotes/~”로 시작함
- 원격 저장소에만 존재하는 팀원에 의해 새로 추가된 branch를 pull하고 싶은 경우 (`git branch --a` 을 입력해도 확인할 수 없음)
  - `git fetch` 명령을 입력하고
  - `git switch -t origin/(새로운 원격 branch name)` 명령을 입력하여 branch를 연결함
  - 이후 `git switch (새로운 원격 branch name)`으로 branch를 이동한 후
  - `git pull` 수행
- 원격 저장소의 branch를 삭제하고 싶은 경우 `git push (원격 이름; origin) --delete (원격의 브랜치명)` 명령을 입력

※ 추가로 공부할 내용 : Commit에 tag다는 방법