#git #vcs #coding

섹션 0. Introduction

1강 강의 소개
2강 강의를 듣는 방법

섹션 1. Git 시작하기 (config, init)

S1-Lesson 1 Git을 배워야 하는 이유

VCS 영상 링크
Git : Version control system 중의 하나
프로젝트의 시간과 차원을 관리하는 Tool

S1-Lesson 2. 강의를 위한 설치와 세팅 (윈도우)

dlp system

- End Point Yum 저장소 추가
  `dlpsudo yum -y install wandisco-git-release-7-1.noarch.rpm`
- 기존 Git 제거
  `dlpsudo yum remove git`
- Git 설치
  `dlpsudo yum -y install git`
- Git 버전 확인
  `git --version`

1. https://git-scm.com/ 로 이동해서 Git을 다운로드 한다.
   설치과정 중 *Git Bash* 를 반드시 설치
   - Git 사용에 적합한 터미널
   - 리눅스/맥(유닉스)에서 사용되는 CLI 명령어들을 윈도우에서 사용 가능 - 타 프로그래밍에도 유용
   - 기본 설정된 그대로 설치를 진행하시면 됩니다.

설치 후 Git Bash에서 아래 명령어로 테스트 해본다.

```
git --version
```

협업 시 윈도우와 맥의 엔터 방식 차이로 인한 오류를 방지

```
git config --global core.autocrlf true
```

2. SourceTree 설치
   https://www.sourcetreeapp.com/ - Git을 GUI로 다룰 수 있도록 해주는 툴
   git bash 에서 global 세팅

```
git config –global –add safe.directory ‘*’
```

3. VS Code 설치
   https://code.visualstudio.com/ - IDE 프로그램
   (윈도우의 경우)

- `Ctrl + ` ` 를 이용하여 터미널을 열 수 있다.
- `Ctrl + Shift + P` 로 "Terminal: select default Profile"을 선택하고 Git bash 선택
- `+` 를 눌러서 새로운 터미널을 열면 Prompt가 바뀐다.
- VS Code에서 ID, PW 설정을 자동 저장

```
git config –global credential.helper store
```

S1-Lesson 3. Git 설정 & 프로젝트 관리 시작

VS Code의 폴더 및 아이콘 바꾸기 Material Icon Theme
Download VSIX file —> Install

1. Git 전역으로 사용자 이름과 이메일 주소를 설정
   GitHub 계정과는 별개
   터미널 프로그램 (Git Bash, iTerm2)에서 아래 명령어 실행

```
git config --global user.name "(본인 이름)"
```

```
git config --global user.email "(본인 이메일)"
```

아래의 명령어들로 확인

```
git config --global user.name
```

```
git config --global user.email
```

Default branch 이름 바꾸기 (master—> main)

```
git config --global init.defaultBranch main
```

2. 프로젝트 생성 및 Git 관리 시작
   적당한 위치에서 `code ./` 으로 VS Code를 열고 해당 폴더에서

```
git init
```

폴더 내 `.git` 숨김 폴더 생성 확인

tigers.yaml 파일 생성

```
team: Tigers

manager: John

members:
- Linda
- William
- David
```

lions.yaml 파일 생성

```
team: Lions

manager: Mary

members:
- Thomas
- Karen
- Margaret
```

현재 상태 체크

```
git status
```

3. 소스트리로 해당 폴더 열고 살펴보기

S1-Lesson 4. Git에서 관리하지 말아야 할 것들

- 보안상 민감한 정보

- 자동으로 생성 및 다운로드 되는 파일들
- gitignore 공식문서

secret.yaml 파일생성

```
ID: choiykyo
PW: 1234
```

명령어로 상태 확인

`.gitignore` 파일 생성
해당 파일 안에 secret.yaml 등록

```
# 이렇게 #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
```

Django 나 Nodejs의 .gitignore 확인 gitignore.io

섹션 2. 시간 넘나들기 (add, commit, reset, reverse)

S2-Lesson 1. Commit: 변화를 타임캡슐에 담아 묻기

Commit = 버전
윈도우 소스트리에서 상태가 바로 업데이트되어 보이지 않는 경우 F5를 눌러 새로고침

1. 프로젝트 변경사항들을 타임캡슐(버전)에 담기
   변경사항 확인

```
git status
```

추적하지 않는 (untracked)파일: Git의 관리에 들어간 적 없는 파일

파일 하나 담기 (타임캡슐에 넣기)

```
git add tigers.yaml
```

`git status`로 확인

모든 파일 담기

```
git add .
```

`git status`로 확인

2. 타임캡슐에 묻기 (버전 관리에 담기)

```
git commit
```

vi editor로 진입 - vi 명령어

작업	Vi 명령어	상세
입력 시작	**i**	명령어 입력 모드에서 텍스트 입력 모드로 전환
입력 종료	`ESC`	텍스트 입력 모드에서 명령어 입력 모드로 전환
저장 없이 종료	**:q**	
저장 없이 강제 종료	**:q!**	입력한 것이 있을 때 사용
저장하고 종료	**:wq**	입력한 것이 있을 때 사용
위로 스크롤	**k**	`git log` 등에서 내역이 길 때 사용
아래로 스크롤	**j**	`git log` 등에서 내역이 길 때 사용

커밋 메세지까지 함께 작성하기

```
git commit -m "This is First Commit"
```

아래 명령어와 소스트리로 확인

```
git log
```

3. 변경사항 만들고 타임캡슐에 묻기
   변경사항

   - `lions.yaml` 파일 삭제
   - `tigers.yaml`의 manager를 `Donald`로 변경
   - `leopards.yaml`파일 추가

```
team: Leopards

manager: Luke

members:
- Linda
- William
- David
```

   `git status`로 확인

   - 파일 추가, 변경, 삭제 모두 내역으로 저장할 대상
   - `git diff` 로 확인

   캡슐에 담기

```
git add .
```

`git status` 로 확인

타임캡슐 묻기

```
git commit -m "Replace Lions with Leopards"
```

TIP add와 commit을 한번에

```
git commit -am "message"
```

단, 새로 추가된 Untracked file이 없을 경우만!

4. 새로운 변화를 만들고 추가 커밋 만들기
   첫 번째 추가 커밋

- `tigers.yaml`의 `members`에 `George`추가
- 커밋 메세지: `Add George to Tigers`

두 번째 추가 커밋

- `cheetas.yaml` 파일의 추가

```
team: Cheetas

manager: Laura

members:
- Ryan
- Anna
- Justin
```

- 커밋 메세지: Add team Cheetas

세 번째 추가 커밋

- `cheetas.yaml` 삭제
- Leopards의 `manager`를 `Nora`로 수정
- `panthers.yaml 추가

```
team: Panthers

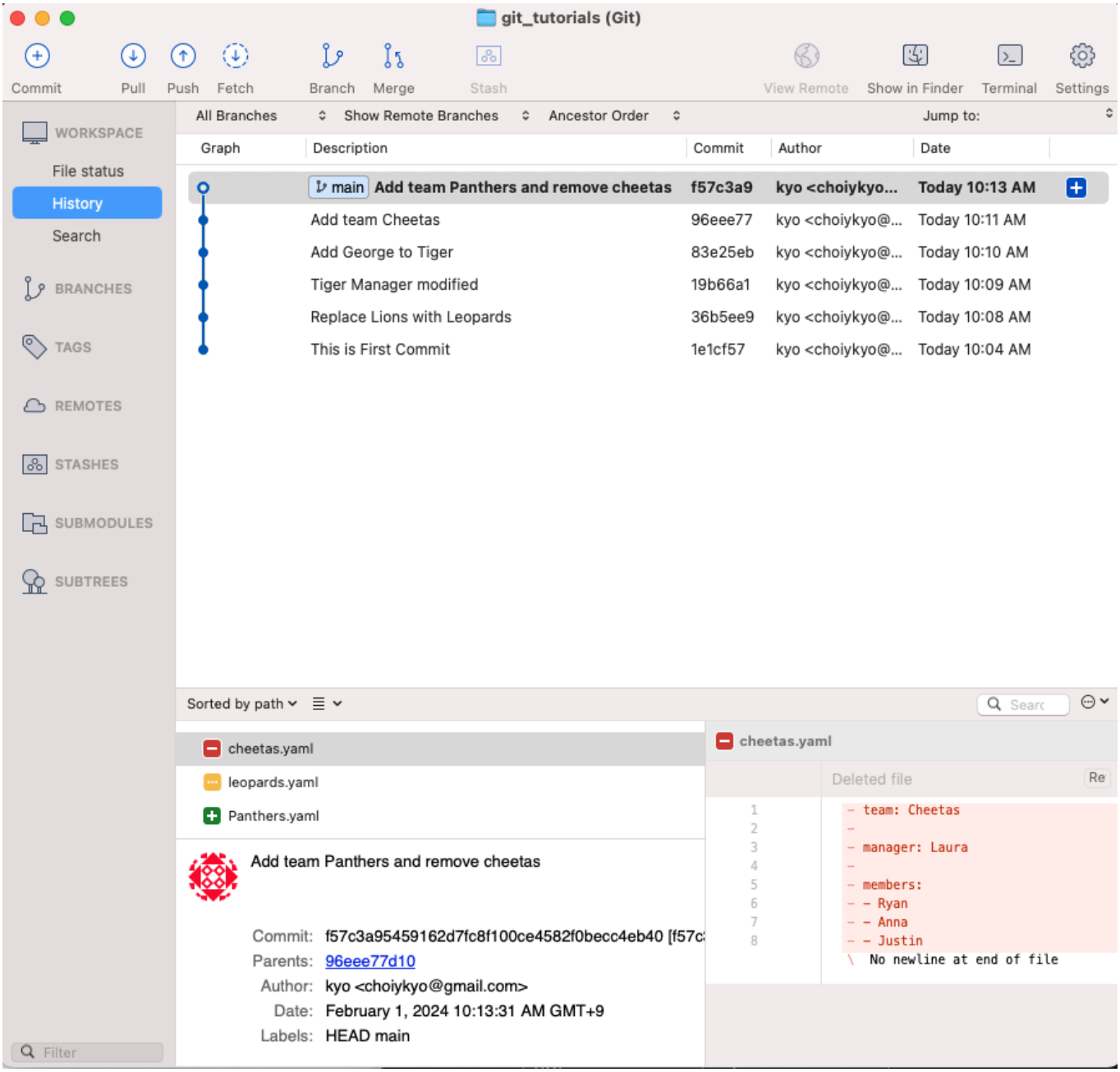
manager: Sebastian

members:
- Violet
- Stella
- Anthony
```

커밋 메세지 "Add team Panthers and remove cheetas"

`git log` 로 history 확인 / 긴 내역일 경우 `j` `k` 또는 위/아래 방향키로 내역 확인

소스트리 결과 확인



S2-Lesson 2. 과거로 돌아가기

Reset vs Revert
reset : 원하는 시점으로 돌아간 뒤 이후 내역들을 지웁니다
revert : 되돌리기 원하는 시점의 커밋을 거꾸로 실행합니다.
linux open current directory

```
nautilus ./
```

1. 실습 전 내역 백업

`.git` 폴더 복사해두기
`.git` 폴더 없앤 다음 git 상태 확인해보기

2. reset 사용해서 과거로 돌아가기

아래 명령어로 커밋 내역 확인
`git log`
되돌아갈 시점: Add team Cheetas의 커밋 해시 복사
`:q` 로 빠져나가기

```
git reset --hard (되돌아갈 커밋 해시)
```

첫 커밋 시점으로 돌아가보기

3. reset 하기 전 시점으로 복원해보기

백업해 둔 `.git` 폴더 사용
- `.git` 폴더 복원
- `git log`, `git status` 로 상태 확인
- 아래 명령어로 현 커밋 상태로 초기화
`git reset --hard - 뒤에 커밋 해시가 없으면 마지막 커밋을 가리킴 -` lions.yaml` 삭제
실제 업무에서는 절!!대!!로!! .git을 편집하거나 지우지 않는다

4. revert 로 과거의 커밋 되돌리기

A. `Add George to Tigers` 의 커밋 해시 구하기
아래 명령어로 **revert**
`bash git revert (되돌릴 커밋 해시)`
`:wp` 로 커밋 메세지 저장

B. `Replace Lions with Leopards` 의 커밋 되돌려 보기
- 이후 `leopards.yaml` 수정 내역 때문에 충돌 발생
- `git rm leopards.yaml` 로 Git에서 해당 파일 삭제
- `git revert --continue` 로 마무리
- `:wp` 로 커밋 메세지 저장

C. **reset** 사용해서 revert 전으로 되돌아가기

D. **commit 하지 않고** revert하기

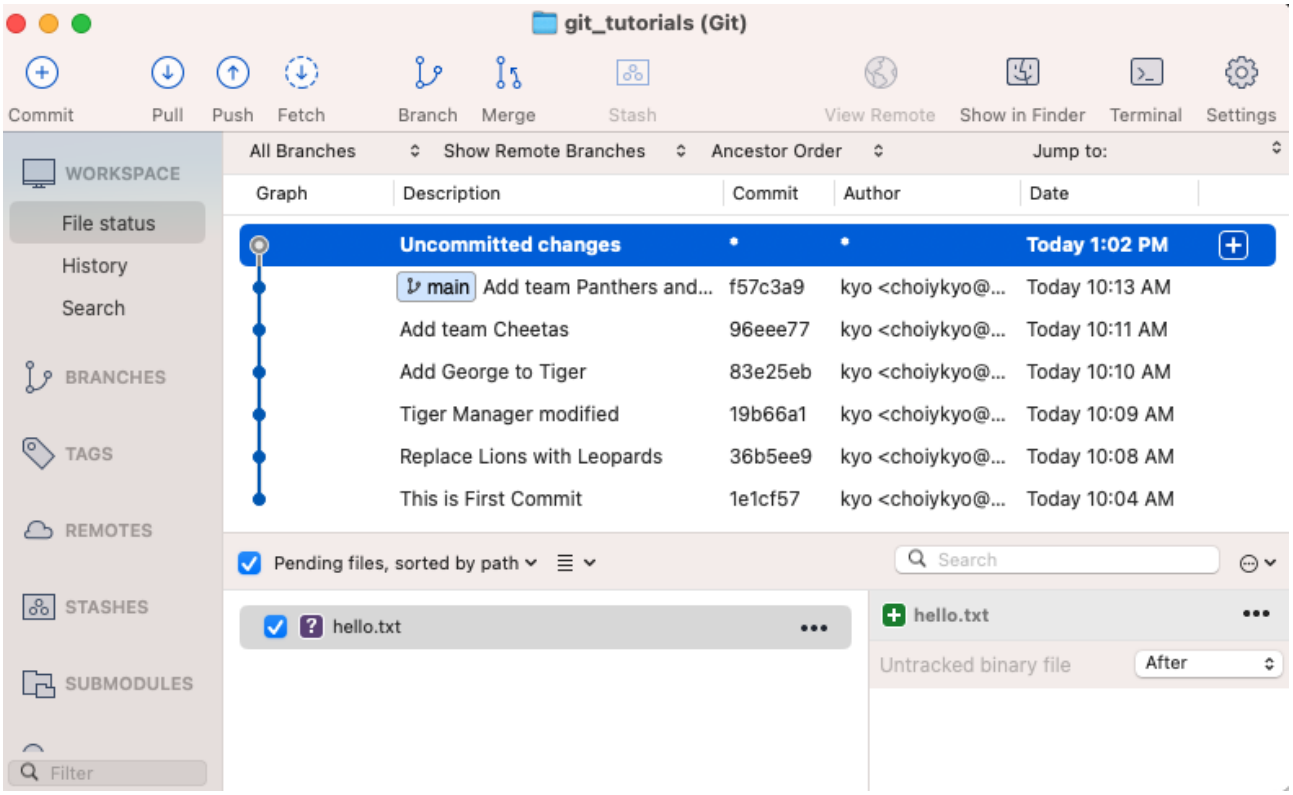
```
git revert --no-commit (되돌릴 커밋 해시)
```

- 원하는 다른 작업을 추가한 다음 함께 커밋
- 취소하려면 `git reset --hard`

S2-Lesson 3. SourceTree 활용하기

1. 변경사항 만들고 커밋하기

- `leopards.yaml` 삭제
- `.gitignore` 에 `*.config` 추가
- `hello.txt` 파일 추가 및 내용 작성
- 커밋 메세지 `Commit from SourceTree`

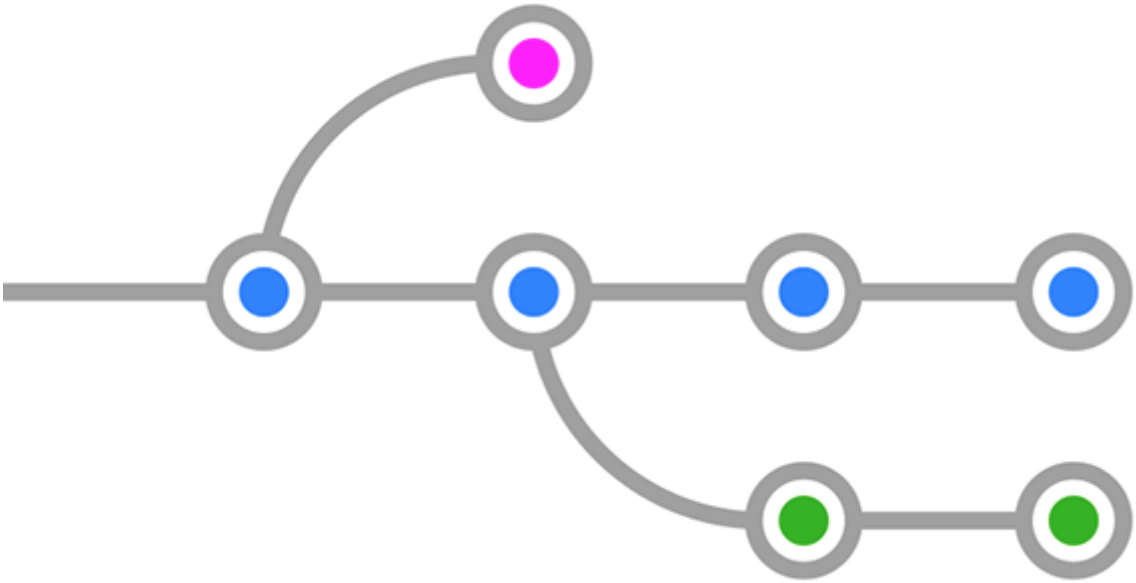


윈도우와 맥이 조금 다른데 맥의 경우 체크박스로 스테이지에 올리고 윈도우는 선택하여 스테이지에 올림

섹션 3. 차원 넘나들기 (branch, merge)

S3-Lesson 1. 여러 branch 만들어보기

Branch : 분기된 가지 (서로 다른 차원)



- 프로젝트를 하나 이상의 모습으로 관리해야 할 때
  (ex: 실배포용, 테스트서버용 등)
- 여러 작업들이 각각 독립되어 진행될 때
  (새로운 기능 1, 새로운 기능 2, 코드 개선, 긴급 수정 등)
  각각의 branch(차원)에서 작업한 뒤 확정된 것을 메인 차원에 통합

이 모든 것을 하나의 프로젝트 폴더에서 진행할 수 있도록 Branch를 활용

1. 브랜치 생성 / 이동 / 삭제하기

A. `add-coach` 란 이름의 브랜치 생성

```
git branch add-coach
```

B. 브랜치 목록 확인

```
git branch
```

`add-coach` 브랜치로 이동

```
git switch add-coach
```

`checkout` 명령어가 Git 2.23 버전부터 `switch`, `restore` 로 분리되었음.

C. 브랜치 생성과 동시에 이동하기

```
git switch -c new_team
```

-c option (--create)

D. 브랜치 삭제하기

```
git branch -d (삭제할 브랜치 이름)
```

`to-delete` 라는 이름의 브랜치 생성 후 삭제해보기

TIP: 지워질 브랜치에만 있는 내용이 있는 경우 -D로 강제삭제 해야한다.

```
git branch -D (삭제할 브랜치 이름)
```

E. 브랜치 이름 바꾸기

```
git branch -m (기존 브랜치 이름) (새 브랜치 이름)
```

-m option (--move)

2. 각각의 브랜치에서 서로 다른 작업 진행하기

A. `main` 브랜치

1. `leopards.yaml` 의 `members` 에 `Olivia` 추가
   커밋 메세지: `Add Olivia to Leopards`
2. `panthers.yaml` 의 `members` 에 `Fredy` 추가
   커밋 메세지: `Add Fredy to Panthers`
   `add-coach` 브랜치로 이동하여 해당 코드를 확인

B. `add-coach` 브랜치

1. `tigers.yaml` 의 `manager` 정보 아래에 `coach: Grace` 추가
   커밋 메세지: `Add Coach Grace to Tigers`
2. `leopards.yaml` 의 `manager` 정보 아래에 `coach: Oscar` 추가
   커밋 메세지: `Add Coach Oscar to Leopards`
3. `panthers.yaml` 의 `manager` 정보 아래에 `coach: Teddy` 추가
   커밋 메세지: `Add Coach Teddy to panthers`

C. `new-teams` 브랜치

1. `pumas.yaml` 추가
   커밋 메세지: `Add team Pumas`

```
team: Pumas

manager: Jude

members:
- Ezra
- Carter
- Finn
```

2. `jaguars.yaml` 추가
   커밋 메세지: `Add team Jaguars`

```
team: Jaguars

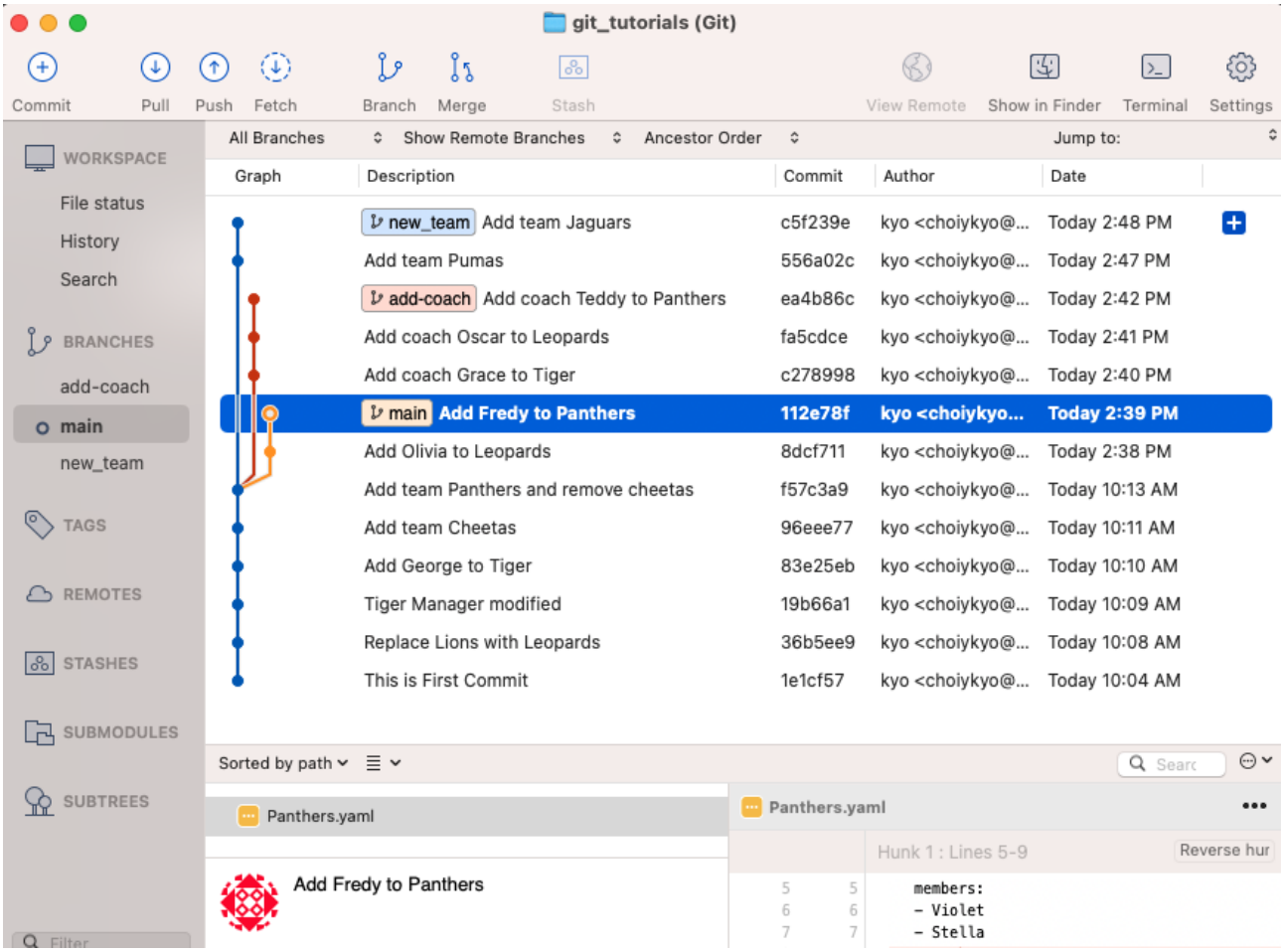
manager: Stanley

members:
- Caleb
- Harvey
- Myles
```

3. 결과 살펴보기
   `git log` : 위치한 브랜치의 내역만 확인 가능
   여러 브랜치의 내역 확인하기

```
git log --all --decorate --oneline --graph
```

소스트리에서 확인



S3-Lesson 2. Branch를 합치는 두 가지 방법

Merge vs **Rebase**

- Merge : 두 브랜치를 한 커밋에 이어붙인다.
  - 브랜치 사용 내역을 남길 필요가 있을 때 적합한 방식
  - 다른 형태의 merge 방식도 존재함
- Rebase : 브랜치를 다른 브랜치에 이어붙인다.
  - 한 줄로 깔끔히 정리된 내역을 유지할 때 적합한 방식
  - 이미 팀원과 공유된 커밋들에서는 사용 금지

---

1. merge로 합치기

`add-coach` 브랜치를 `main` 브랜치로 **merge**

- `main` 브랜치로 이동
- 아래의 명령어로 병합

```
git merge add-coach
```

- `:wp` 로 자동입력된 커밋 내용 저장하여 종료
- 소스트리에서 확인

`merge` 는 `reset` 으로 되돌리기 가능

- `merge` 도 하나의 커밋
- `merge` 하기 전 해당 브랜치의 마지막 시점으로

병합된 브랜치는 삭제
삭제 전 소스트리에서 `add-coach` 위치 확인

```
git branch -d add-coach
```

2. rebase로 합치기

`new-teams` 브랜치를 `main` 브랜치로 rebase

- `new-teams` 브랜치로 이동
  merge와는 반대!!!
- 아래의 명령어로 병합

```
git rebase main
```

- 소스트리에서 상태 확인
  `main` 브랜치는 뒤쳐져 있는 상황 확인
- `main` 브랜치로 이동 후 아래 명령어로 `new-teams` 의 시점으로 **fast-forward**

```
git merge new-teams
```

- `new-teams` 브랜치 삭제

S3-Lesson 3. 충돌 해결

충돌(conflict) : 한 파일의 같은 위치에 다른 내용이 입력된 상황

1. 충돌 상황 만들기

A. `conflict-1` , `conflict-2` 브랜치 생성
B. `main` 브랜치

- Tigers의 manager를 Kenneth로 변경
- Leopards의 coach를 Nicholas로 변경
- Panthers의 coach를 Shirley로 변경
- 커밋 내용 : `Edit Tigers, Leopards, Panthers`
  C. `conflict-1` 브랜치
- Tigers의 manager를 Deborah 로 변경
- 커밋 내용 : `Edit Tigers`
  D. `conflict-2` 브랜치 1st
- Leopards의 coach를 Melissa로 변경
- 커밋 내용 : `Edit Leopards`
  E. `conflict-2` 브랜치 2nd
- Panthers의 coach를 Raymond로 변경
- 커밋 내용 : `Edit Panthers`

2. `merge` 충돌 해결하기

`git merge conflict-1` 로 병합을 시도하면 충돌 발생

- 오류 메세지와 `git status` 확인
- VS Code에서 해당 부분 확인

당장 충돌 해결이 어려울 경우 아래 명령어로 `merge` 중단

```
git merge --abort
```

해결 가능 시 충돌 부분을 수정한 뒤 `git add .`, `git commit` 으로 병합 완료

3. `rebase` 충돌 해결하기

`conflict-2` 에서 `git rebase main` 로 리베이스 시도 시 충돌 발생

- 오류 내용과 `git status` 확인
- VS Code에서 해당 내용 확인
  당장 충돌 해결이 어려울 경우 아래 명령어로 `rebase` 중단

```
git rebase --abort
```

해결 가능 시

- 충돌 부분을 수정한 뒤 `git add .`
- 아래 명령어로 계속 진행

```
git rebase --continue
```

- 충돌이 모두 해결될 때까지 반복
  `main`에서 `git merge conflict-2`로 마무리

rebase로 2칸까지 conflict-2를 병합했는데 왜 새로운 노드는 1개인가?

`conflict-1`, `conflict-2` 삭제

섹션 4 Git & Github (GitLab)

S4-Lesson 1. Git 원격 저장소 만들기

Git으로 관리되고 있는 프로젝트의 원격 저장소
(Git, VS Code, Tensorflow의 Git 저장소 확인)

1. GitHub 저장소 만들기

A. `Sing Up`으로 가입 후 로그인
B. *Personal access token* 만들기

- 우측 상단의 profile - Settings
- Developer Settings
- Personal access tokens - Generate new token
- repo 및 원하는 기능에 체크, 기간 설정 뒤 Generate token
- token은 안전한 곳에 보관

C. Token 컴퓨터에 저장하기

- 윈도우 가이드
  - Windows 자격 증명 관리자
  - Windows 자격 증명 선택

    - git:https://github.com 자격 정보 생성
    - 사용자명과 토큰 붙여 넣기
- 맥 가이드
    - Keychain Access 앱 실행
    - github의 인터넷 암호 항목 선택
    - 사용자명 (계정 칸)과 토큰 (암호보기 누른 뒤 오른쪽 칸) 붙여넣기
    - 키체인 관련 팝업이 나오면 맥 로그인 암호 입력

D. Github에 새 Repository 생성

- public : 모두에게 보일 수 있는 프로젝트
- private : 허용된 인원만 볼 수 있는 프로젝트

E. 협업할 팀원 추가

- 리포지토리의 Settings - Collaborators
- Add people

S4-Lesson 2. 원격 저장소 사용하기

1. Github 레포지토리 연결하기

```
git remote add origin (원격 저장소 주소)
```

- 로컬의 Git 저장소에 원격 저장소로의 연결 추가
- 원격 저장소 이름에 흔히 `origin` 사용 / 다른 것으로 수정 가능

```
git branch -M main
```

- Github 권장 - 기본 브랜치 명을 main으로

```
git push -u origin main
```

로컬 저장소의 커밋 내역을 원격으로 push (업로드)

- **`git push`** : 로컬 브랜치의 커밋을 원격저장소로 업로드
- **`-u`** 또는 **`--set-upstream`** : 다음 push 에서는 `git push` 명령어로 바로 올릴 수 있다.
- **`-f`** : 강제 옵션

- `origin` : 원격저장소의 default 이름
- `main` : push 하고자 하는 로컬 브랜치 이름
  `git remote` 명령으로 현재 프로젝트에 등록된 리모트 저장소를 확인할 수 있음.
  `git remote -v` 연결된 리모트 저장소 상세 내역 확인 가능.
  `git remote add <단축이름> <url>` 로 워킹 디렉토리에 새 리모트 저장소 추가

원격 연결 지우기 (로컬 프로젝트와의 연결만 삭제)

```
git remote remove (origin 등 원격 이름)
```

2. GitHub에서 프로젝트 다운받기

- Download ZIP : 파일만 다운받음, Git 관리 내역 제외
- `git clone` : git 관리내역 포함 다운로드
- `git clone {저장소 이름}` 하면 `origin` 이라는 리모트 저장소가 자동 등록
- **HTTPS** 방식과 **SSH** 방식이 있으며 강의에서는 **HTTPS** 방식 사용

- VS Code로 해당 폴더 내용 살펴보기

S4-Lesson 3. Push & Pull

1. 원격으로 커밋 밀어올리기 (**push**)
   A. `leopards.yaml` 의 members에 Evie 추가
   - 커밋 메세지: `Add Evie to Leopards`
   B. 아래 명령어로 push

   ```
   git push
   ```

   이미 `git push -u origin main` 으로 대상 원격 브랜치가 지정되었기에 바로 push 가능

   C. GitHub or GitLab 페이지에서 확인
   - 파일 내역 및 커밋 내역 확인
2. 원격의 커밋 당겨오기 (**pull**)
   A. GitHub에서 `ledpards.yaml` 의 `members` 에 `Dongho` 추가
   - 커밋 내용 : `Add Dongho to Leopards`
   B. 아래 명령어로 pull

```
git pull
```

```
        C. 로컬에서 파일과 로그 살펴보기

3. **pull** 할 것이 있을 때 **push**를 한다면??
        A. 로컬에서 `leopards.yaml`의 `manager`를 `Dooli`로 수정
                - 커밋 내용 : Edit Leopards manager
        B. GitHub에서 `leopards.yaml`의 `coach`를 `Lupi`로 수정
         - 커밋 내용 : Edit Leopards coach
    C. push 해보기
                - 원격에 먼저 적용된 새 버전이 있으면 적용 불가
                - pull 해서 원격의 버전을 받아온 다음 push 가능
        D. push 할 것이 있을 시 pull 하는 두 가지 방법
         - `git pull --no-rebase` - **merge** 방식
             - 소스트리에서 확인해보기
             - reset으로 되돌린 다음 아래 방식 확인해보기
        - `git pull --rebase` - **rebase** 방식
             - pull 상의 rebase는 다름 (협업시 사용 OK)
        E. push 하기

4. 협업상 충돌 발생 해결하기
        A. 로컬에서 `panthers.yaml`에 `Vai` 추가
                - 커밋 내용  : `Add Vai to Panthers`
        B. 원격에서 `pantehrs.yaml`에 `Noru` 추가
                - 커밋 내용 : `Add Noru to Panthers`
        C. pull 하여 충돌상황 마주하기
                - `--no-rebase` 와 `--rebase` 모두 해보기

5. 로컬의 내용 강제 push 하기
        A. 로컬의 내역 충돌 전으로 `reset`
        B. 아래 명령어로 원격에 강제 적용
        ```bash
        git push --force
```

S4-Lesson 4. 원격의 브랜치 다루기
```

1. 로컬에서 브랜치 원격에 push 하기

A. `from-local 브랜치 만들기
B. 아래 명령어로 원격에 push

```
git push
```

대상 지정 에러 발생

```
git push -u origin from-local
```

C. 브랜치 목록 살펴보기

- GitHub에서 목록 보기
- 아래 명령어로 로컬과 원격의 브랜치 확인

```
git branch --all
```

2. 원격의 브랜치 로컬에 받아오기

A. GitHub에서 `from-remote` 브랜치 만들기
- `git branch -a` 에서 현재는 보이지 않음
B. 아래 명령어로 원격의 변경사항 확인

```
git fetch
```

- `git branch` -a 로 확인
  C. 아래의 명령어로 로컬에 같은 이름의 브랜치를 생성하여 연결하고 `switch`

```
git switch -t origin/from-remote
```

D. 원격의 브랜치 삭제

```
git push (원격 이름) --delete (원격의 브랜치 명)
```