

Git & Github



Git & Github 소개

깃(Git)이 뭔데요

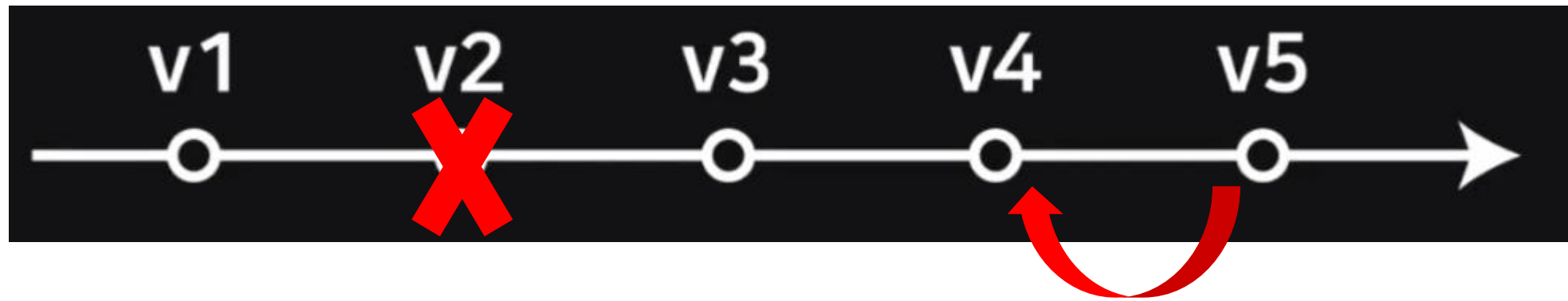
깃 홈페이지의 소개에 따르면, 깃은 소규모 프로젝트에서 초대형 프로젝트에 이르기까지 모든 것을 신속하고 효율적으로 처리하도록 설계된 오픈 소스 버전 제어 시스템이다.



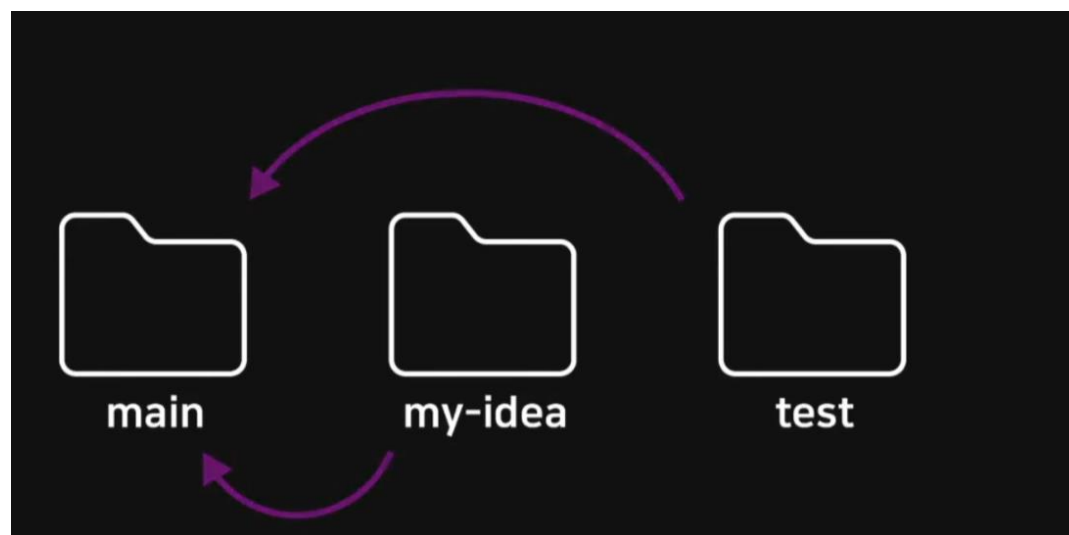
깃(Git)

깃은 VCS (Version Control System)의 한 종류임. 프로그램의 버전 관리를 위한 툴.

- Git은 프로젝트의 시간과 차원을 자유롭게 넘나들수 있도록 해줌
 - 시간 - 프로젝트의 버전을 과거로 되돌리거나 특정 내역을 취소할 수 있음.



- 차원 - 프로젝트의 여러 모드를 쉽게 전환하고 관리할 수 있음



깃의 주요 기능

- 버전 관리

깃을 이용하면 프로젝트 내 문서의 수정 이력을 체계적으로 관리할 수 있다. 문서의 내용을 특정 시점으로 되돌리거나, 서비스 버전과 개발 버전을 별도로 운영하는 등 프로젝트 관리에 대해 다양한 편의를 제공받을 수 있다.

- 협업 체계

여러 사람이 하나의 프로젝트를 함께 수행할 때, 하나의 저장소를 기반으로 자신만의 개발 버전을 관리할 수 있어 협업에 도움이 된다. 또한 원격 저장소를 운영할 경우에는 물리적인 장치(USB)나 메일 송수신 없이도 서로의 코드를 주고받거나 합치는 작업을 손쉽게 처리할 수 있다.



딤러닝-구현-초안.txt



딤러닝-구현-초안-수정.txt



딤러닝-구현-초안-최종.txt



딤러닝-구현-진짜-최종.txt



딤러닝-구현-이젠-진짜-최종.txt



계속되는 문서 복사로
저장 공간을 낭비하는 건 좋지 않다!
깃으로 수정 이력을 관리하자!

딥러닝-구현-초안.txt

딥러닝-구현-초안-수정.txt

딥러닝-구현-초안-최종.txt

딥러닝-구현-진짜-최종.txt

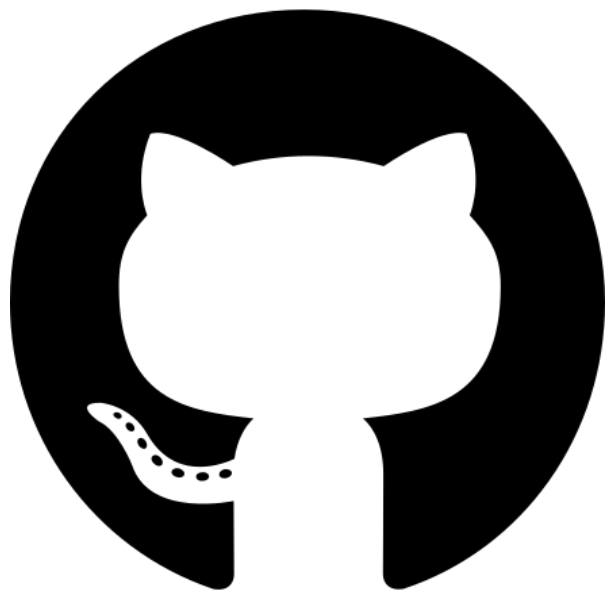
딥러닝-구현-이젠-진짜-최종.txt

깃허브(Github)

깃허브는 깃 저장소 호스팅을 지원하는 웹 서비스이다. 깃허브를 이용하면 온라인 상에 저장소를 만들어 원격으로 이를 관리할 수 있다.

또한, 깃이 명령행 인터페이스를 제공하는데 반해 깃허브는 그래픽 인터페이스를 제공하기 때문에 사용자 입장에서 보다 편리하게 깃 저장소를 관리할 수 있다.

결론! 깃허브는 깃 이용자들에게 편의를 제공하기 위해 존재하는 하나의 플랫폼이다.



Git 실습 준비하기

시작하기 전에 알아두자

깃을 사용하는 방법은 많다. 그러나 그 중에서도 가장 많이 쓰이는 방법은 역시 커맨드라인인터페이스(Command Line Interface, CLI)를 사용하는 방법이다. 이는 사용자가 약속된 명령어를 입력하면 시스템이 이를 수행하는 방식이다.

별도의 프로그램 설치 없이 깃만 설치하면 바로 사용할 수 있는 기본적인 방법이고, 깃의 모든 기능을 지원하는 인터페이스는 CLI 뿐이므로 깃 입문자라면 CLI 기반의 깃 사용법을 익히는 것이 여러모로 좋다고 할 수 있다.

깃 설치하기

깃을 사용하기 위해서는 먼저 사용할 PC에 깃을 설치해야 한다. 깃은 누구나 무료로 설치 및 사용할 수 있다. 윈도우, 맥 등 운영체제에 따라 설치 과정이 조금씩 달라지긴 하지만, 대체로 설치 방법은 무척 간단하다. 깃 설치하는 다음 페이지에서 진행하자.

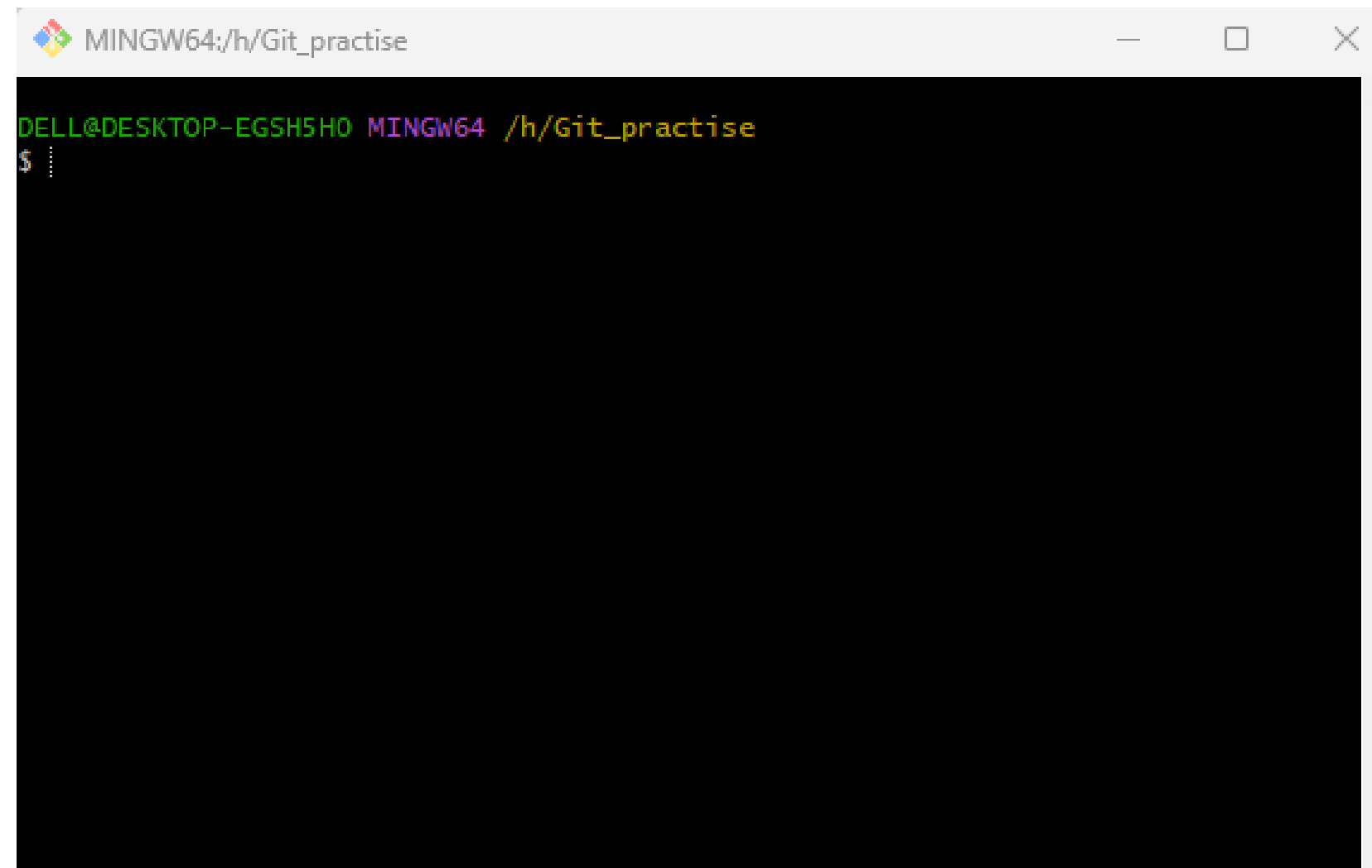
=> <https://git-scm.com/downloads>



페이지에서 운영체제를 감지하여 그에 맞는 설치 파일을 제공해 준다!
설치 파일을 내려받고, 이를 실행하여 설치를 진행하자.
특별한 사항 없이 [Next]를 눌러가며 설치를 마무리하면 된다

깃 설치하기

- 설치과정에서 **Git Bash**를 반드시 포함시켜야 함.
- 설치 후 Git Bash에서 아래 명령어로 테스트 해봄
`$ git --version`
- 협업시 윈도우와 맥에서 엔터 방식 차이로 인한 오류를 방지함
`$ git config --global core.autocrlf true`



```
DELL@DESKTOP-EGSH5H0 MINGW64 /h/Git_practise
$
```

윈도우에서 깃 배시(git bash)

SourceTree 설치하기

- <https://www.sourcetreeapp.com/>
 - Git을 GUI로 다룰 수 있도록 해주는 툴
- 설치시 BitBucket 계정 관련은 건너뛰기해도 좋음

VS Code의 기본 터미널을 Git Bash로 설정

- Git 뿐 아니라 다른 프로그래밍 작업에 있어서도 유용
 - VS Code에서 Ctrl + Shift + P
 - Select Default Profile 검색하여 선택
 - Git Bash 선택
 - 터미널에서 +로 새 창을 열어서 기본으로 Git Bash가 설정된 것 확인

명령어의 종류

우리가 사용해 볼 명령어는 크게 두 종류로 구분할 수 있다.

깃 명령어:

저장소 내에서 버전 관리, 협업 등 깃이 제공하는 다양한 기능을 수행하기 위해 입력하는 명령어로, git 으로 시작한다.

시스템 명령어:

폴더 이동, 파일 생성 및 삭제 등 컴퓨터 시스템 관련 기능을 수행하기 위해 입력하는 명령어로, 리눅스 운영체제의 시스템 명령어에 기반을 두고 있다.

명령어

VS code에서 폴더 생성후 실행

```
$ git config --global user.name "사용자명(닉네임)"
```

깃 저장소에서 사용할 사용자 정보 중 사용자명을 설정하는 깃 명령어

```
$ git config --global user.email "사용자메일"
```

깃 저장소에서 사용할 사용자 정보 중 사용자 메일 주소를 설정하는 깃 명령어

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

사용자 이름 확인

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

사용자 email 확인

명령어

```
$ git init
```

터미널이 열려 있는 폴더를 깃 저장소로 초기화하는 깃 명령어

- 폴더에 숨김모드로 .git 폴더 생성 확인
 - 이 폴더를 지우면 Git 관리내역이 삭제됨. (현 파일들은 유지)

파일 생성

아래 파일 생성

tigers.yaml

```
team: Tigers
```

```
manager: John
```

```
members:
```

- Linda
- William
- David

파일 생성

아래 파일 생성

lions.yaml

```
team: Lions
```

```
manager: Mary
```

```
members:
```

- Thomas
- Karen
- Margaret

명령어

\$ git status 실행

- 아래 메시지 생성시 git config --global --add safe.directory H:/Git_practice 실행

```
DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise
⊗ $ git status
fatal: detected dubious ownership in repository at 'H:/Git_practise'
'H:/Git_practise' is on a file system that does not record ownership
To add an exception for this directory, call:

    git config --global --add safe.directory H:/Git_practise

DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise
● $ git config --global --add safe.directory H:/Git_practise
```

명령어

\$ git status 실행

- 정상 실행

```
DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (master)
● $ git status
On branch master

No commits yet

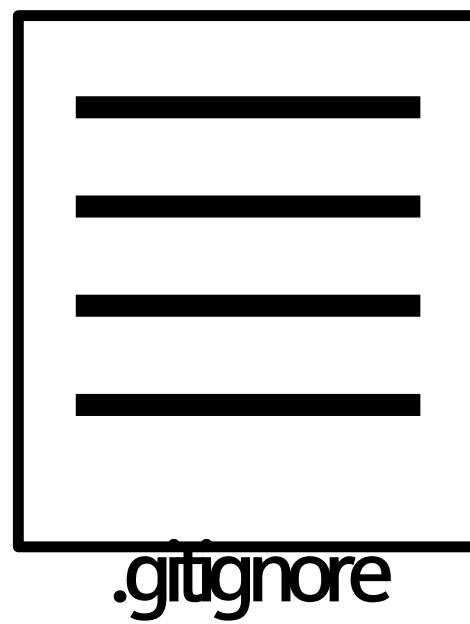
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    ttigers.yaml
    lions.yaml

nothing added to commit but untracked files present (use "git add" to track)
```

.gitignore 사용하기

Git의 관리에서 특정 파일/폴더를 배제해야 할 경우

깃 프로젝트 내 문서 중 수정 이력에서 제외하고 싶은 문서가 있다면 이를 추적 대상에서 완전히 제외시킬 수 있다. 이때에는 깃 설정 파일인 `.gitignore`를 사용한다.



`.gitignore`는 처음부터 존재하는 파일이 아니므로, 직접 만들어서 사용해야 한다.
시스템 명령어 `touch` 를 사용하면 새 파일을 생성할 수 있다.

Git의 관리에서 특정 파일/폴더를 배제해야 할 경우

a. 포함할 **필요가 없을** 때

- 자동으로 생성 또는 다운로드되는 파일들 (빌드 결과물, 라이브러리)

b. 포함하지 **말아야 할** 때

- 보안상 민감한 정보를 담은 파일

Git의 관리에서 특정 파일/폴더를 배제해야 할 경우

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/

Git의 관리에서 특정 파일/폴더를 배제해야 할 경우

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/**/*.log

실습(Git의 관리에서 특정 파일/폴더를 배제해야 할 경우)

.gitignore 사용

1) secrets.yaml 생성

```
id: admin  
pw: 1234abcd
```

2) git status 실행

```
$ git status  
On branch master  
  
No commits yet  
  
Untracked files:  
  (use "git add <file>..." to include in what will be committed)  
    ttigers.yaml  
    lions.yaml  
    secrets.yaml  
  
nothing added to commit but untracked files present (use "git add" to track)
```

3) .gitignore 파일 생성

```
secrets.yaml
```

실습(Git의 관리에서 특정 파일/폴더를 배제해야 할 경우)

4) git status 실행

```
DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (master)
● $ git status
On branch master

No commits yet

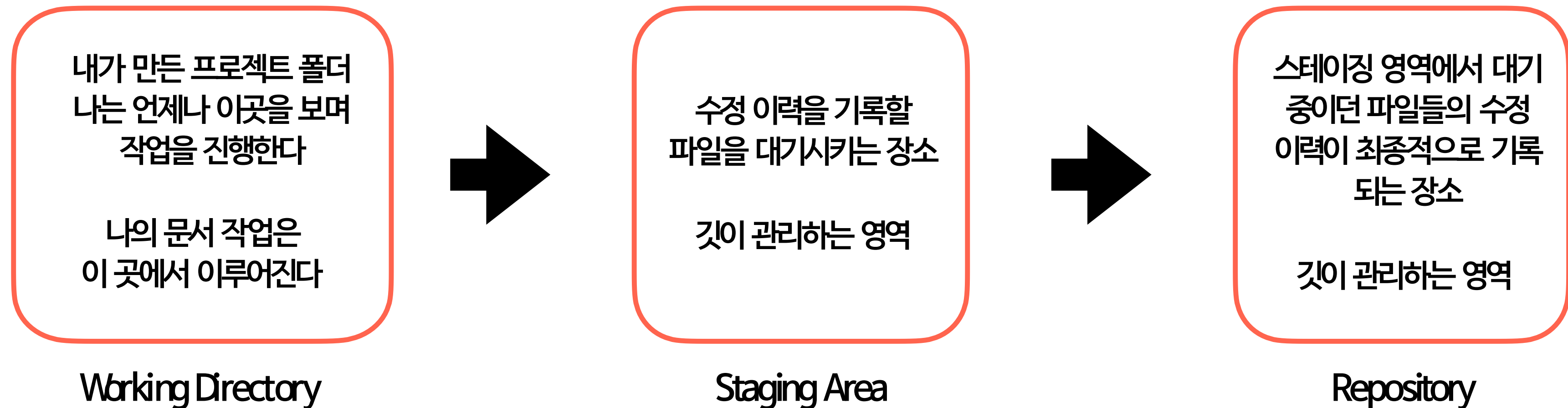
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore
        Ttigers.yaml
        lions.yaml

nothing added to commit but untracked files present (use "git add" to track)
```

Git 저장소 관리

Git 저장소 관리

깃 프로젝트는 내부에 가상의 관리 영역을 만들어 파일의 상태를 구분하고 버전을 관리한다. 관리 영역은 세 가지로, 각 영역의 이름과 역할은 다음과 같다.



Git 저장소 관리

깃은 깃 프로젝트의 작업 디렉터리 내 문서들의 수정 사항을 추적(tracking)한다.
깃은 문서의 상황에 따른 문서의 상태를 다음과 같이 표현한다.

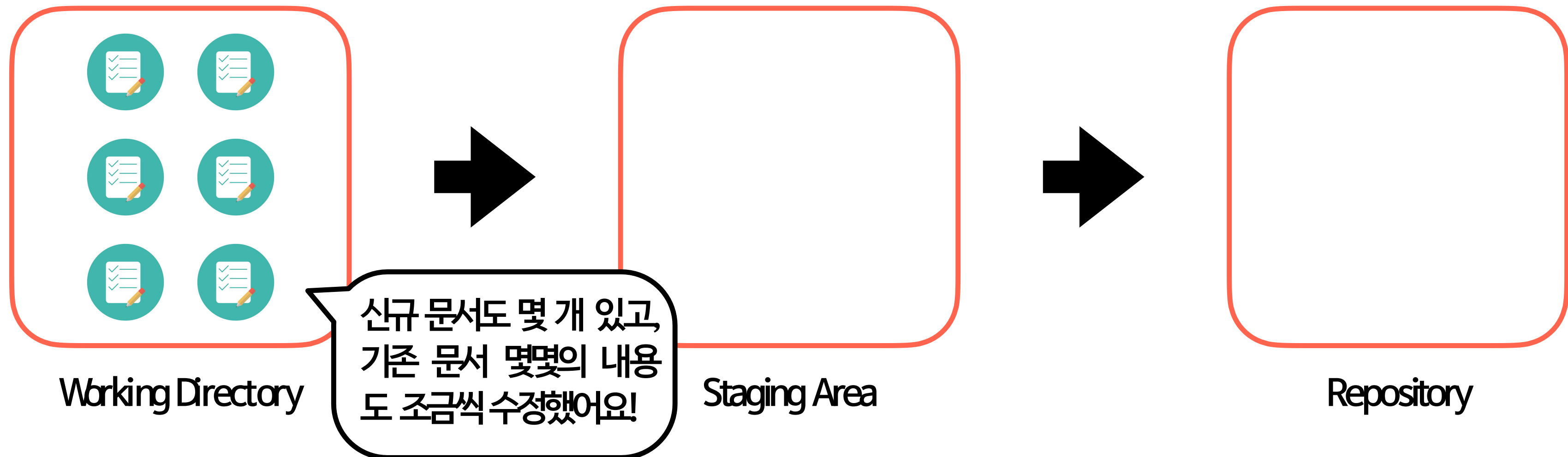


Working Directory

- untracked file : 이제 막 생성된 파일로, 추적이 되고 있지 않은 상태.
- unmodified file : 추적 중인 파일이나, 딱히 수정 사항이 없는 상태.
- modified file : 추적 중인 파일이며, 수정 사항이 감지된 상태.

Git 저장소 관리

깃 사용자는 워킹 디렉터리에서 감지된 신규 문서나 수정 문서를 스테이징 영역으로 이동시켜야 한다. 스테이징 영역으로 이동한 문서는 커밋(commit)이라는 작업을 거쳐 최종적으로 리포지토리에 기록된다.



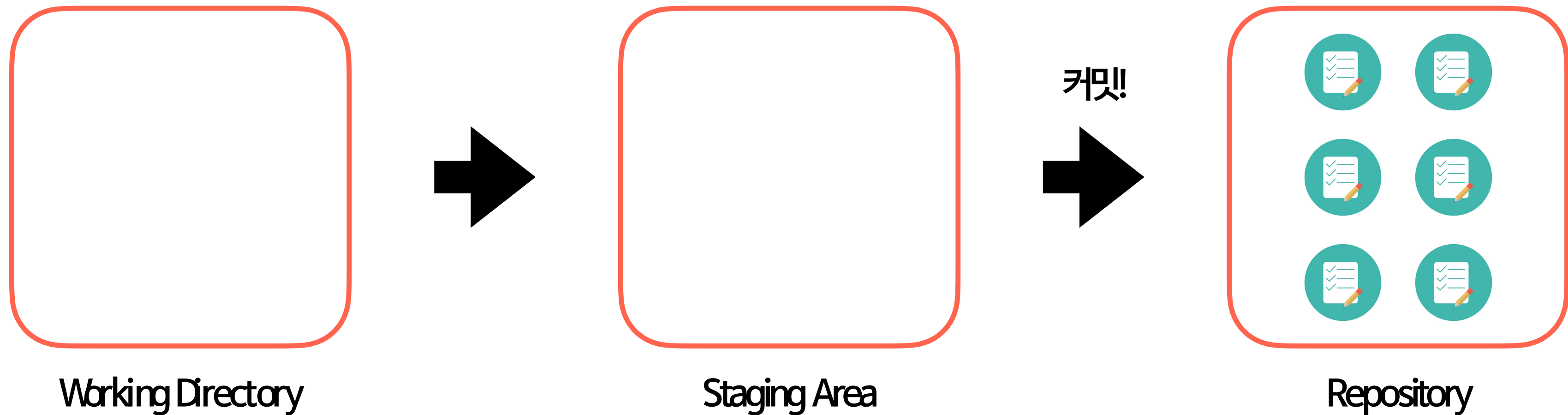
Git 저장소 관리

깃 사용자는 워킹 디렉터리에서 감지된 신규 문서나 수정 문서를 스테이징 영역으로 이동시켜야 한다. 스테이징 영역으로 이동한 문서는 커밋(commit)이라는 작업을 거쳐 최종적으로 리포지토리에 기록된다.



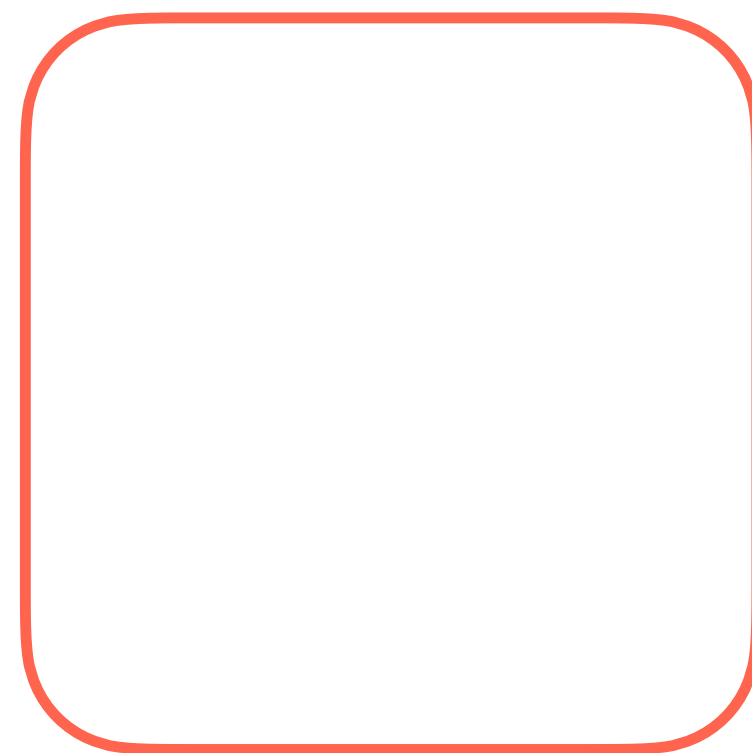
Git 저장소 관리

깃 사용자는 워킹 디렉터리에서 감지된 신규 문서나 수정 문서를 스테이징 영역으로 이동시켜야 한다. 스테이징 영역으로 이동한 문서는 커밋(commit)이라는 작업을 거쳐 최종적으로 리포지토리에 기록된다.

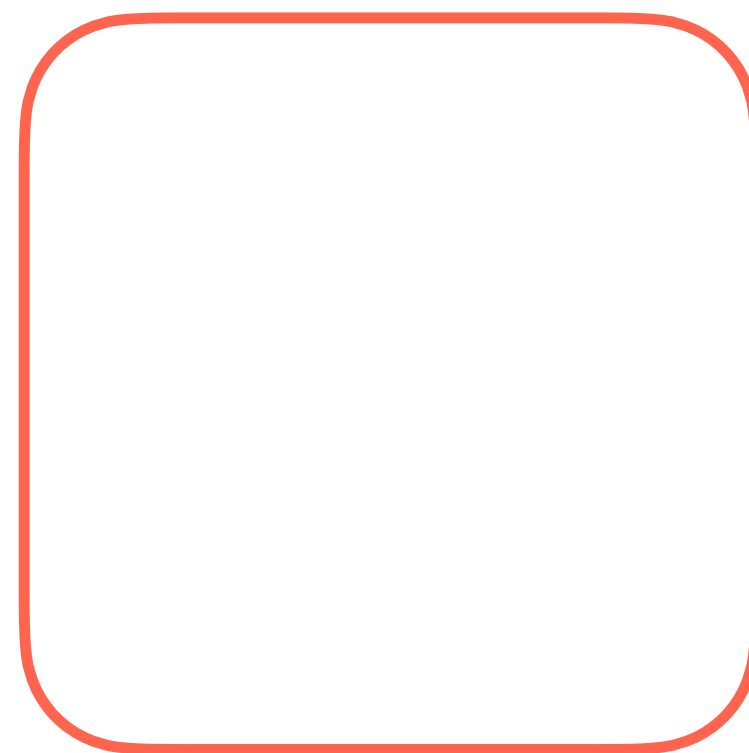
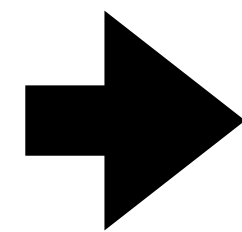


Git 저장소 관리

깃 사용자는 워킹 디렉터리에서 감지된 신규 문서나 수정 문서를 스테이징 영역으로 이동시켜야 한다. 스테이징 영역으로 이동한 문서는 커밋(commit)이라는 작업을 거쳐 최종적으로 리포지토리에 기록된다.

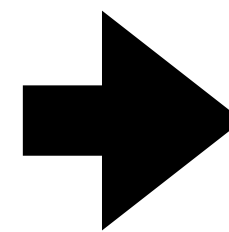


Working Directory



Staging Area

커밋!



저장소는 커밋한 내용을 기억하고, 이 기억을 쌓아 간다. 추후에 새로운 커밋이 또 진행되면, 언제 어떤 커밋이 있었는지 까지도 상세히 구분해 가며 기억한다.

Repository

깃 프로젝트 관리하기 관련 명령어들

```
$ git status
```

깃 프로젝트 상태를 확인하는 깃 명령어

```
$ git add
```

워킹 디렉터리 내 문서를 스테이징 영역에 추가하는 깃 명령어

```
$ git commit
```

스테이징 영역 내에 대기 중인 문서를 리포지토리에 추가하는 깃 명령어

```
$ git log
```

커밋한 수정 이력을 확인하는 깃 명령어

실습(프로젝트의 변경사항들을 버전에 담기)

1) git status 실행

```
DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (master)
● $ git status
○ On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    .gitignore
    ttigers.yaml
    lions.yaml

nothing added to commit but untracked files present (use "git add" to track)
```

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

2) 파일 하나 담기

```
$ git add tigers.yaml
```

실습(프로젝트의 변경사항들을 버전에 담기)

3) git status 실행

```
DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   tigers.yaml

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore
        lions.yaml
```

4) 모든 파일 담기

\$ git add .

```
DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   .gitignore
        new file:   lions.yaml
        new file:   tigers.yaml
```

실습(타임캡슐 묻기)

1) git commit 실행

2) Vi 입력 모드로 진입

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 log등에서 내역이 길 때 사용
아래로 스크롤	j	git log등에서 내역이 길 때 사용

- FIRST COMMIT 입력한 뒤 저장하고 종료

실습(타임캡슐 묻기)

3) git status 실행

```
DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (master)
• $ git status
On branch master
nothing to commit, working tree clean
```

4) git log 실행

```
DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (master)
• $ git log
commit d62e8e1668c647103231b1bd77643a3c08d47630 (HEAD -> master)
Author: Steve Choi <stechoi2015@gmail.com>
Date: Mon Jul 1 12:57:32 2024 +0900

    FIRST COMMIT
```

실습(다음 변경사항들을 만들고 타임캡슐에 묻기)

1) 변경 사항

- lions.yaml 파일 삭제
- tigers.yaml의 manager를 Donald로 변경
- leopards.yaml 파일 추가

```
team: Leopards
```

```
manager: Luke
```

```
members:
```

- Linda
- William
- David

실습(다음 변경사항들을 만들고 타임캡슐에 묻기)

2) git status 로 확인

```
DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (master)
● $ git status
○ On branch master
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    deleted:    lions.yaml
    modified:   tigers.yaml

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    leopards.yaml

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

실습(다음 변경사항들을 만들고 타임캡슐에 묻기)

3) git diff로 확인

```
DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (master)
• $ git diff
diff --git a/lions.yaml b/lions.yaml
deleted file mode 100644
index 416a903..0000000
--- a/lions.yaml
+++ /dev/null
@@ -1,8 +0,0 @@
-team: Lions
-
-manager: Mary
-
-members:
-- Thomas
-- Karen
-- Margaret
\ No newline at end of file
diff --git a/tigers.yaml b/tigers.yaml
index 6276780..c13deb3 100644
--- a/tigers.yaml
+++ b/tigers.yaml
@@ -1,6 +1,6 @@
 team: Tigers

-manager: John
+manager: Donald

members:
- Linda
```

실습(다음 변경사항들을 만들고 타임캡슐에 묻기)

4) git add . 실행

5) git status로 확인

```
DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (master)
● $ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   leopards.yaml
    deleted:    lions.yaml
    modified:   tigers.yaml
```

6) git commit -m "Replace Lions with Leopards" 로 commit

```
DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (master)
$ git commit -m "Replace Lions with Leopards"
● [master f2f11a1] Replace Lions with Leopards
 3 files changed, 9 insertions(+), 9 deletions(-)
 create mode 100644 leopards.yaml
 delete mode 100644 lions.yaml
```

실습(다음 변경사항들을 만들고 타임캡슐에 묻기)

[Tip] add와 commit을 한꺼번에 (새로 추가된(untracked) 파일이 없을 때 한정

```
git commit -am "(메시지)"
```

7) git status로 확인

실습(커밋 추가)

1) 첫번째 커밋 추가

- Tigers의 members에 George 추가
- 커밋 메시지: Add George to Tigers

```
DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (master)
● $ git commit -am "Add George to Tigers"
[master 764b31d] Add George to Tigers
1 file changed, 1 insertion(+), 1 deletion(-)
```

2) 두번째 커밋 추가

- cheetas.yaml 추가

team: Cheetas

manager: Laura

members:

- Ryan
- Anna
- Justin

- 커밋 메시지: Add team Cheetas

```
DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (master)
● $ git add cheetas.yaml

DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (master)
● $ git commit -m "Add team Cheetas"
[master ab20d04] Add team Cheetas
1 file changed, 8 insertions(+)
create mode 100644 cheetas.yaml
```

실습(커밋 추가)

3) 세번째 커밋 추가

- cheetas.yaml 삭제
- Leopards의 manager를 Nora로 수정
- panthers.yaml 추가

team: Panthers

manager: Sebastian

members:

- Violet
- Stella
- Anthony

- 커밋 메시지: Replace Cheetas with Panthers

```
DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (master)
• $ git status
On branch master
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        deleted:    cheetas.yaml
        modified:   leopards.yaml

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        panthers.yaml

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

```
DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (master)
• $ git add .

DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (master)
• $ git commit -m "Replace Cheetas with Panthers"
[master a70aade] Replace Cheetas with Panthers
3 files changed, 9 insertions(+), 10 deletions(-)
delete mode 100644 cheetas.yaml
create mode 100644 panthers.yaml
```


실습(커밋 추가)

4) git log 로 확인

```
DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (master)
• $ git log
commit a70aade7cb484672ec1d2b6d8e5b1f8503860dd4 (HEAD -> master)
Author: Steve Choi <stechoi2015@gmail.com>
Date:   Mon Jul 1 14:05:55 2024 +0900

    Replace Cheetas with Panthers

commit ab20d04a120fafeb6bf3cafb3ac69878b38c5689
Author: Steve Choi <stechoi2015@gmail.com>
Date:   Mon Jul 1 13:53:40 2024 +0900

    Add team Cheetas

commit 764b31d39bb9f45fc3f7f98136e2334462f38228
Author: Steve Choi <stechoi2015@gmail.com>
Date:   Mon Jul 1 13:46:15 2024 +0900

    Add George to Tigers

commit f2f11a19007c8cc460f5a4e010132fd33f1bd254
Author: Steve Choi <stechoi2015@gmail.com>
Date:   Mon Jul 1 13:24:41 2024 +0900

    Replace Lions with Leopards

commit d62e8e1668c647103231b1bd77643a3c08d47630
Author: Steve Choi <stechoi2015@gmail.com>
Date:   Mon Jul 1 12:57:32 2024 +0900

    FIRST COMMIT
```

실습(Sourcetree 확인)

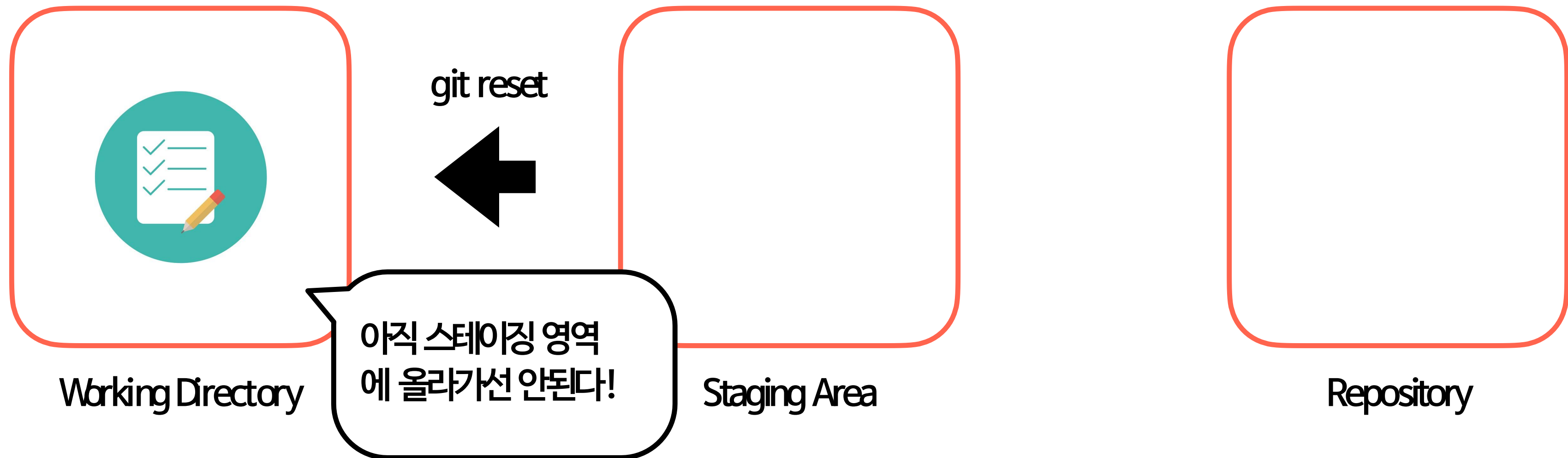
The screenshot shows the Sourcetree application interface. The top menu bar includes options like '파일(F)', '편집(E)', '보기(V)', '저장소(R)', '액션(A)', '도구', and '도움말(H)'. Below the menu is a toolbar with icons for '커밋', 'Pull', 'Push', '패치', '브랜치', '병합', '스태시', '폐기', and '태그'. The left sidebar shows the 'WORKSPACE' section with '파일 상태', 'History' (selected), and 'Search'. Below this is a search bar and a list of '브랜치' (branches) including 'master'. The main area displays the commit history for the 'master' branch, sorted by '날짜순으로 정렬' (sorted by date). The table shows the following commits:

그래프	설명	수정된 파일
○	Replace Cheetas with Panthers	3
●	Add team Cheetas	1
●	Add George to Tigers	1
●	Replace Lions with Leopards	3
●	FIRST COMMIT	3

Git에서 과거로 돌아가는 두 방식

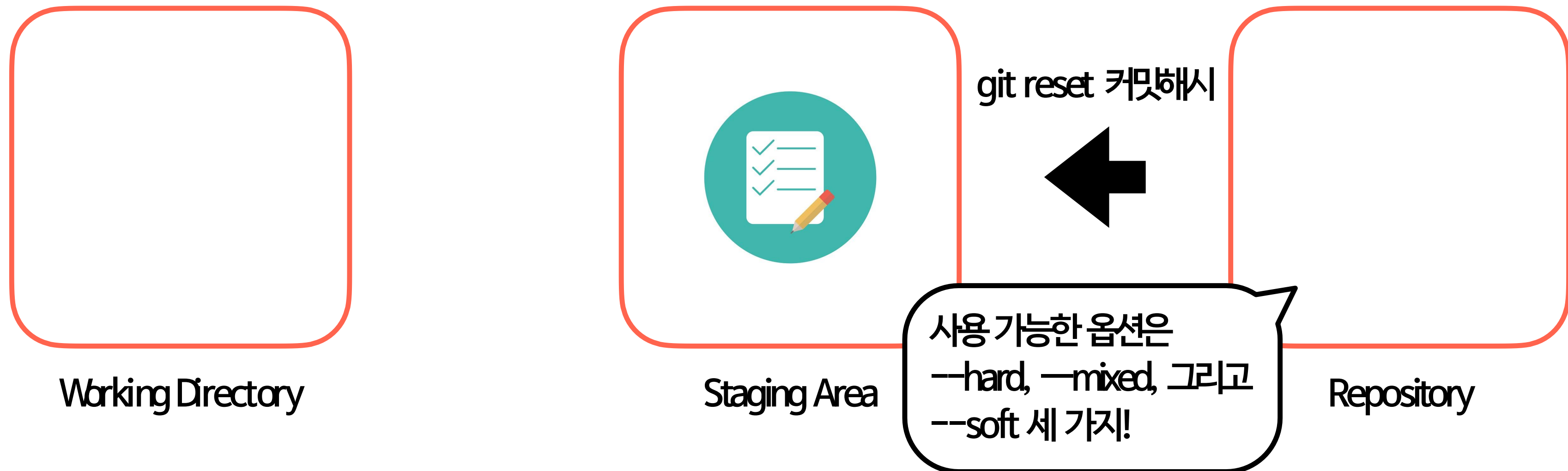
커밋할 생각이 없는데 스테이징 해버림

실수로 `git add` 명령을 수행해버린 경우, `git reset` 명령어를 입력하면 스테이징 영역에 올라가 있던 파일을 초기화할 수 있다. 즉, 워킹 디렉터리로 되돌릴 수 있다.

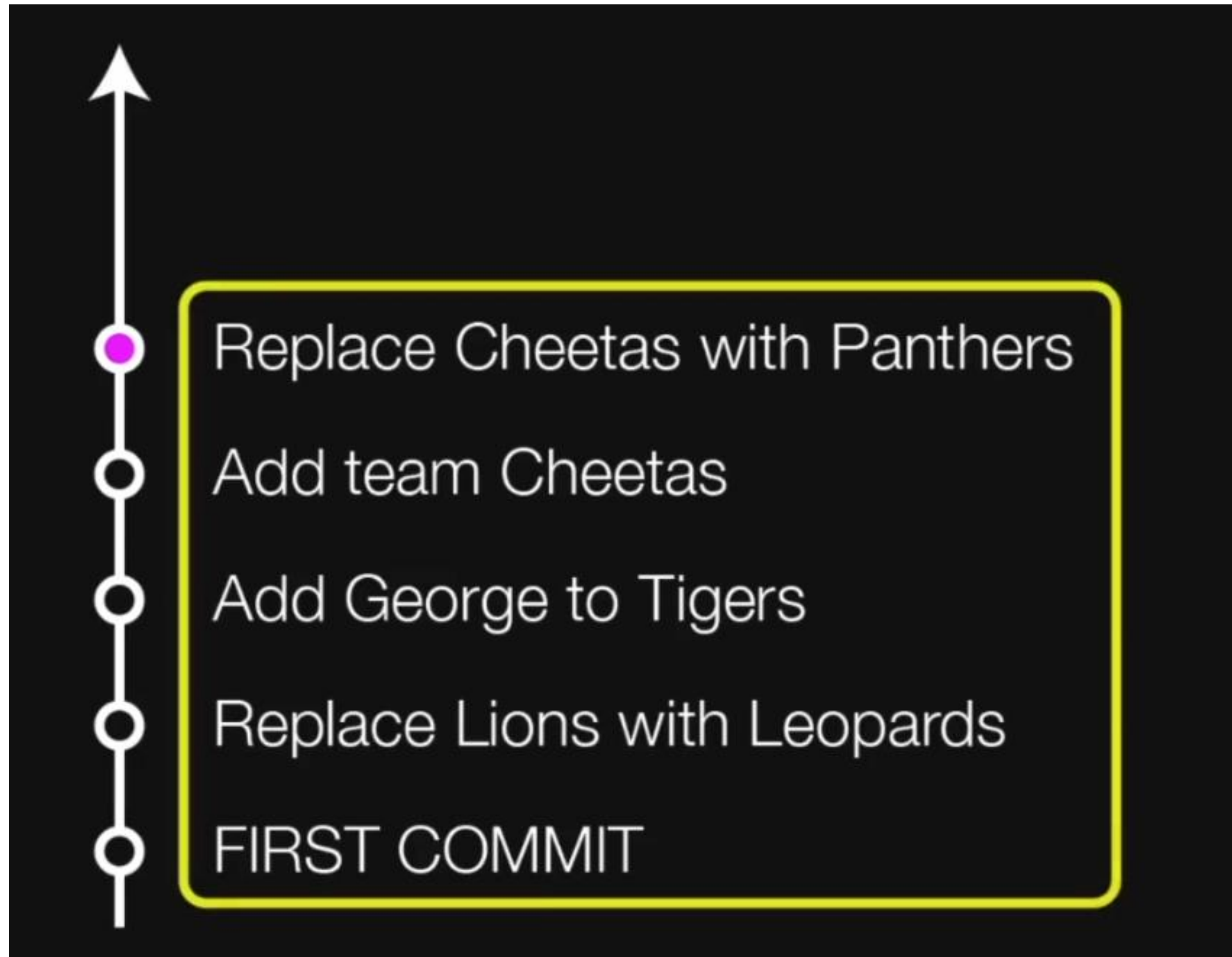


커밋하면 안 되는 사항인데 커밋해버림

실수로 `git commit` 명령을 수행 해버려 잘못된 커밋 이력이 추가된 경우, `git reset` 명령어와 커밋 해시를 함께 입력하면 커밋 이력 되돌리기가 수행된다.
`git reset` 커밋 되돌리기를 수행할 때는 세 가지 옵션 중 하나를 선택할 수 있다.



현재작업



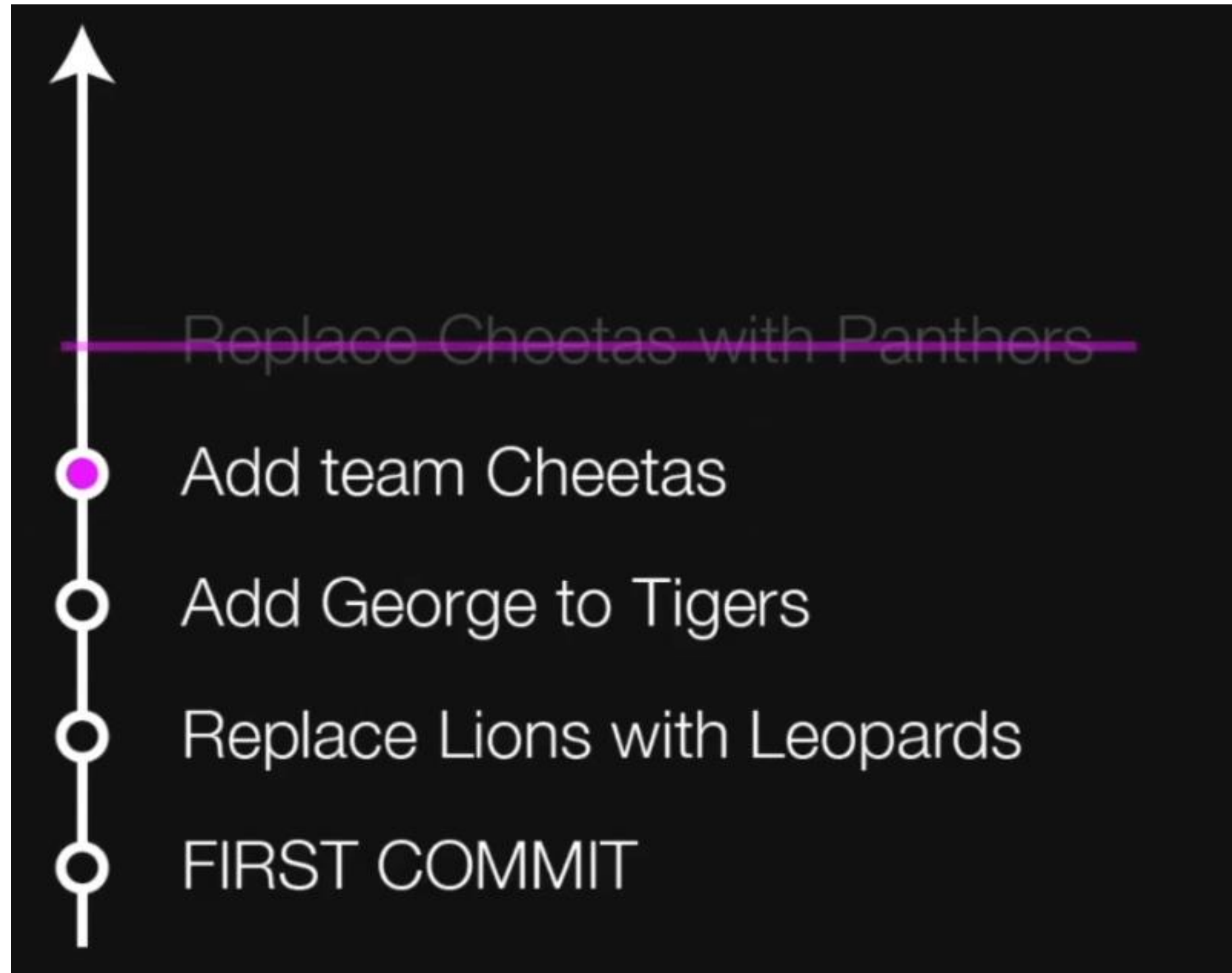
과거로 전환



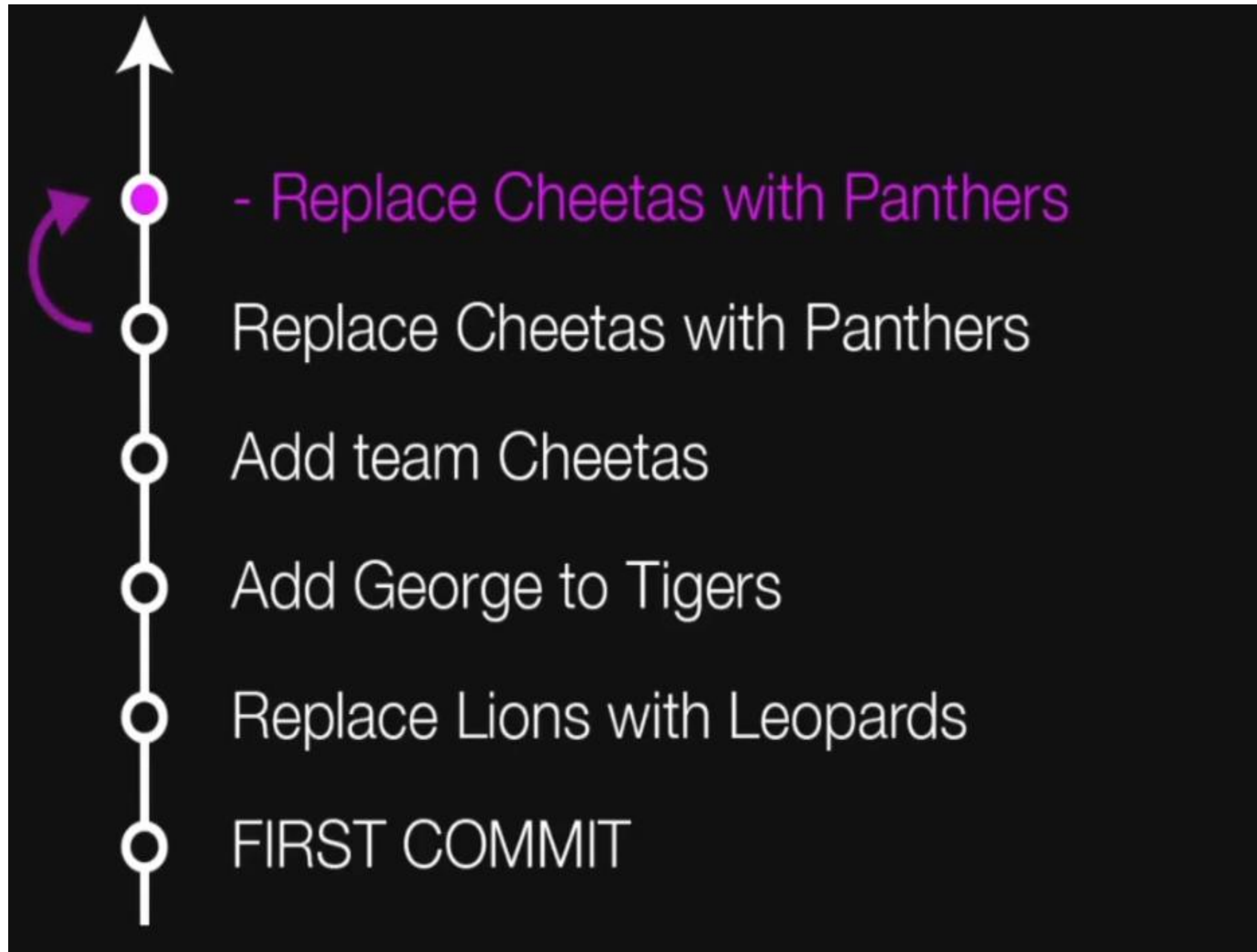
reset



reset



revert



revert



불필요한 수정 사항을 없애고 싶은데, 이조차도 기록으로 남기고 싶다

git revert 명령은 특정 커밋의 수정 사항을 되돌리되, '커밋 수정 사항을 되돌렸다'라는 사실을 이력으로 남기는 명령이다. 따라서 이를 수행하면 이력(커밋해시)이 줄어드는 게 아니라, 오히려 늘어난다.

```
$ git revert 321abcd
```

⇒ 321abcd 커밋을 없애거나 되돌리는 게 아니다.
해당 커밋에서 기록된 수정 사항을 되돌리는 것이다!

실습(Git에서 과거로 돌아가는 방식 : reset)

1. 사전 작업

. git 폴더를 복사해두기

. git 폴더 없앤 다음 git 상태 확인해보기

2. git log로 이력 확인

```
$ git log
commit a70aade7cb484672ec1d2b6d8e5b1f8503860dd4 (HEAD -> master)
Author: Steve Choi <stechoi2015@gmail.com>
Date: Mon Jul 1 14:05:55 2024 +0900

    Replace Cheetas with Panthers

commit ab20d04a120fafeb6bf3cafb3ac69878b38c5689
Author: Steve Choi <stechoi2015@gmail.com>
Date: Mon Jul 1 13:53:40 2024 +0900

    Add team Cheetas

commit 764b31d39bb9f45fc3f7f98136e2334462f38228
Author: Steve Choi <stechoi2015@gmail.com>
Date: Mon Jul 1 13:46:15 2024 +0900

    Add George to Tigers
```

실습(Git에서 과거로 돌아가는 방식)

3. 되돌아갈 시점 : Add team Cheetas의 커밋 해시 복사
- git reset --hard 돌아갈 커밋 해시

```
DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (master)
● $ git reset --hard ab20d04a120fafeb6bf3cafb3ac69878b38c5689
HEAD is now at ab20d04 Add team Cheetas
```

이후의 커밋 삭제

- cheetas.yaml 삭제 -> 복원
- Leopards의 manager를 Nora로 수정 -> 복원
- panthers.yaml 추가 -> 삭제
- 커밋 메시지: Replace Cheetas with Panthers

실습(Git에서 과거로 돌아가는 방식)

3. 되돌아갈 시점 : Add team Cheetas의 커밋 해시 복사
- git reset --hard 돌아갈 커밋 해시

```
DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (master)
● $ git reset --hard ab20d04a120fafeb6bf3cafb3ac69878b38c5689
HEAD is now at ab20d04 Add team Cheetas
```

이후의 커밋 삭제

- cheetas.yaml 삭제 -> 복원
- Leopards의 manager를 Nora로 수정 -> 복원
- panthers.yaml 추가 -> 삭제
- 커밋 메시지: Replace Cheetas with Panthers

실습(Git에서 과거로 돌아가는 방식)

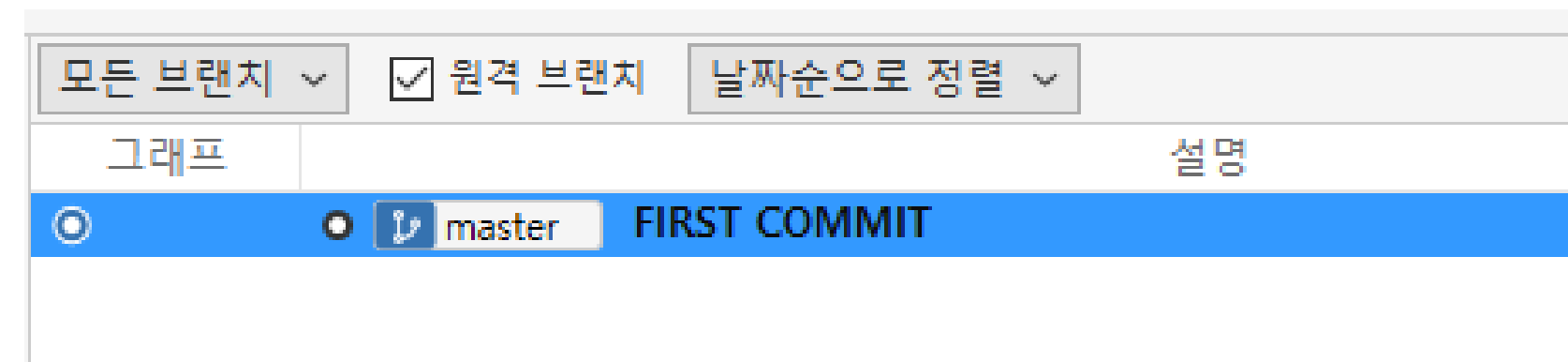
4. first commit으로 돌아가기

```
DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (master)
$ git reset --hard d62e8e1668c
HEAD is now at d62e8e1 FIRST COMMIT
```

5. git log로 확인 (sourcetree로도 확인)

```
DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (master)
$ git log
commit d62e8e1668c647103231b1bd77643a3c08d47630 (HEAD -> master)
Author: Steve Choi <stechoi2015@gmail.com>
Date: Mon Jul 1 12:57:32 2024 +0900

    FIRST COMMIT
```



6. git 폴더 삭제 및 백업해둔 git 폴더 복원

7. git log 실행

실습(Git에서 과거로 돌아가는 방식)

8. git status 실행

```
• $ git status
On branch master
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        deleted:    leopards.yaml
        deleted:    panthers.yaml
        modified:   tigers.yaml

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        lions.yaml

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

git은 마지막 commit에서 변화가 있는 것으로 생각

9. git reset --hard 로 마지막 commit으로 복원

```
DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (master)
• $ git reset --hard
HEAD is now at a70aade Replace Cheetas with Panthers
```

10. lions.yaml 삭제

실습(Git에서 과거로 돌아가는 방식)

11. git status 실행

```
DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (master)
● $ git status
On branch master
nothing to commit, working tree clean
```

실습(Git에서 과거로 돌아가는 방식 : revert)

1. git log로 이력 확인

```
DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (master)
• $ git log
commit a70aade7cb484672ec1d2b6d8e5b1f8503860dd4 (HEAD -> master)
Author: Steve Choi <stechoi2015@gmail.com>
Date: Mon Jul 1 14:05:55 2024 +0900

    Replace Cheetas with Panthers

commit ab20d04a120fafeb6bf3cafb3ac69878b38c5689
Author: Steve Choi <stechoi2015@gmail.com>
Date: Mon Jul 1 13:53:40 2024 +0900

    Add team Cheetas

commit 764b31d39bb9f45fc3f7f98136e2334462f38228
Author: Steve Choi <stechoi2015@gmail.com>
Date: Mon Jul 1 13:46:15 2024 +0900

    Add George to Tigers
```

2. Add George to Tigers의 커밋 되돌려보기

[git revert 되돌릴 커밋 해시]

- git revert 764b31d39bb9f45fc3f7f98136e2334462f38228

실습(Git에서 과거로 돌아가는 방식 : revert)

3. Vi 모드에서 :wq 입력

```
DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (master)
$ git revert 764b31d39bb9f45fc3f7f98136e2334462f38228
[master 65f7d79] Revert "Add George to Tigers"
1 file changed, 1 insertion(+), 1 deletion(-)
```

4. sourcetree에서 확인

모든 브랜지 ▾			<input checked="" type="checkbox"/> 원석 브랜지	날짜순으로 정렬 ▾
그래프	설명		수정된 파일	
○	○ master	Revert "Add George to Tigers"	1	
●	Replace Cheetas with Panthers		3	
●	Add team Cheetas		1	
●	Add George to Tigers		1	
●	Replace Lions with Leopards		3	
●	FIRST COMMIT		3	

실습(Git에서 과거로 돌아가는 방식 : revert)

5. git log로 이력 확인

```
commit ab20d04a120fafeb6bf3cafb3ac69878b38c5689
Author: Steve Choi <stechoi2015@gmail.com>
Date: Mon Jul 1 13:53:40 2024 +0900

    Add team Cheetas

commit 764b31d39bb9f45fc3f7f98136e2334462f38228
Author: Steve Choi <stechoi2015@gmail.com>
Date: Mon Jul 1 13:46:15 2024 +0900

    Add George to Tigers

commit f2f11a19007c8cc460f5a4e010132fd33f1bd254
Author: Steve Choi <stechoi2015@gmail.com>
Date: Mon Jul 1 13:24:41 2024 +0900

    Replace Lions with Leopards
```

6. Replace Lions with Leopards의 커밋 되돌려보기

- leopards.yaml을 삭제해야 하는데 [Replace cheetas with Panthers]에서 leopards.yaml 수정한 내역 때문에 충돌

[git revert 되돌릴 커밋 해시]

- git revert f2f11a19007c8cc460f5a4e010132fd33f1bd254

실습(Git에서 과거로 돌아가는 방식 : revert)

```
DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (master)
$ git revert f2f11a19007c8cc460f5a4e010132fd33f1bd254
CONFLICT (modify/delete): leopards.yaml deleted in parent of f2f11a1 (Replace Lions with Leopards) and modified in HEAD. Version HEAD of leopards.yaml left in tree.
error: could not revert f2f11a1... Replace Lions with Leopards
hint: After resolving the conflicts, mark them with
hint: "git add/rm <pathspec>", then run
hint: "git revert --continue".
hint: You can instead skip this commit with "git revert --skip".
hint: To abort and get back to the state before "git revert",
hint: run "git revert --abort".
```

- git rm leopards.yaml 실행

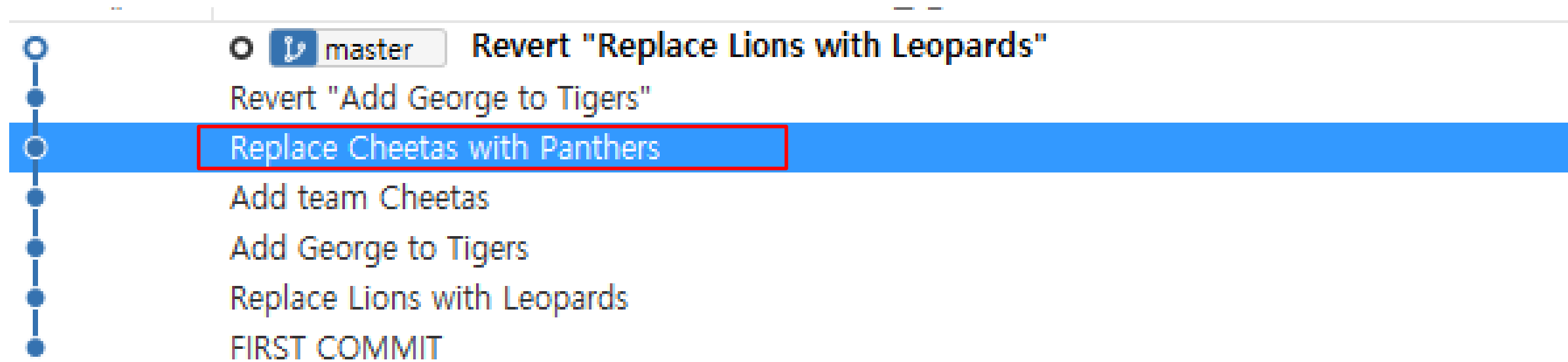
```
DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (master|REVERTING)
$ git rm leopards.yaml
rm 'leopards.yaml'
```

- git revert --continue 실행
- Vi 모드에서 :q 입력

```
DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (master|REVERTING)
$ git revert --continue
[master 5a73e15] Revert "Replace Lions with Leopards"
3 files changed, 9 insertions(+), 8 deletions(-)
delete mode 100644 leopards.yaml
create mode 100644 lions.yaml
```

실습(Git에서 과거로 돌아가는 방식 : revert)

7. revert 전으로 돌아가기



- git reset --hard a70aade

```
DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (master)
$ git reset --hard a70aade
HEAD is now at a70aade Replace Cheetas with Panthers
```

그래프	설명	수정
○	master Replace Cheetas with Panthers	3
●	Add team Cheetas	1
●	Add George to Tigers	1
●	Replace Lions with Leopards	3
●	FIRST COMMIT	3

실습(Git에서 과거로 돌아가는 방식 : revert)

[Tip] commit 안하고 revert하기

- add George to Tiger 에서의 revert
- git revert --no-commit 764b31d39bb9f45fc3f7f9813

```
DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (master)
● $ git revert --no-commit 764b31d39bb9f45fc3f7f9813
```

- git status

```
DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (master|REVERTING)
● $ git status
○ On branch master
You are currently reverting commit 764b31d.
(all conflicts fixed: run "git revert --continue")
(use "git revert --skip" to skip this patch)
(use "git revert --abort" to cancel the revert operation)

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   tigers.yaml
```

- git reset --hard : 마지막 커밋 상태로 돌아가고 싶을때

```
DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (master|REVERTING)
● $ git reset --hard
HEAD is now at a70aade Replace Cheetas with Panthers
```


Sourcetree로 진행하기

실습(sourcetree에서 진행하기)

1. 변경사항 만들고 커밋하기

- leopards.yaml 삭제
- gitignore에 *.config 추가
- hello.txt 추가 (내용 자유)
- 커밋 메시지: Commit with SourceTree < = sourcetree에서 작업

커밋하지 않은 변경사항	3	1 7 20
o master Replace Cheetas with Panthers	3	1 7 20
Add team Cheetas	1	1 7 20
Add George to Tigers	1	1 7 20
Replace Lions with Leopards	3	1 7 20
FIRST COMMIT	3	1 7 20

2. Staging Area (add)

대기 중인 파일, 파일 상태순 정렬

스테이지에 올라간 파일 (모두 스테이지에서 내리기) 선택 내용 스테이지에서 내리기

- .gitignore
- hello.txt
- leopards.yaml

1. working stage

스테이지에 올라가지 않은 파일

모두 스테이지에 올리기

선택 내용 스테이지에 올리기

.gitignore

+

leopards.yaml

+

hello.txt

+

3. commit

커밋 Pull Push 패치 브랜치 병합

WORKSPACE

파일 상태 History Search

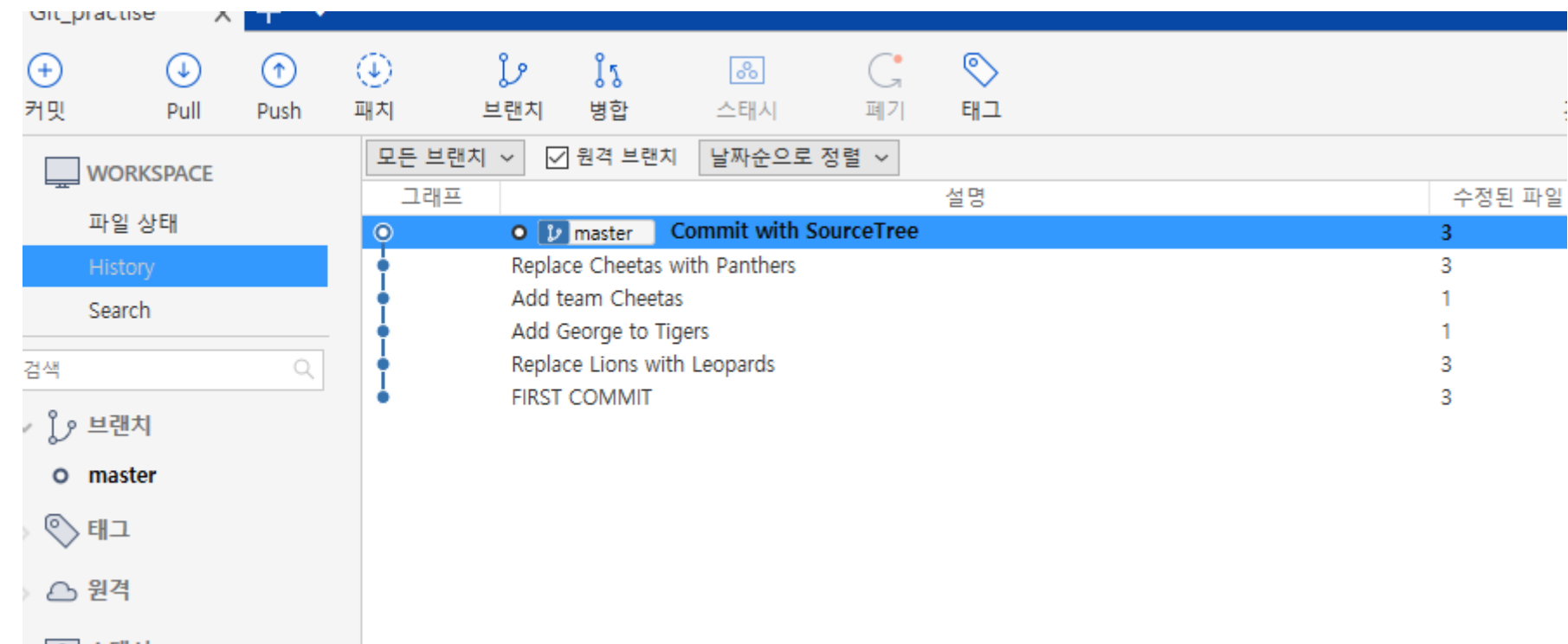
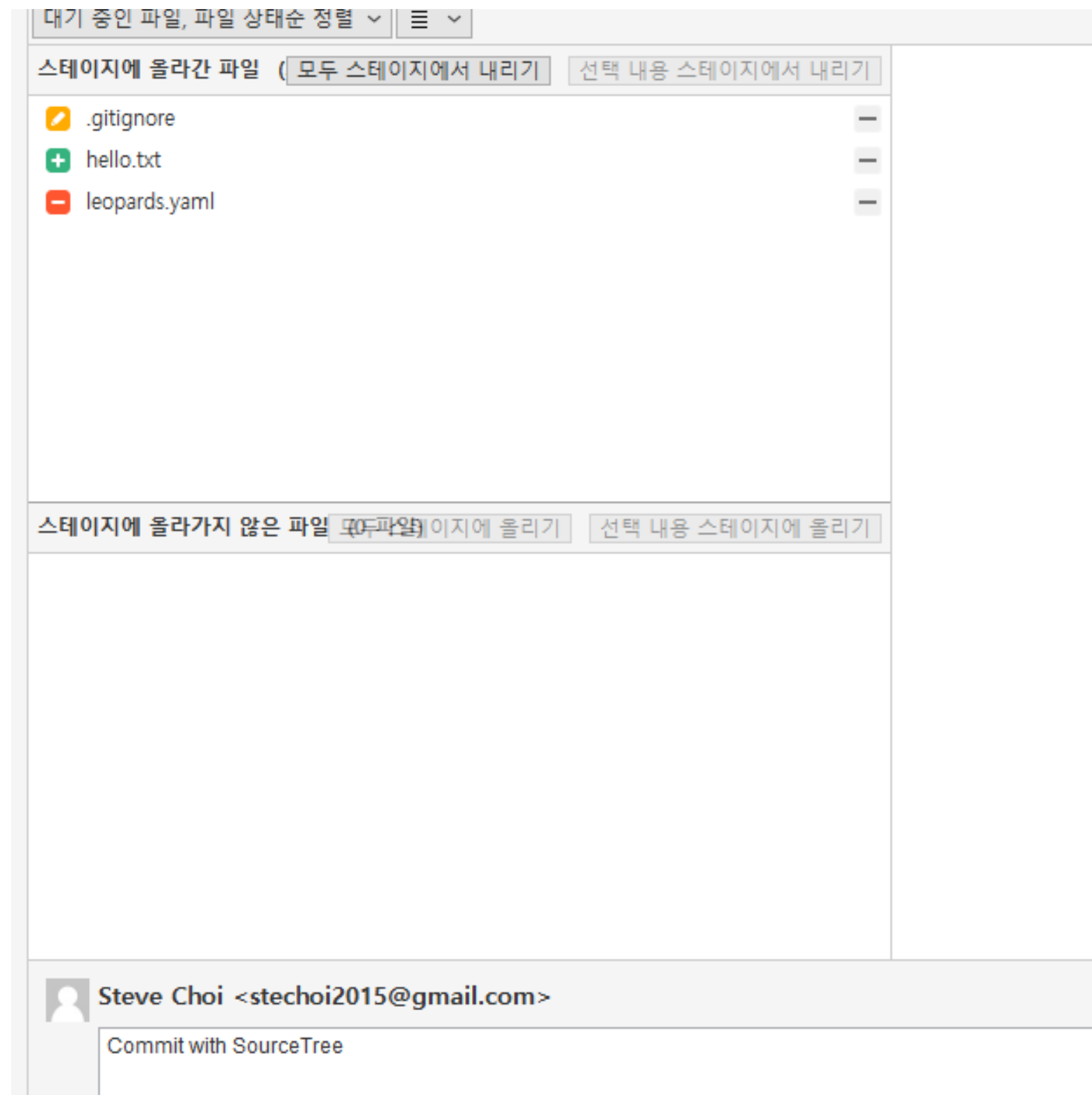
모든 브랜치 원격 브랜치

그래프

커밋하지 않은 변경사항

- o master Replace Cheetas with Panthers
- Add team Cheetas
- Add George to Tiger
- Replace Lions with L
- FIRST COMMIT

실습(sourcetree에서 진행하기)



실습(sourcetree에서 진행하기)

2. git log로 확인

```
DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (master)
$ git log
commit a0fd4795554dd5c0fbd7d450ce8548980d21b1ae (HEAD -> master)
Author: Steve Choi <stechoi2015@gmail.com>
Date: Mon Jul 1 17:07:29 2024 +0900

    Commit with SourceTree

commit a70aade7cb484672ec1d2b6d8e5b1f8503860dd4
Author: Steve Choi <stechoi2015@gmail.com>
Date: Mon Jul 1 14:05:55 2024 +0900

    Replace Cheetas with Panthers

commit ab20d04a120fafeb6bf3cafb3ac69878b38c5689
Author: Steve Choi <stechoi2015@gmail.com>
Date: Mon Jul 1 13:53:40 2024 +0900
```

실습(sourcetree에서 진행하기)

3. Add George to Tigers 를 revert 하기

SourceTree commit history table:

그래프	설명	수정된 파일	날짜
○	Commit with SourceTree	3	1 7 2024 17:07
●	Replace Cheetas with Panthers	3	1 7 2024 14:05
●	Add team Cheetas	1	1 7 2024 13:53
○	Add George to Tigers	1	1 7 2024 13:46
●	Replace Lions with Leopards	3	1 7 2024 13:24
●	FIRST COMMIT	3	1 7 2024 12:57

파일 상태순 정렬 (1 파일)

커밋: 764b31d39bb9f45fc3f7f98136e23
상위 항목: f2f11a1900
작성자: Steve Choi <stechoi2015@gmail.com>
날짜: 2024년 7월 1일 월요일 오후 1:46:15
커밋한 사람: Steve Choi

SourceTree commit history table after revert:

그래프	설명	수정된 파일
○	Revert "Add George to Tigers" reverted	1
●	Commit with SourceTree	3
●	Replace Cheetas with Panthers	3
●	Add team Cheetas	1
○	Add George to Tigers	1
●	Replace Lions with Leopards	3
●	FIRST COMMIT	3

실습(sourcetree에서 진행하기)

4. Replace Cheetas with Panthers 를 reset하기

SourceTree interface showing the commit list and the context menu for the commit 'Replace Cheetas with Panthers'.

The commit list shows the following commits:

그래프	설명	수정된 파일	날짜
○ master	Revert "Add George to Tigers"	1	1 7 2024 17:16
○	Commit with SourceTree	3	1 7 2024 17:07
○	Replace Cheetas with Panthers	3	1 7 2024 14:05
○	Add team Cheetas	1	1 7 2024 13:53
○	Add George to Tigers	1	1 7 2024 13:46
○	Replace Lions with Leopards	3	1 7 2024 13:24
○	FIRST COMMIT	3	1 7 2024 12:57

The context menu for the 'Replace Cheetas with Panthers' commit includes the following options:

- 체크아웃...
- 병합...
- 재배치...
- 태그...
- 아카이브...
- 브랜치...
- a70aade의 자식 커밋을 쌍방향 재배치...
- 이 커밋까지 현재 브랜치를 초기화**
- 커밋 되돌리기...
- 패치 생성...
- 체리 픽
- SHA 값을 클립보드에 복사
- 커스텀 액션

The 'Commit Reset...' dialog box shows the following information:

- 커밋 초기화...
- 브랜치 포인터를 옮기겠습니까?
- 브랜치 초기화: master
- 커밋할 것: a70aade: Replace Cheetas with Panthers
- 사용 중인 모드: Hard - 모든 작업 상태 내 변경 사항을 버림

The commit list after the reset operation shows the following commits:

그래프	설명	수정된 파일
○ master	Replace Cheetas with Panthers	3
○	Add team Cheetas	1
○	Add George to Tigers	1
○	Replace Lions with Leopards	3
○	FIRST COMMIT	3

reset

실습(sourcetree에서 진행하기)

5.Replace Lions with Leopards 를 revert하기

The screenshot displays the Sourcetree application interface. At the top, there are tabs for '패치' (Patch), '브랜치' (Branch), '병합' (Merge), '스태시' (Stash), '폐기' (Discard), and '태그' (Tag). Below these, there are filters for '모든 브랜치' (All Branches), '원격 브랜치' (Remote Branches), and '날짜순으로 정렬' (Sort by Date). The main area shows a commit history table with columns for '그래프' (Graph), '설명' (Description), '수정된 파일' (Files Changed), '날짜' (Date), and '작성자' (Author).

그래프	설명	수정된 파일	날짜	작성자
○ master	Replace Cheetas with Panthers	3	1 7 2024 14:05	Steve Choi
●	Add team Cheetas	1	1 7 2024 13:53	Steve Choi
●	Add George to Tigers	1	1 7 2024 13:46	Steve Choi
○	Replace Lions with Leopards	3	1 7 2024 13:24	Steve Choi
●	FIRST COMMIT	3	1 7 2024 12:57	Steve Choi

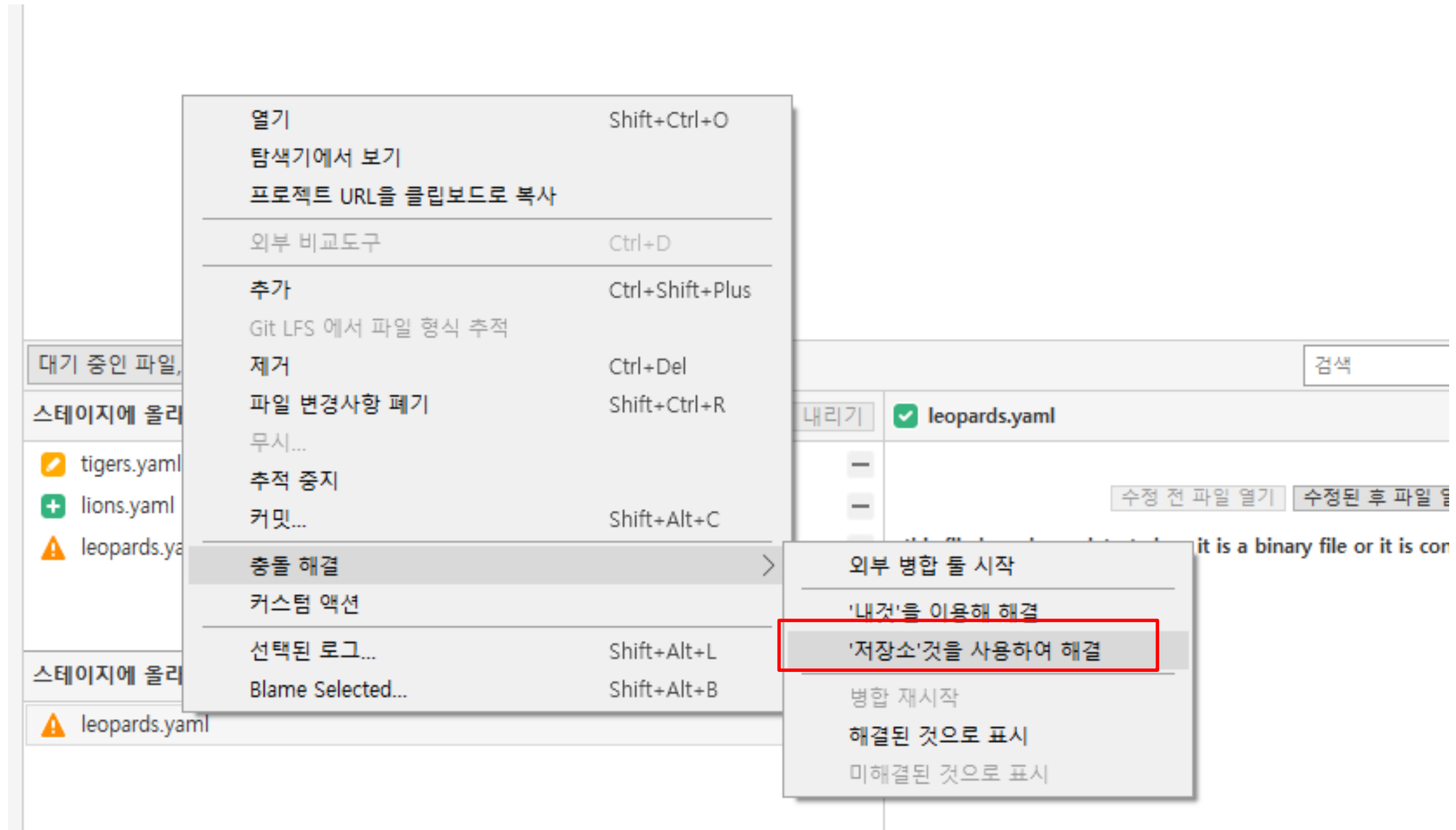
A context menu is open over the 'Replace Lions with Leopards' commit, showing options like '체크아웃...' (Checkout...), '병합...' (Merge...), '재배치...' (Rebase...), '태그...' (Tag...), '아카이브...' (Archive...), '브랜치...' (Branch...), 'f2f11a1의 자식 커밋을 쌍방향 재배치...' (Rebase f2f11a1's child commits bidirectionally...), '이 커밋까지 현재 브랜치를 초기화' (Reset current branch to this commit), '커밋 되돌리기...' (Revert commit...), '패치 생성...' (Create patch...), and '해지...' (Unstage...). The '커밋 되돌리기...' option is highlighted with a red box.

A '충돌 병합' (Merge Conflict) dialog box is displayed, warning that there are merge conflicts and advising the user to resolve them before continuing. The dialog includes a '닫기(C)' (Close) button and a checkbox for '다시 묻지 않습니다' (Don't ask me again).

At the bottom, the file explorer shows the 'leopards.yaml' file with the following content:

```
1 + team: Leopards
2 +
3 + manager: Luke
4 +
5 + members:
```

실습(sourcetree에서 진행하기)



실습(sourcetree에서 진행하기)

The screenshot shows the Sourcetree application window for a repository named 'Git_practise'. The top toolbar contains several icons, with the '커밋' (Commit) icon, represented by a plus sign inside a circle, highlighted with a red box. Below the toolbar, the left sidebar shows the 'History' tab selected. The main area displays a commit history table with columns for '그래프' (Graph), '설명' (Description), and '수정된 파일' (Modified Files). The table shows a series of commits on the 'master' branch, with the most recent commit being 'Replace Cheetas with Panthers'. The bottom status bar shows the user 'Steve Choi <stechoi2015@gmail.com>' and the word 'Revert'.

그래프	설명	수정된 파일
○	커밋하지 않은 변경사항	3
○ master	Replace Cheetas with Panthers	3
●	Add team Cheetas	1
●	Add George to Tigers	1
●	Replace Lions with Leopards	3
●	FIRST COMMIT	3

This screenshot shows the same Sourcetree interface after a revert operation. The commit history table now shows a 'Revert' commit at the top, which has reverted the previous changes. The word 'reverted' is written in red text next to the 'Revert' commit description. The rest of the interface, including the sidebar and status bar, remains the same.

그래프	설명	수정된 파일
○	○ master Revert	3
●	Replace Cheetas with Panthers	3
●	Add team Cheetas	1
●	Add George to Tigers	1
●	Replace Lions with Leopards	3
●	FIRST COMMIT	3

브랜치 관리하기

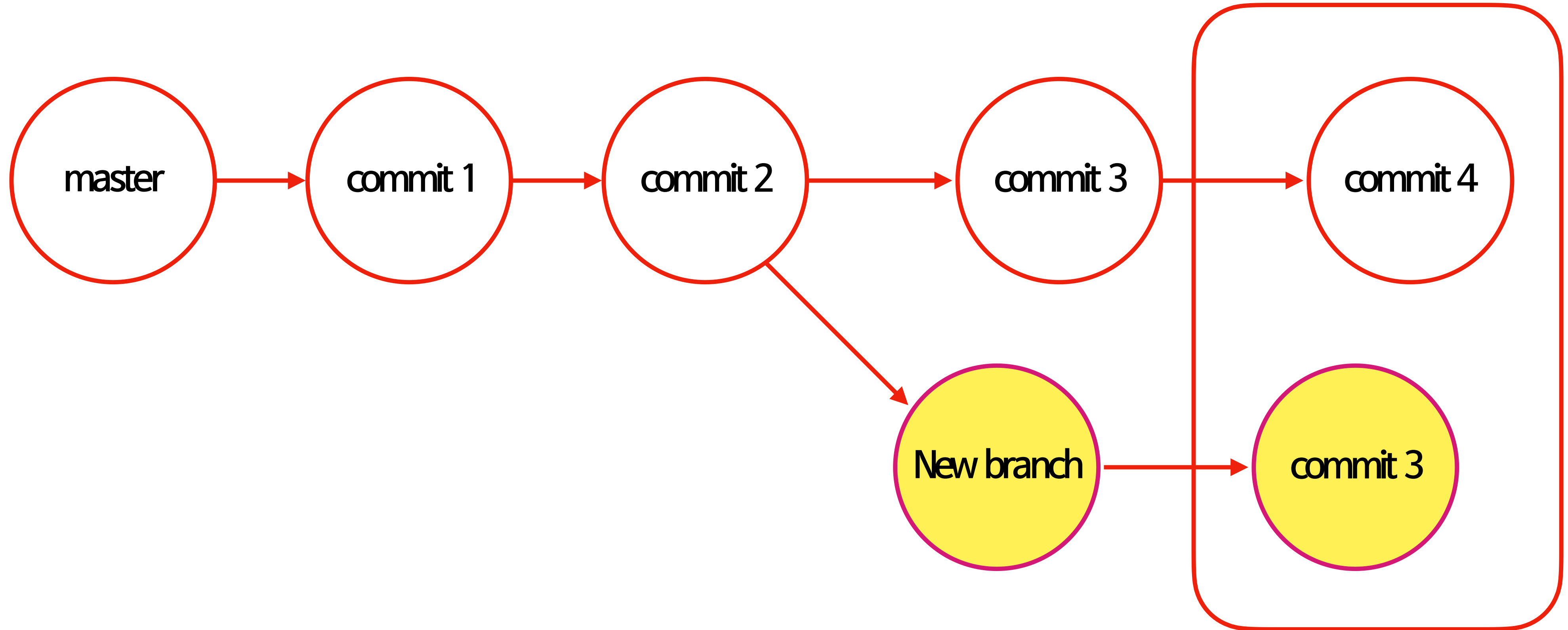
브랜치

개발을 하다 보면 코드를 여러 개로 복사해야 하는 일이 자주 생기는데, 코드를 통째로 복사하고 나서 원래 코드와는 상관없이 독립적으로 개발을 진행할 수 있도록 하는 것이 브랜치의 사용 목적이다. 브랜치란 기존 저장소에서 분기된 저장소의 '복사본'이다.

깃 프로젝트가 처음 만들어지면, 'master'라는 이름의 기본 브랜치 하나가 만들어진다. 여기에서부터 개발은 시작되고, 이를 바탕으로 여러 브랜치를 분기해가며 작업할 수 있다.

⇒ 경우에 따라 기본 브랜치가 'main'인 경우도 있으나, 단지 이름일 뿐이라 상관없다!

서로 다른 브랜치의 내용을 병합하는 'merge'
명령을 수행할 수 있다!



깃 브랜치 관리하기 관련 명령어들

```
$ git branch
```

현재 브랜치 목록을 볼 수 있는 깃 명령어

```
$ git branch 브랜치 이름
```

새로운 브랜치를 생성하는 깃 명령어

```
$ git switch 브랜치 이름
```

작업 중인 브랜치를 변경하는 깃 명령어

```
$ git merge 브랜치 이름
```

현재 브랜치에 다른 브랜치의 내용을 병합하는 깃 명령어

실습 (브랜치)

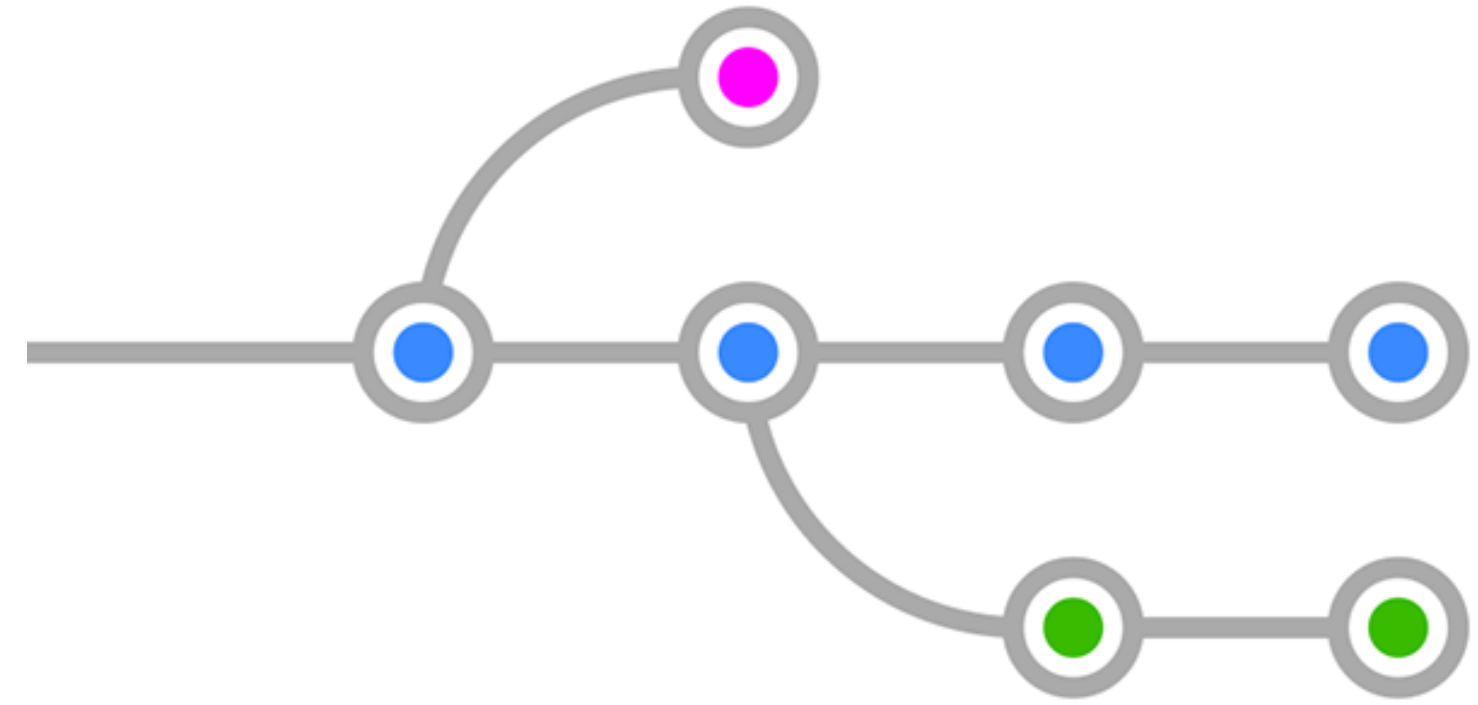
- 프로젝트를 하나 이상의 모습으로 관리해야 할 때

예) 실배포용, 테스트서버용, 새로운 시도용

- 여러 작업들이 각각 독립되어 진행될 때

예) 신기능 1, 신기능 2, 코드개선, 긴급수정...

각각의 차원에서 작업한 뒤 확정된 것을 메인 차원에 통합



실습(브랜치)

1. 브랜치 생성 / 이동 / 삭제하기

1) add-coach란 이름의 브랜치 생성

- git branch add-coach

```
DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (main)
• $ git branch add-coach
```

2) 브랜치 목록 확인

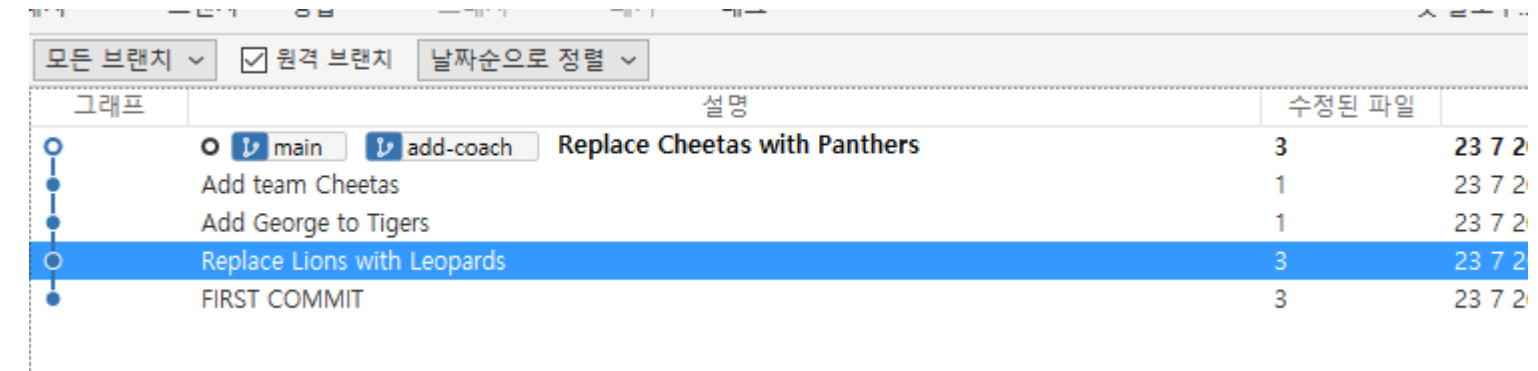
- git branch

```
DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (main)
• $ git branch
  add-coach
* main
```

3) add-coach 브랜치로 이동

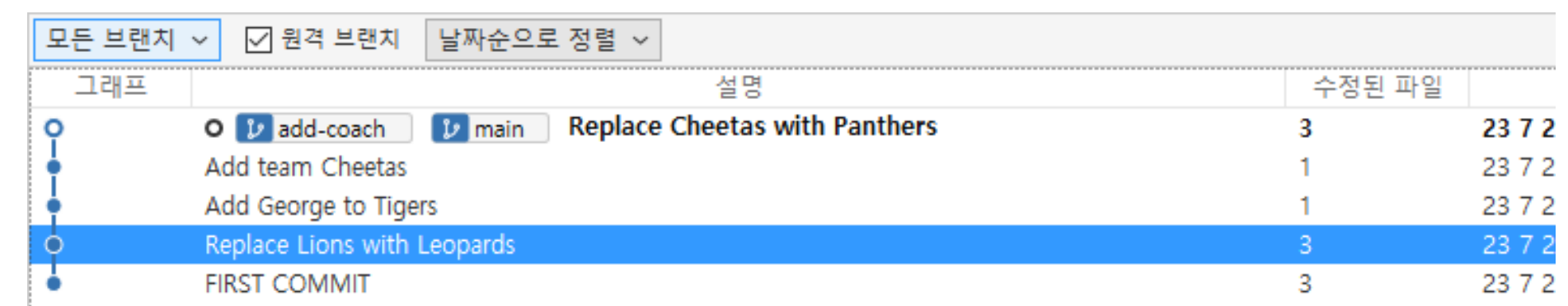
- git switch add-coach

```
DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (main)
• $ git switch add-coach
Switched to branch 'add-coach'
```



Git GUI showing branch history. The 'add-coach' branch is selected. The history shows a sequence of commits: FIRST COMMIT, Replace Lions with Leopards, Add George to Tigers, Add team Cheetas, and Replace Cheetas with Panthers.

그래프	설명	수정된 파일
○ main	Replace Cheetas with Panthers	3 23 7 2
○	Add team Cheetas	1 23 7 2
○	Add George to Tigers	1 23 7 2
○	Replace Lions with Leopards	3 23 7 2
○	FIRST COMMIT	3 23 7 2



Git GUI showing branch history. The 'main' branch is selected. The history shows a sequence of commits: FIRST COMMIT, Replace Lions with Leopards, Add George to Tigers, Add team Cheetas, and Replace Cheetas with Panthers.

그래프	설명	수정된 파일
○ add-coach	Replace Cheetas with Panthers	3 23 7 2
○	Add team Cheetas	1 23 7 2
○	Add George to Tigers	1 23 7 2
○	Replace Lions with Leopards	3 23 7 2
○	FIRST COMMIT	3 23 7 2

실습(브랜치)

4) main로 이동

- git switch main

5) 브랜치 생성과 동시에 이동하기

- git switch -c new-teams

```
DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (main)
● $ git switch -c new-teams
○ Switched to a new branch 'new-teams'

DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (new-teams)
$ █
```

6) 브랜치 목록 확인

- git branch

```
DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (new-teams)
● $ git branch
  add-coach
  main
* new-teams
```


실습(브랜치)

7) 브랜치 삭제하기

- git branch -d (삭제할 브랜치명)
- to-delete란 브랜치 만들고 삭제해보기

```
DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (new-teams)
• $ git branch to-delete

DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (new-teams)
• $ git branch
  add-coach
  main
* new-teams
  to-delete

DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (new-teams)
• $ git branch -d to-delete
Deleted branch to-delete (was 5c7dc2a).

DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (new-teams)
• $ git branch
  add-coach
  main
* new-teams
```

8) 브랜치 이름 바꾸기

- git branch -m (기존 브랜치명) (새 브랜치명)

실습(브랜치)

8) 브랜치 이름 바꾸기

- git branch -m (기존 브랜치명) (새 브랜치명)
- git branch -m to-delete to-erase

```
DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (new-teams)
● $ git branch to-delete

DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (new-teams)
● $ git branch -m to-delete to-erase

DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (new-teams)
● $ git branch
  add-coach
  main
* new-teams
  to-erase

DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (new-teams)
● $ git branch -d to-erase
Deleted branch to-erase (was 5c7dc2a).
```

실습(각각의 브랜치에서 서로 다른 작업해보기)

1. main 브랜치

1) Leopards의 members에 Olivia 추가

- 커밋 메시지: Add Olivia to Leopards

2) Panthers의 members에 Freddie 추가

- 커밋 메시지: Add Freddie to Panthers

3) add-coach 브랜치로 이동하여 해당 코드들 확인

```
DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (main)
● $ git commit -am 'Add Oliver to Leopards'
[main c34a88f] Add Oliver to Leopards
1 file changed, 2 insertions(+), 1 deletion(-)

DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (main)
● $ git commit -am 'Add Freddie to Panthers'
[main a877c43] Add Freddie to Panthers
1 file changed, 2 insertions(+), 1 deletion(-)
```

모든 브랜치

☒ 원격 브랜치

날짜순으로 정렬

그래프		설명	수정!
	<div>main</div> <div>Add Freddie to Panthers</div>	1	
	Add Oliver to Leopards	1	
	<div>new-teams</div> <div>add-coach</div> <div>Replace Cheetas with Panthers</div>	3	
	Add team Cheetas	1	
	Add George to Tigers	1	
	Replace Lions with Leopards	3	
	FIRST COMMIT	3	

실습(각각의 브랜치에서 서로 다른 작업해보기)

2. add-coach 브랜치

1) Tigers의 매니저 정보 아래 coach: Grace 추가

- 커밋 메시지: Add Coach Grace to Tigers

2) Leopards의 매니저 정보 아래 coach: Oscar 추가

- 커밋 메시지: Add Coach Oscar to Leopards

3) Panthers의 매니저 정보 아래 Oscar to Leopards 추가

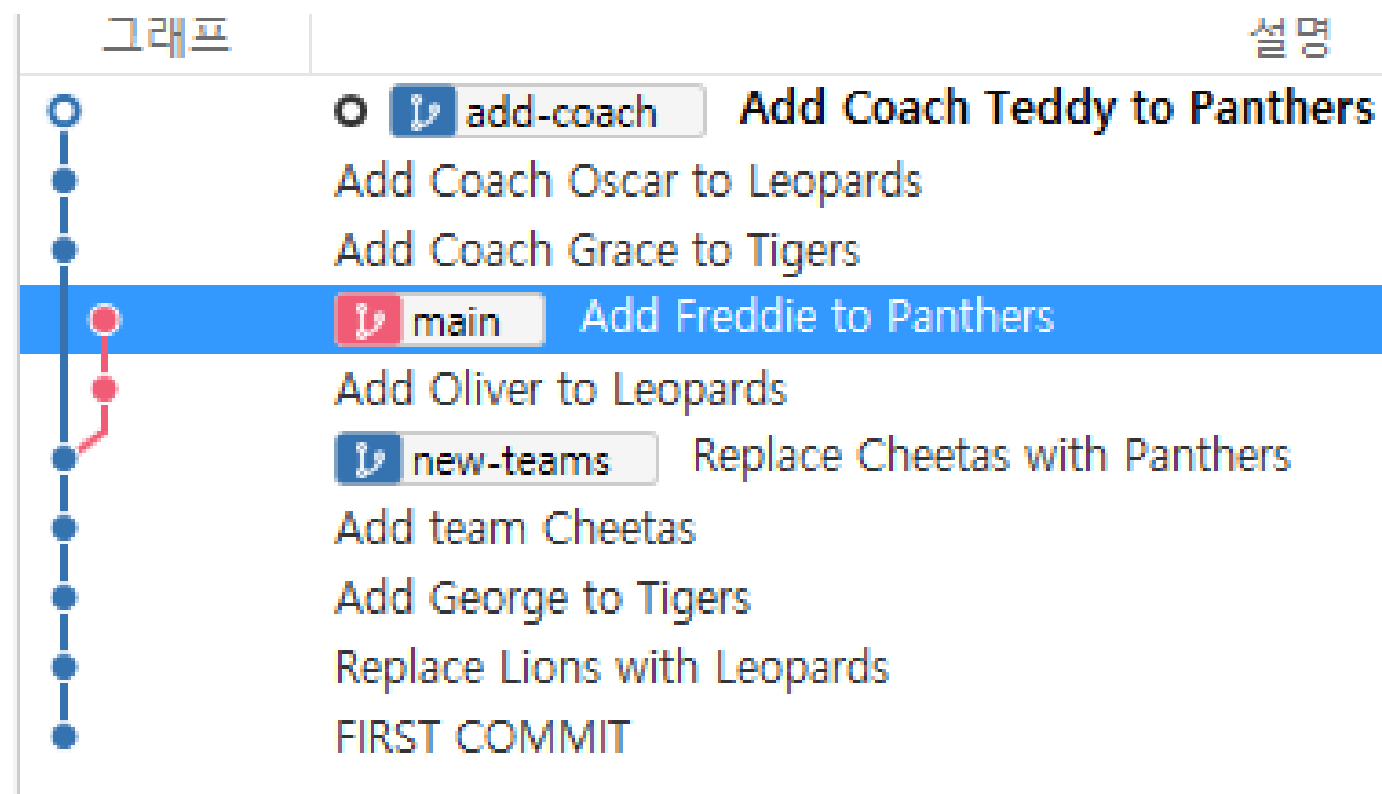
- 커밋 메시지: Add Coach Teddy to Panthers

```
DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (main)
• $ git switch add-coach
  Switched to branch 'add-coach'

DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (add-coach)
• $ git commit -am 'Add Coach Grace to Tigers'
[add-coach 051105f] Add Coach Grace to Tigers
 1 file changed, 2 insertions(+)

DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (add-coach)
• $ git commit -am 'Add Coach Oscar to Leopards'
[add-coach 50093f1] Add Coach Oscar to Leopards
 1 file changed, 2 insertions(+)

DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (add-coach)
• $ git commit -am 'Add Coach Teddy to Panthers'
[add-coach 249f17c] Add Coach Teddy to Panthers
 1 file changed, 2 insertions(+)
```



실습(각각의 브랜치에서 서로 다른 작업해보기)

4) main으로 이동후 파일 확인

5) new-teams로 이동후 파일 확인

실습(각각의 브랜치에서 서로 다른 작업해보기)

3. new-teams 브랜치

1) pumas.yaml 추가

team: Pumas

manager: Jude

members:

- Ezra
- Carter
- Finn

- 커밋 메시지: Add team Pumas

```
DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (new-teams)
● $ git add .

DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (new-teams)
● $ git commit -m 'Add team Pumas'
[new-teams d255528] Add team Pumas
1 file changed, 8 insertions(+)
create mode 100644 pumas.yaml
```

실습(각각의 브랜치에서 서로 다른 작업해보기)

2) jaguars.yaml 추가

team: Jaguars

manager: Stanley

members:

- Caleb
- Harvey
- Myles

- 커밋 메시지: Add team Jaguars

```
DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (new-teams)
● $ git add .

DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (new-teams)
● $ git commit -m 'Add team Jaguars'
○ [new-teams 48065c6] Add team Jaguars
  1 file changed, 8 insertions(+)
  create mode 100644 jaguars.yaml
```

실습(각각의 브랜치에서 서로 다른 작업해보기)

4. 결과 살펴보기

- 1) git log : 위치한 브랜치에서의 내역만 볼 수 있음
- 2) 여러 브랜치의 내역 편리하게 보기

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

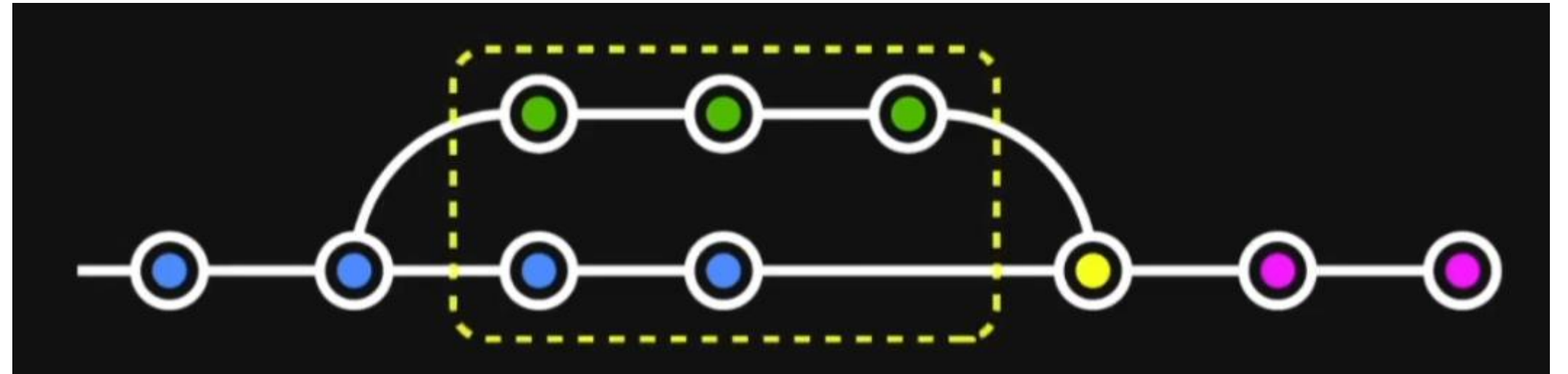
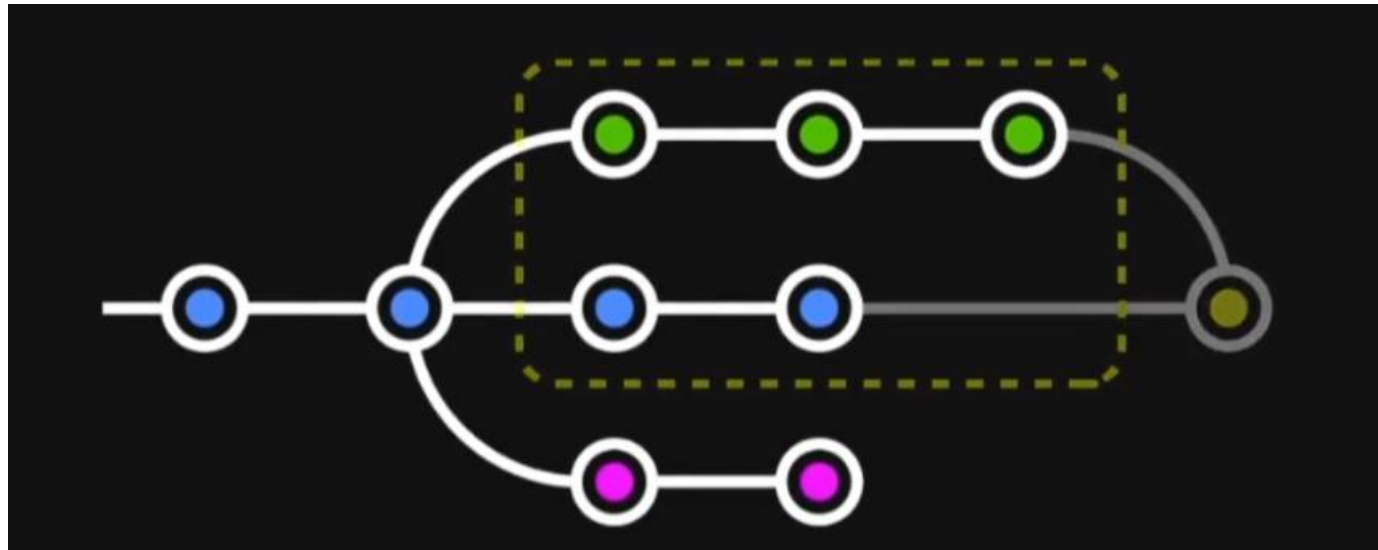
```
DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (new-teams)
$ git log
commit 48065c678b6916e8eeb4b1532cf991919d622916 (HEAD -> new-teams)
Author: Steve Choi <stechoi2015@gmail.com>
Date: Mon Jul 1 18:59:30 2024 +0900

    Add team Jaguars
```

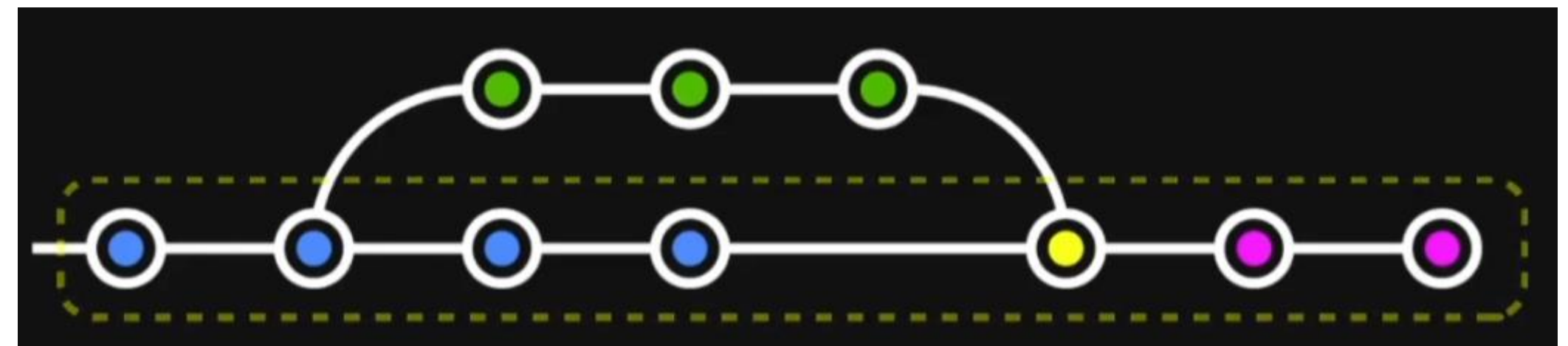
```
DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (new-teams)
● $ git log --all --decorate --oneline --graph
* 48065c6 (HEAD -> new-teams) Add team Jaguars
* d255528 Add team Pumas
| * 249f17c (add-coach) Add Coach Teddy to Panthers
| * 50093f1 Add Coach Oscar to Leopards
| * 051105f Add Coach Grace to Tigers
|/
| * a877c43 (main) Add Freddie to Panthers
| * c34a88f Add Oliver to Leopards
|/
* 5c7dc2a Replace Cheetas with Panthers
* 3a82dcc Add team Cheetas
* f6f00bb Add George to Tigers
* d7856a5 Replace Lions with Leopards
* a294563 FIRST COMMIT
```


서로 다른 브랜치를 합치는 두 방식

- merge : 두 브랜치를 한 커밋에 이어붙입니다.
 - 브랜치 사용내역을 남길 필요가 있을 때 적합한 방식



- rebase : 브랜치를 다른 브랜치에 이어붙입니다.
 - 한 줄로 깔끔히 정리된 내역을 유지하기 원할 때 적합
 - 이미 팀원과 공유된 커밋들에 대해서는 사용하지 않는 것이 좋음



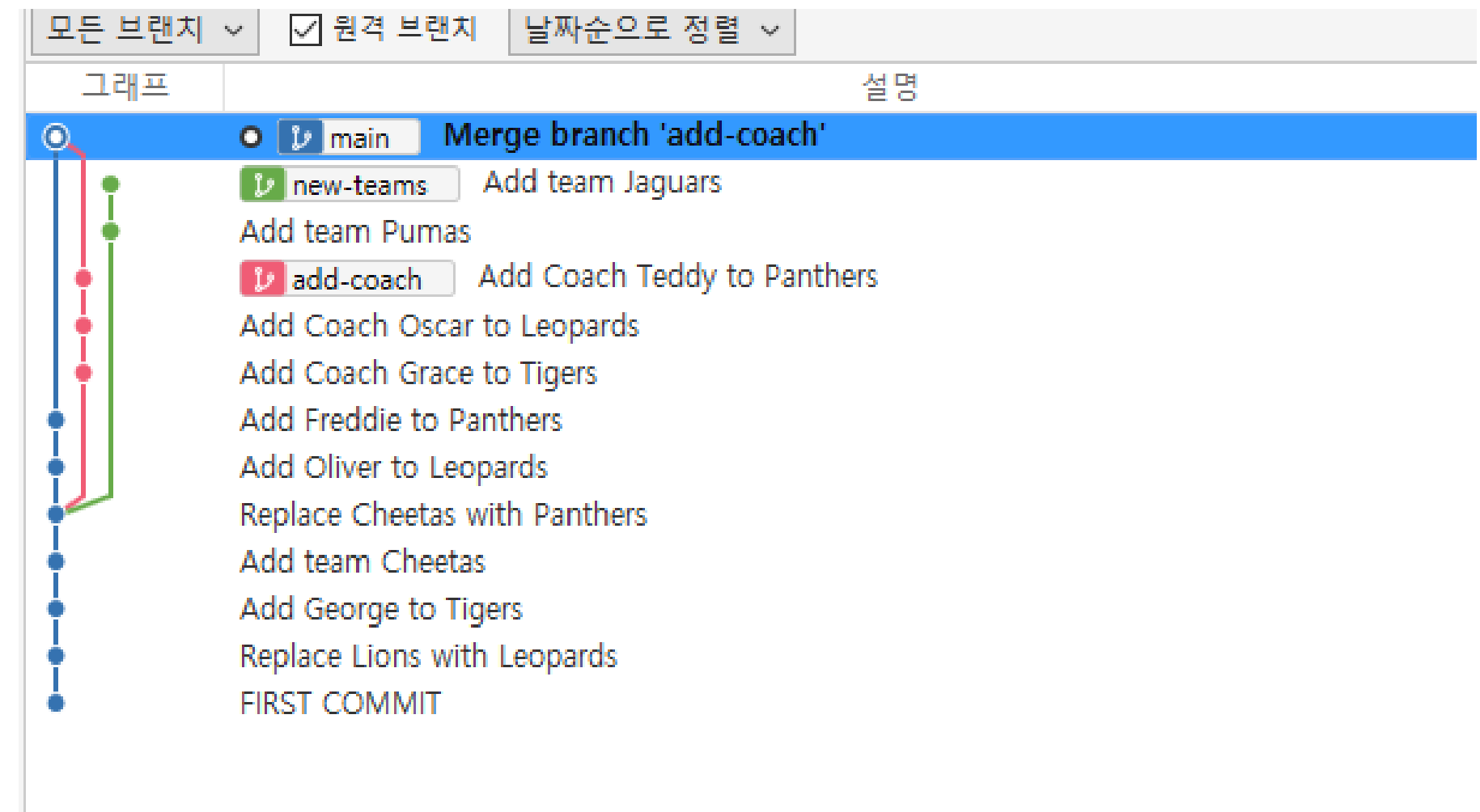
실습(서로다른브랜치를합치는두방식)

1. merge로 합치기

1) add-coach 브랜치를 main 브랜치로 merge

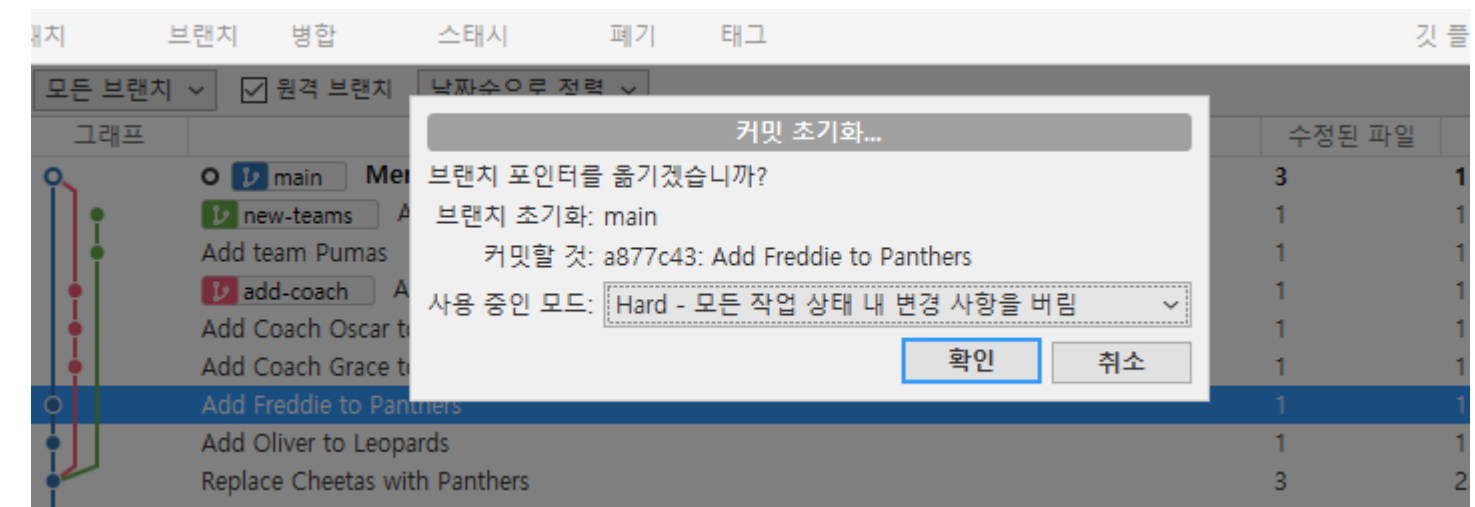
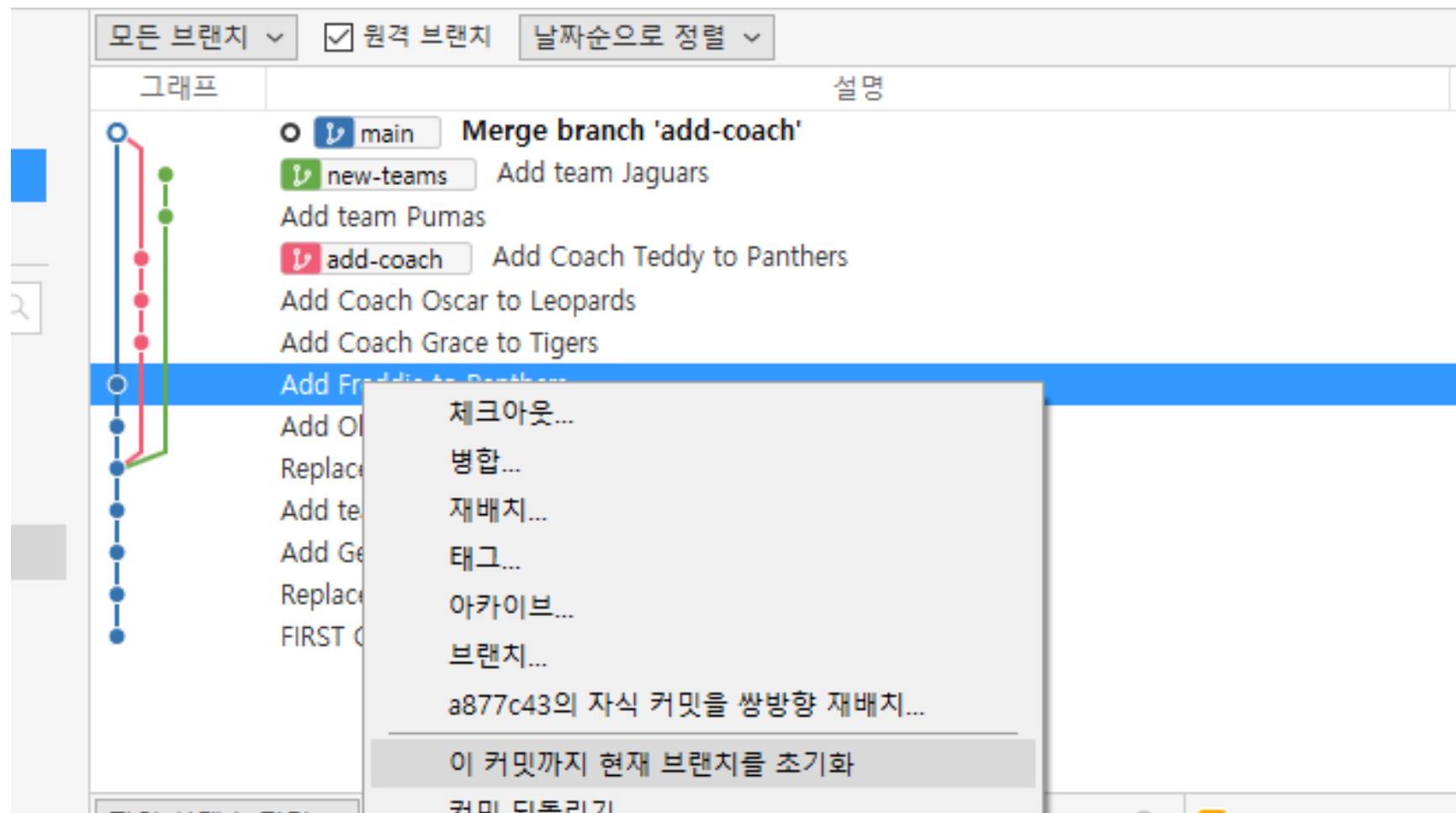
- main 브랜치로 이동
- git merge add-coach

```
DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (main)
● $ git merge add-coach
Auto-merging leopards.yaml
Auto-merging panthers.yaml
Merge made by the 'ort' strategy.
 leopards.yaml | 2 ++
 panthers.yaml | 2 ++
 tigers.yaml   | 2 ++
 3 files changed, 6 insertions(+)
```



실습(서로다른브랜치를합치는두방식)

2) merge 전으로 복원(reset)



실습(서로다른브랜치를합치는두방식)

3) add-coach 브랜치를 main 브랜치로 merge 후 add-coach 브랜치 삭제

- main 브랜치로 이동
- git branch -d add-coach

```
DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (main)
● $ git branch -d add-coach
Deleted branch add-coach (was 249f17c).

DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (main)
● $ git branch
* main
  new-teams
```

실습(서로다른브랜치를합치는두방식)

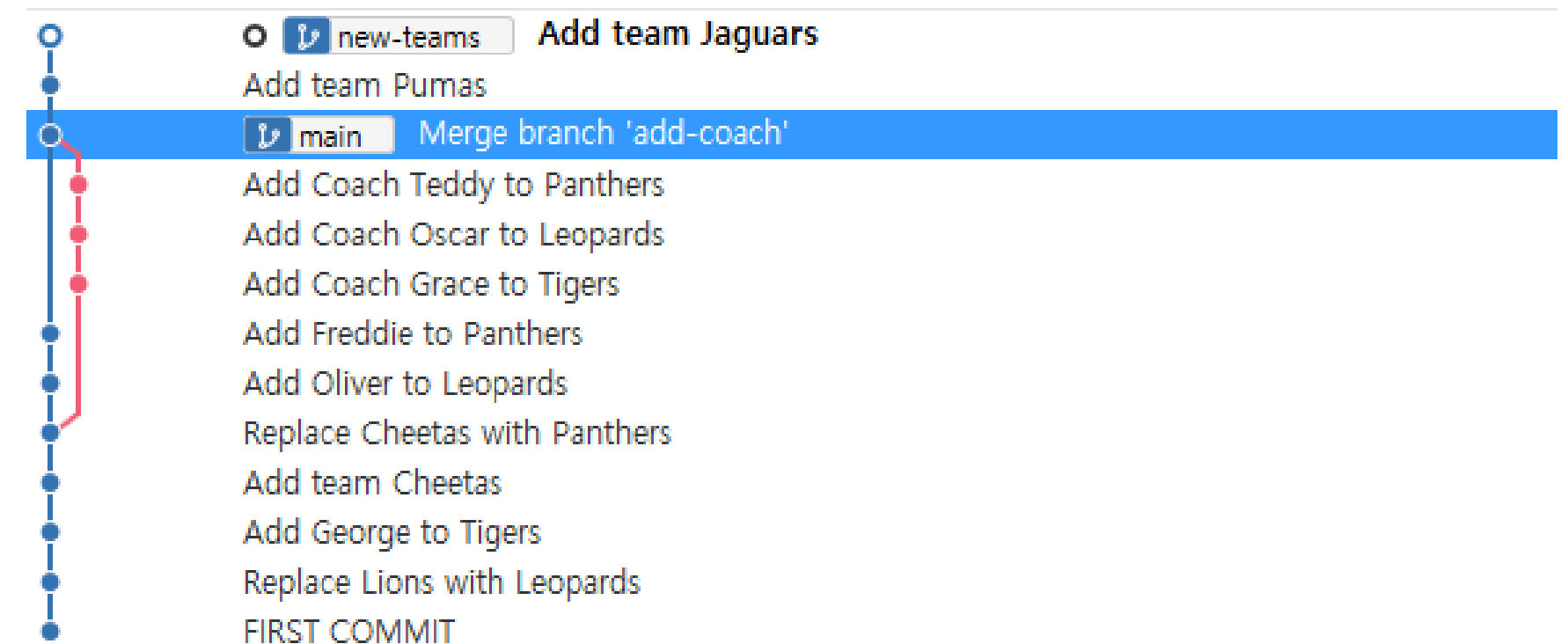
2. rebase로 합치기

1) new-teams 브랜치를 main 브랜치로 rebase

- new-teams 브랜치로 이동(merge와 반대)
- git rebase main

```
DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (main)
• $ git switch new-teams
  Switched to branch 'new-teams'

DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (new-teams)
• $ git rebase main
  Successfully rebased and updated refs/heads/new-teams.
```



2) main 브랜치로 이동 후 아래 명령어로 new-teams의 시점으로 fast-forward

- git merge new-teams

```
DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (new-teams)
• $ git switch main
  Switched to branch 'main'

DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (main)
• $ git merge new-teams
  Updating 2f397ae..ec6f315
  Fast-forward
   jaguars.yaml | 8 ++++++
   pumas.yaml   | 8 ++++++
  2 files changed, 16 insertions(+)
  create mode 100644 jaguars.yaml
  create mode 100644 pumas.yaml
```

실습(서로다른브랜치를합치는두방식)

2) main 브랜치로 이동 후 아래 명령어로 new-teams의 시점으로 fast-forward

- git merge new-teams

```
DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (new-teams)
• $ git switch main
Switched to branch 'main'

DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (main)
• $ git merge new-teams
Updating 2f397ae..ec6f315
Fast-forward
 jaguars.yaml | 8 ++++++
 pumas.yaml   | 8 ++++++
 2 files changed, 16 insertions(+)
 create mode 100644 jaguars.yaml
 create mode 100644 pumas.yaml
```

3) new-teams 브랜치 삭제

```
DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (main)
• $ git branch -d new-teams
Deleted branch new-teams (was ec6f315).
```

실습(브랜치간 충돌 해결하기)

- 파일의 같은 위치에 다른 내용이 입력된 상황

1. 상황 만들기

1) conflict-1, conflict-2 브랜치 생성

```
DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (main)
● $ git branch conflict-1

DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (main)
● $ git branch conflict-2

DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (main)
● $ git branch
  conflict-1
  conflict-2
* main
```

2) main 브랜치

- Tigers의 manager를 Kenneth로 변경
- Leopards의 coach를 Nicholas로 변경
- Panthers의 coach를 Shirley로 변경
- 커밋 메시지: Edit Tigers, Leopards, Panthers

```
DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (main)
● $ git commit -am 'Edit Tigers, Leopards, Panthers'
[main 6556630] Edit Tigers, Leopards, Panthers
 3 files changed, 3 insertions(+), 3 deletions(-)
```

실습(브랜치간 충돌 해결하기)

1) conflict-1 브랜치

- Tigers의 manager를 Deborah로 변경
- 커밋 메시지: Edit Tigers

```
DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (conflict-1)
• $ git commit -am 'Edit Tigers'
[conflict-1 abbc94c] Edit Tigers
1 file changed, 1 insertion(+), 1 deletion(-)
```

2) conflict-2 브랜치 1차

- Leopards의 coach를 Melissa로 변경
- 커밋 메시지: Edit Leopards

```
DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (conflict-2)
Ⓢ $ git commit -am 'Edit Leopards'
On branch conflict-2
nothing to commit, working tree clean
```

3) conflict-2 브랜치 2차

- Panthers의 coach를 Raymond로 변경
- 커밋 메시지: Edit Panthers

```
DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (conflict-2)
• $ git commit -am 'Edit Panthers'
[conflict-2 b3ea74b] Edit Panthers
1 file changed, 1 insertion(+), 1 deletion(-)
```


실습(브랜치간 충돌 해결하기)

2. merge 충돌 해결하기

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

```
DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (main)
⊗ $ git merge conflit-1
Auto-merging tigers.yaml
CONFLICT (content): Merge conflict in tigers.yaml
Automatic merge failed; fix conflicts and then commit the result.
```

```
! tigers.yaml
1  team: Tigers
2
   Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
3  <<<<<< HEAD (Current Change)
4  manager: Kenneth
5  =====
6  manager: Deborah
7  >>>>>> conflit-1 (Incoming Change)
8
9  coach: Grace
```

2) 당장 충돌 해결이 어려울 경우 아래 명령어로 merge 중단

- git merge --abort

```
DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (main|MERGING)
● $ git merge --abort
```

실습(브랜치간 충돌 해결하기)

3) 해결 가능 시 충돌 부분을 수정한 뒤 git add ., git commit으로 병합 완료

```
DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (main)
$ git merge conflit-1
Auto-merging tigers.yaml
CONFLICT (content): Merge conflict in tigers.yaml
Automatic merge failed; fix conflicts and then commit the result.

DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (main|MERGING)
$ git add .

DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (main|MERGING)
$ git commit
[main 9a1b72c] Merge branch 'conflit-1'
```

실습(브랜치간 충돌 해결하기)

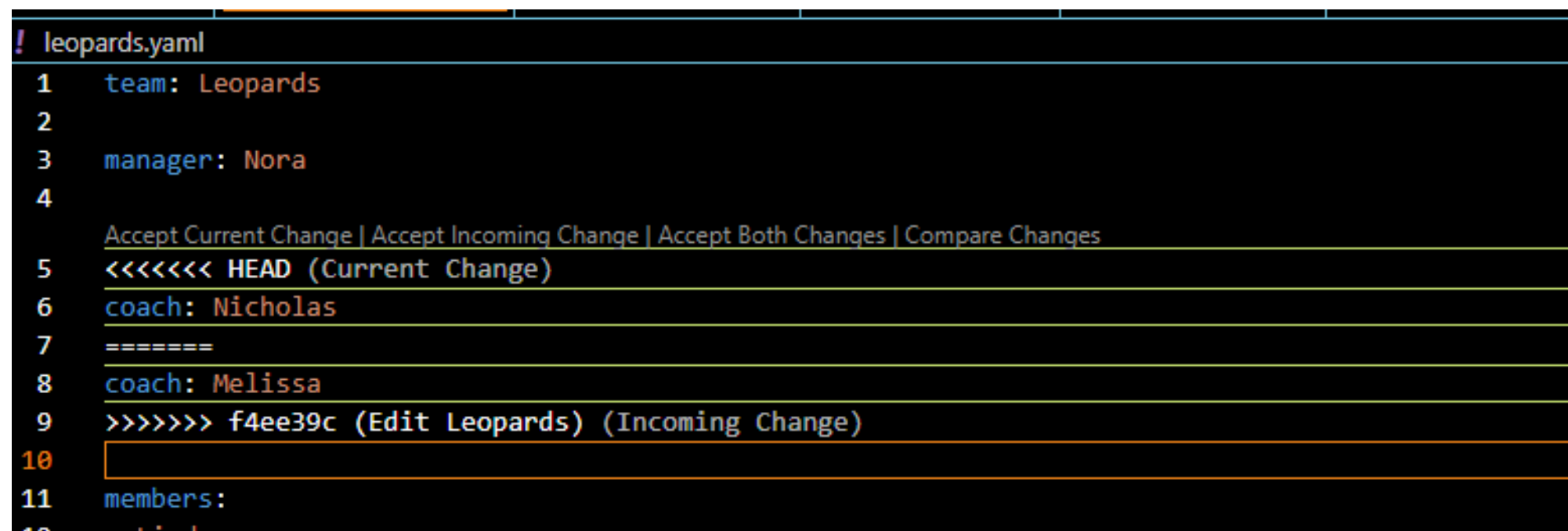
3. rebase 충돌 해결하기

1) conflict-2에서 git rebase main로 리베이스 시도하면 충돌 발생

- conflict-2 브랜치로 이동
- git rebase main 실행
- 오류 메시지와 git status 확인

```
DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (conflict-2)
$ git rebase main
Auto-merging leopards.yaml
CONFLICT (content): Merge conflict in leopards.yaml
error: could not apply f4ee39c... Edit Leopards
hint: Resolve all conflicts manually, mark them as resolved with
hint: "git add/rm <conflicted_files>", then run "git rebase --continue".
hint: You can instead skip this commit: run "git rebase --skip".
hint: To abort and get back to the state before "git rebase", run "git rebase --abort".
Could not apply f4ee39c... Edit Leopards
```

- VS Code에서 해당 부분 확인



```
! leopards.yaml
1 team: Leopards
2
3 manager: Nora
4
5 Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
6 <<<<<<< HEAD (Current Change)
7 coach: Nicholas
8 =====
9 >>>>>> f4ee39c (Edit Leopards) (Incoming Change)
10 coach: Melissa
11 members:
12
```

```
DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (conflict-2|REBASE 1/2)
$ git status
interactive rebase in progress; onto 9a1b72c
Last command done (1 command done):
  pick f4ee39c Edit Leopards
Next command to do (1 remaining command):
  pick b3ea74b Edit Panthers
(use "git rebase --edit-todo" to view and edit)
You are currently rebasing branch 'conflict-2' on '9a1b72c'.
(fix conflicts and then run "git rebase --continue")
(use "git rebase --skip" to skip this patch)
(use "git rebase --abort" to check out the original branch)

Unmerged paths:
  (use "git restore --staged <file>..." to unstage)
  (use "git add <file>..." to mark resolution)
    both modified:   leopards.yaml

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

실습(브랜치간 충돌 해결하기)

3. rebase 충돌 해결하기

2) 당장 충돌 해결이 어려울 경우 아래 명령어로 rebase 중단

- git rebase --abort

3) 해결 가능 시

- 충돌 부분을 수정한 뒤 git add .
- git rebase --continue
- 충돌이 모두 해결될 때까지 반복

```
DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (conflict-2|REBASE 1/2)
$ git add .

DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (conflict-2|REBASE 1/2)
$ git rebase --continue
[detached HEAD a6fbccd] Edit Leopards
1 file changed, 1 insertion(+), 1 deletion(-)
Auto-merging panthers.yaml
CONFLICT (content): Merge conflict in panthers.yaml
error: could not apply b3ea74b... Edit Panthers
hint: Resolve all conflicts manually, mark them as resolved with
hint: "git add/rm <conflicted_files>", then run "git rebase --continue".
hint: You can instead skip this commit: run "git rebase --skip".
hint: To abort and get back to the state before "git rebase", run "git rebase --abort".
Could not apply b3ea74b... Edit Panthers
```

```
! panthers.yaml
1  team: Panthers
2
3  manager: Sebastian
4
5  Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
6  <<<<<< HEAD (Current Change)
7  coach: Shirley
8  =====
9  >>>>>> b3ea74b (Edit Panthers) (Incoming Change)
10 coach: Raymond
```

실습(브랜치간 충돌 해결하기)

```
DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (conflict-2|REBASE 2/2)
$ git add .

DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (conflict-2|REBASE 2/2)
$ git rebase --continue
[detached HEAD 22efde7] Edit Panthers
1 file changed, 1 insertion(+), 1 deletion(-)
Successfully rebased and updated refs/heads/conflict-2.
```

3) main에서 git merge conflict-2로 마무리

```
DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (main)
• $ git merge conflict-2
Updating 9a1b72c..22efde7
Fast-forward
 leopards.yaml | 2 +-
 panthers.yaml | 2 +-
 2 files changed, 2 insertions(+), 2 deletions(-)
```

4) conflict-1, conflict-2 삭제

```
DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (main)
• $ git branch -d conflict-1
Deleted branch conflict-1 (was de8c568).

DELL@DESKTOP-EGSH5H0 MSYS /h/Git_practise (main)
• $ git branch -d conflict-2
Deleted branch conflict-2 (was 22efde7).
```

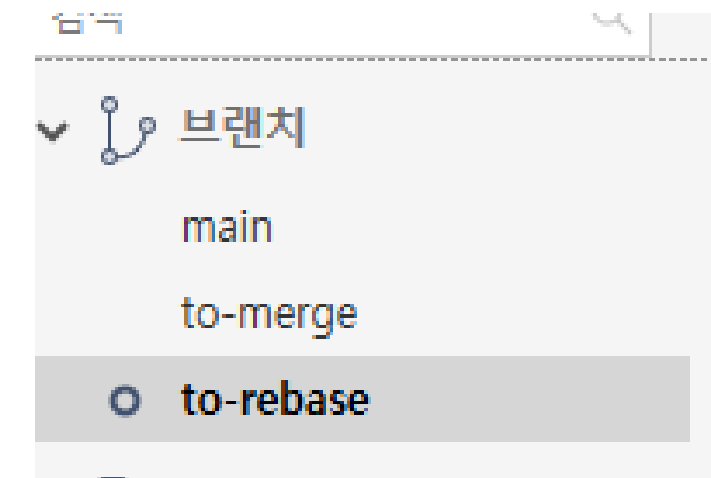
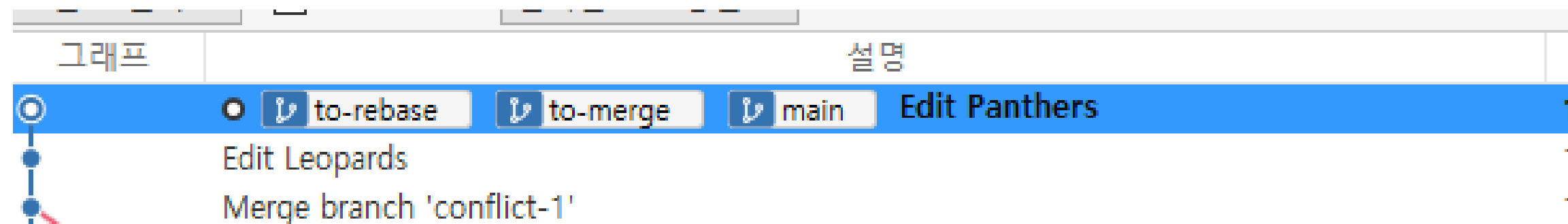
Sourcetree로 진행하기

실습(sourcetree에서 진행하기)

1. 브랜치 만들고 merge, rebase 하기

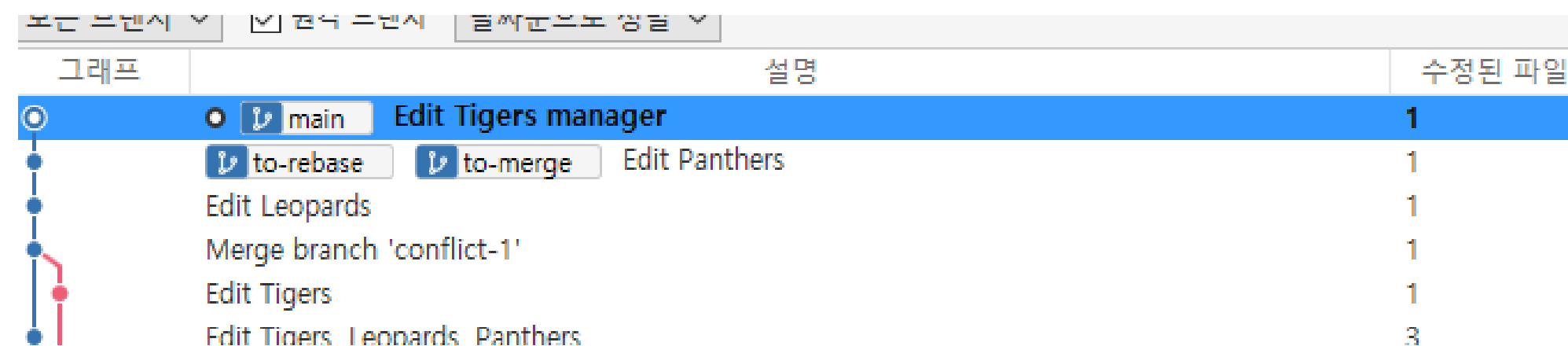
1) to-merge, to-rebase 브랜치 생성

- 상단의 브랜치 버튼 클릭
- 왼쪽의 브랜치 탭에서 클릭하여 이동



2) main 브랜치

- Tigers의 manager를 Brenda로 변경
- 커밋 메시지: Edit Tigers manager



실습(sourcetree에서 진행하기)

- 3) to-merge 브랜치
- Tigers의 coach를 Ruth로 변경
 - 커밋 메시지: Edit Tigers coach

모든 브랜치 ▾			<input checked="" type="checkbox"/> 원격 브랜치	날짜순으로 정렬 ▾
그래프		설명	수정된 파일	
	to-merge	Edit Tigers manager	1	
	main	Edit Tigers manager	1	
	to-rebase	Edit Panthers	1	
	Edit Leonards		1	

- 4) to-rebase 브랜치
- Tigers의 memebers에 Tyler 추가
 - 커밋 메시지: Edit Tigers members

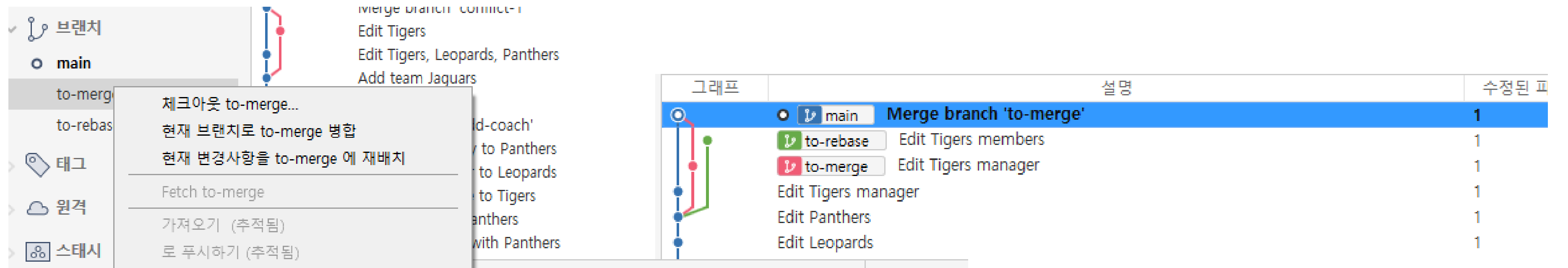
모든 브랜치 ▾			<input checked="" type="checkbox"/> 원격 브랜치	날짜순으로 정렬 ▾
그래프		설명	수정된 파일	날짜
	to-rebase	Edit Tigers members	1	1 7 2024 22:00
	to-merge	Edit Tigers manager	1	1 7 2024 21:00
	main	Edit Tigers manager	1	1 7 2024 21:00
	Edit Panthers		1	22 7 2023 00:00

실습(sourcetree에서 진행하기)

2. 브랜치를 이동하며 파일 살펴보기

1) to-merge 브랜치 main으로 merge

- main에 위치한 뒤 to-merge 브랜치를 우클릭하여 Merge ... 클릭



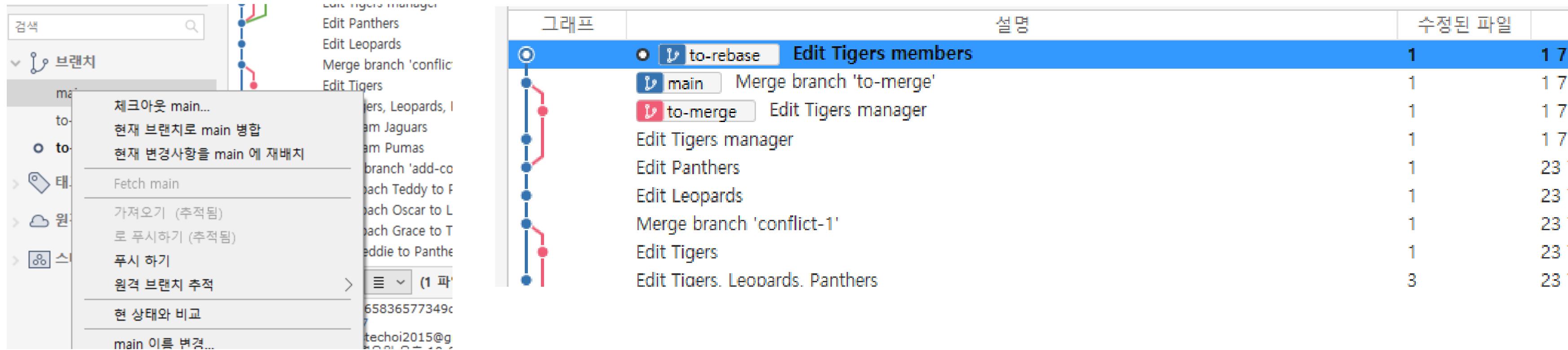
2) main 브랜치

- Tigers의 manager를 Brenda로 변경
- 커밋 메시지: Edit Tigers manager

실습(sourcetree에서 진행하기)

2) to-rebase 브랜치 main으로 rebase

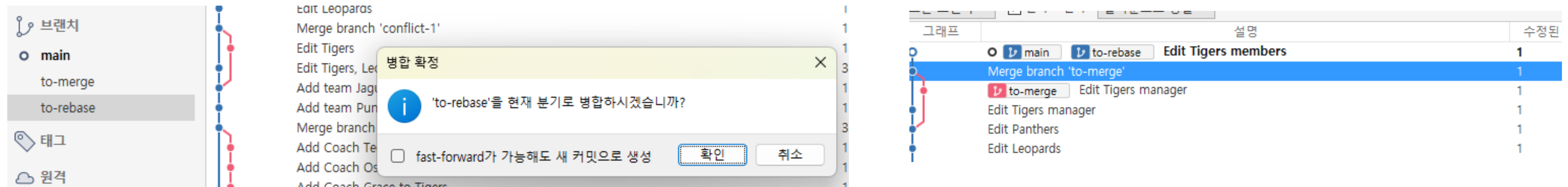
- to-rebase에 위치한 뒤 main 브랜치를 우클릭하여 ... 재배치 클릭



The screenshot shows the Sourcetree interface. On the left, the 'branches' panel shows 'to-rebase' selected. A context menu is open over 'to-rebase' with options like 'Fetch main', '가져오기 (추적됨)', '로 푸시하기 (추적됨)', '푸시 하기', '원격 브랜치 추적', '현 상태와 비교', and 'main 이름 변경...'. On the right, the 'Commit' panel shows a list of changes with columns '그래프', '설명', '수정된 파일', and '수정된 라인'. The first change is 'to-rebase Edit Tigers members' with 1 file changed and 17 lines added.

그래프	설명	수정된 파일	수정된 라인
to-rebase	Edit Tigers members	1	17
main	Merge branch 'to-merge'	1	17
to-merge	Edit Tigers manager	1	17
	Edit Tigers manager	1	17
	Edit Panthers	1	23
	Edit Leopards	1	23
	Merge branch 'conflict-1'	1	23
	Edit Tigers	1	23
	Edit Tigers, Leopards, Panthers	3	23

- main에 위치한 뒤 to-rebase 브랜치를 우클릭하여 Merge ... 클릭

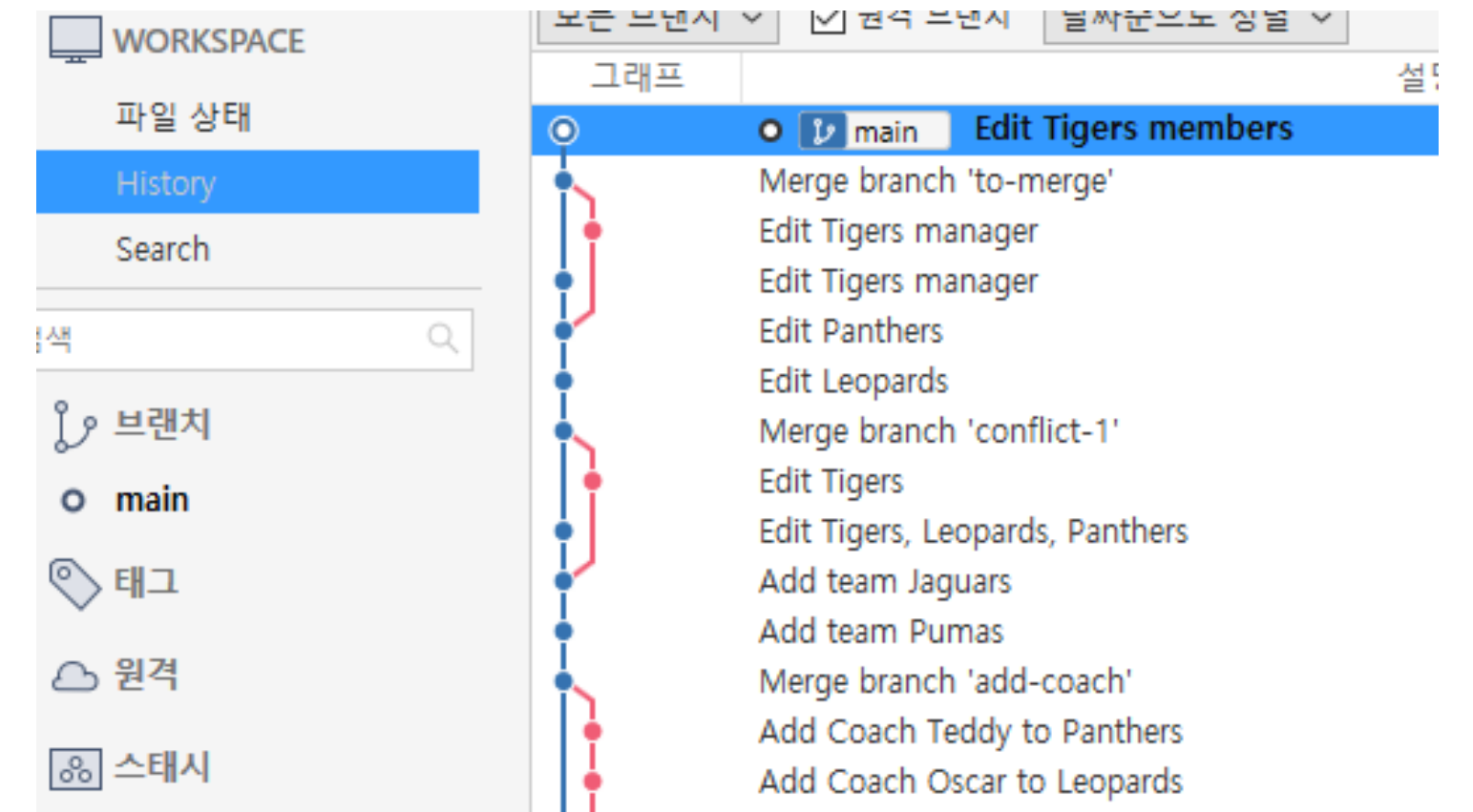
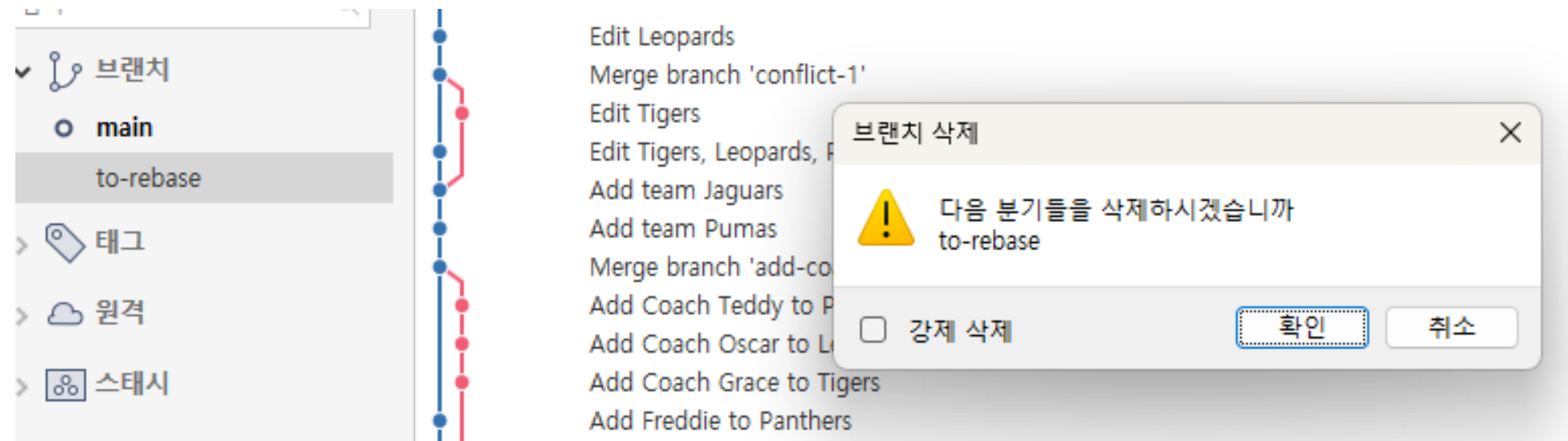
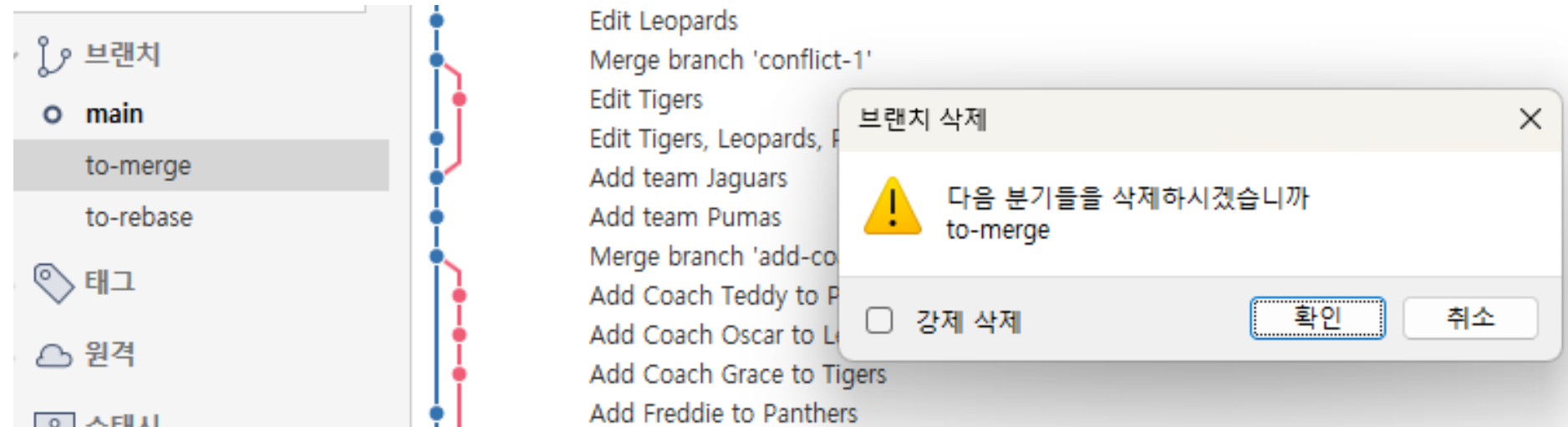


The screenshot shows the Sourcetree interface. On the left, the 'branches' panel shows 'main' selected. A 'Merge Confirmation' dialog box is open with the message: "'to-rebase'를 현재 분기로 병합하시겠습니까?". There is a checkbox for 'fast-forward가 가능해도 새 커밋으로 생성' and buttons for '확인' (Confirm) and '취소' (Cancel). On the right, the 'Commit' panel shows a list of changes with columns '그래프', '설명', and '수정된'. The first change is 'main to-rebase Edit Tigers members' with 1 file changed.

그래프	설명	수정된
main	to-rebase Edit Tigers members	1
	Merge branch 'to-merge'	1
to-merge	Edit Tigers manager	1
	Edit Tigers manager	1
	Edit Panthers	1
	Edit Leopards	1

실습(sourcetree에서 진행하기)

3) main으로 이동 후 to-merge와 to-rebase 우클릭하여 삭제



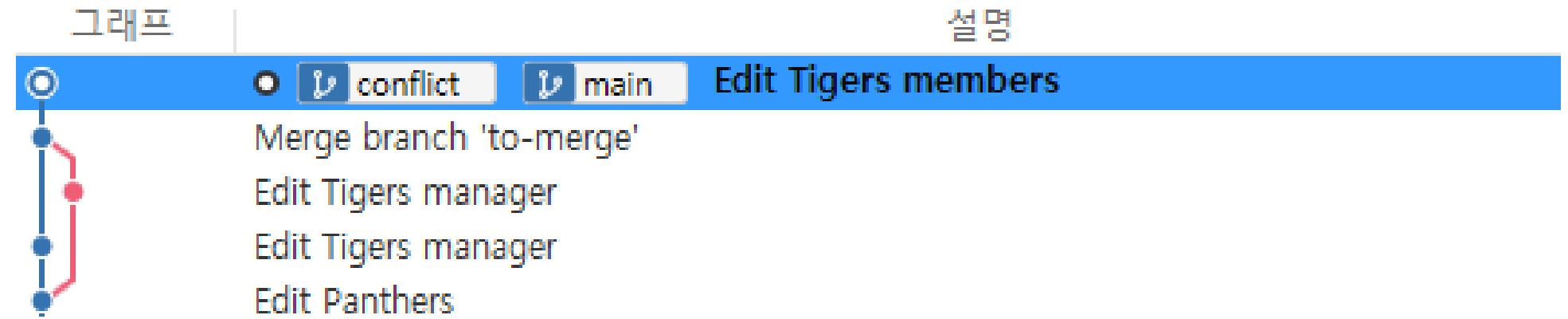
실습(sourcetree에서 진행하기)

2. merge 충돌 해결해보기

rebase는 충돌 가능시 CLI로 진행 권장

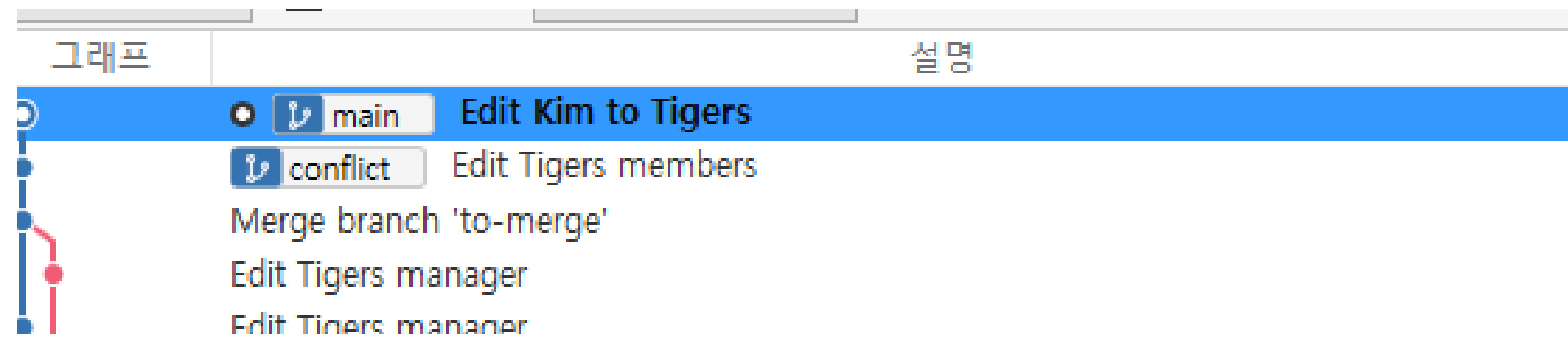
1) conflict 브랜치 생성

- 상단의 브랜치 버튼 클릭
- 왼쪽의 브랜치 탭에서 클릭하여 이동



2) main 브랜치

- Tigers의 members에 Kim 추가
- 커밋 메시지: Edit Kim to Tigers



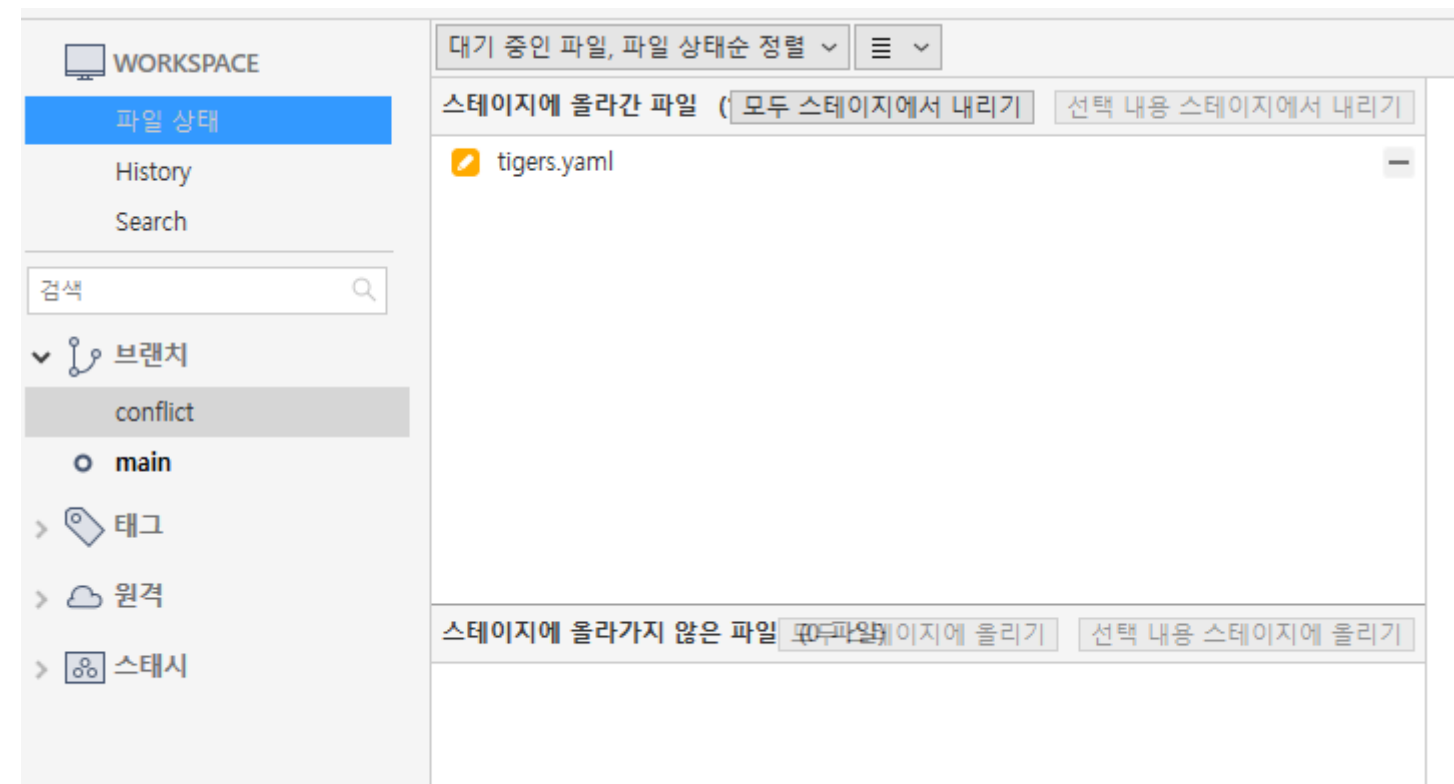
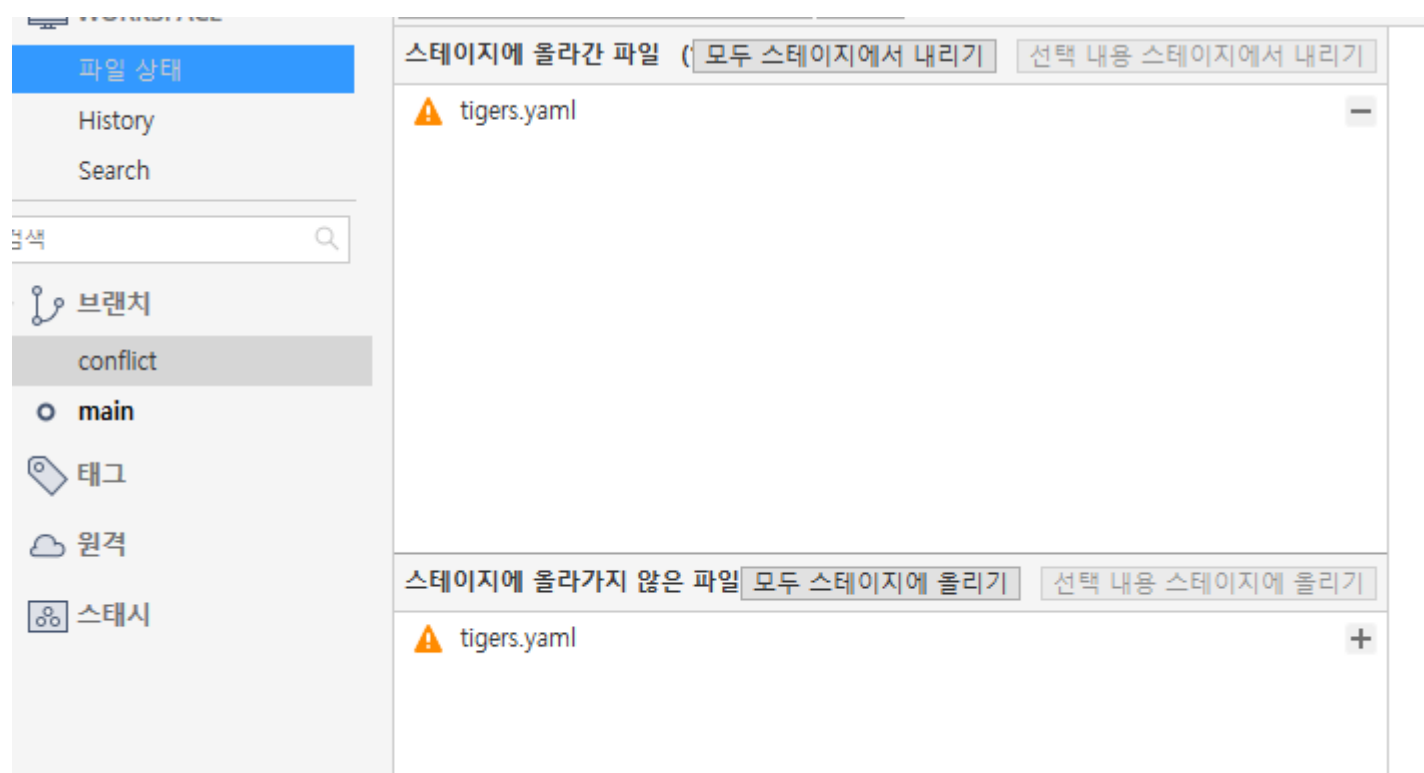
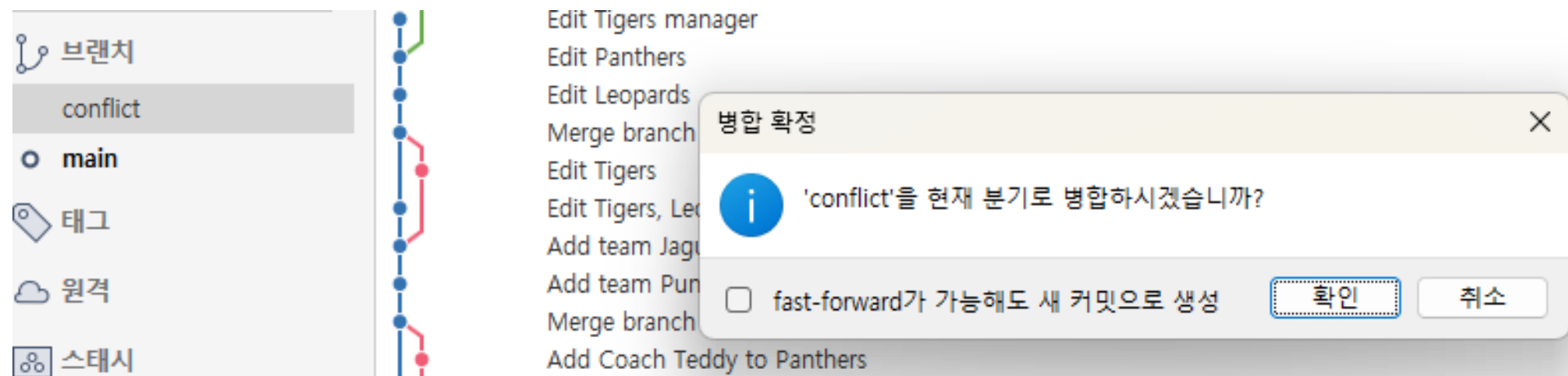
실습(sourcetree에서 진행하기)

3) conflict 브랜치

- Tigers의 members에 Park 추가
- 커밋 메시지: Edit Park to Tigers

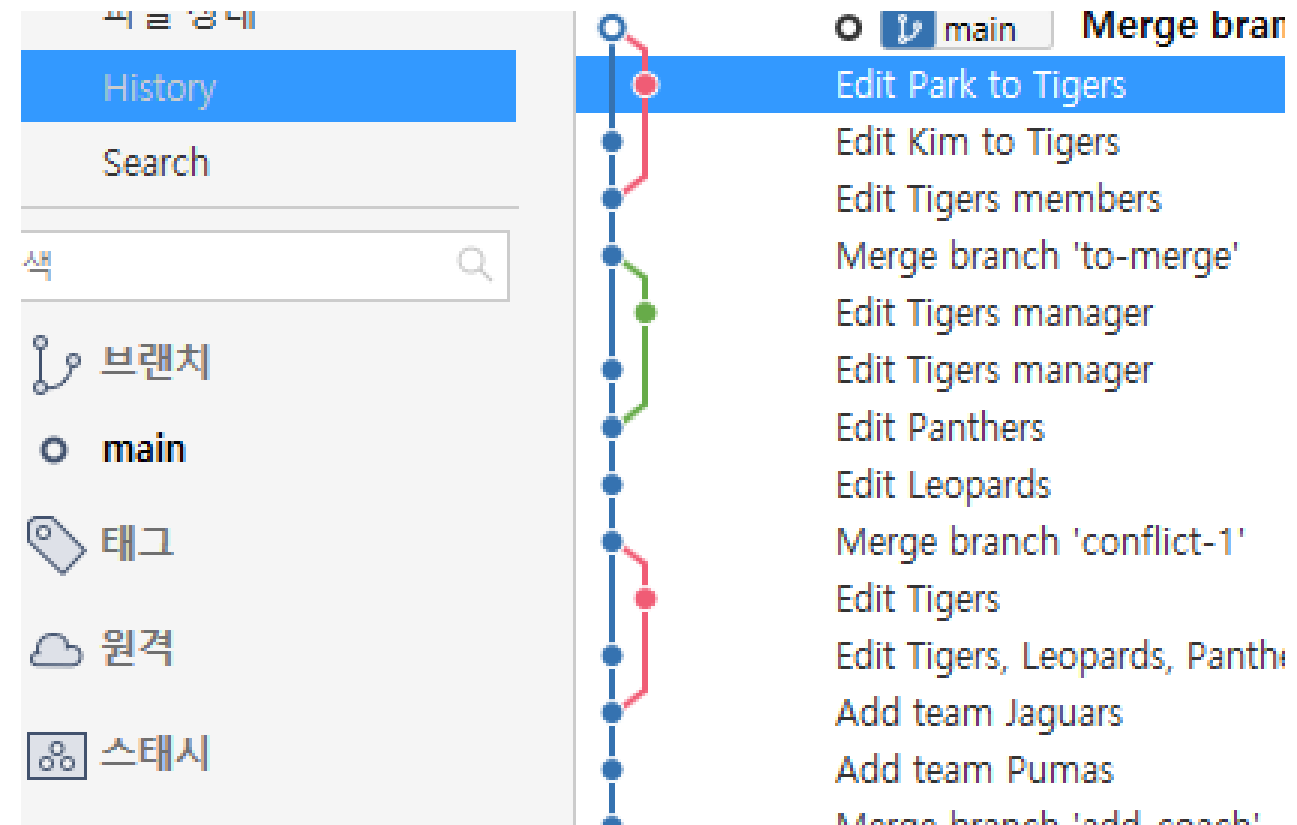
그래프	설명	수정된 파일
	conflict Edit Park to Tigers	1
	main Edit Kim to Tigers	1
	Edit Tigers members	1
	Merge branch 'to-merge'	1

4) merge하여 충돌 해결해보기



실습(sourcetree에서 진행하기)

5) conflict 브랜치 삭제



깃허브 원격 저장소

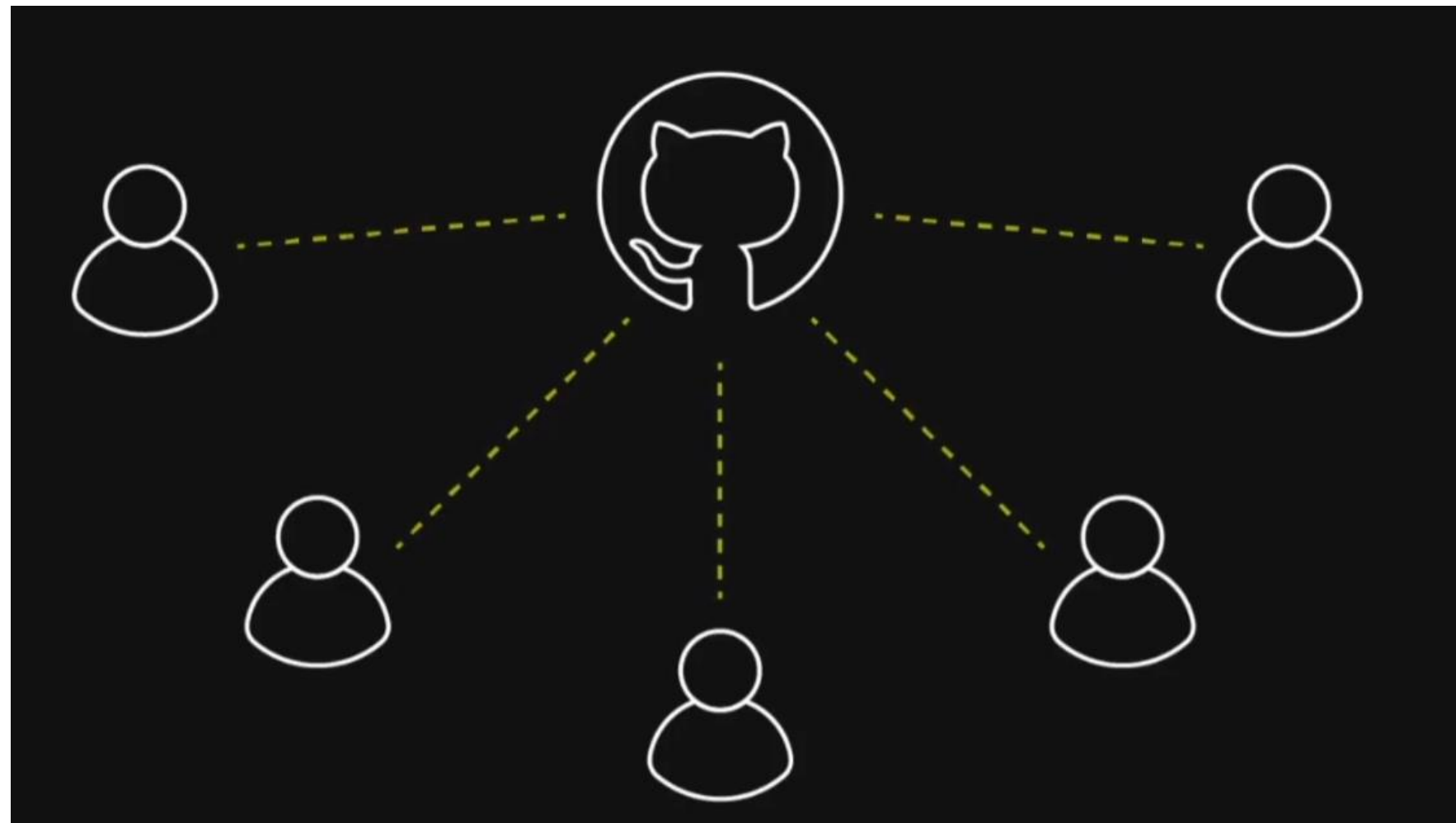
깃허브



깃허브는 깃 저장소 호스팅을 지원하는 웹 서비스이다. 깃허브를 이용하면 온라인 상에 저장소를 만들어 원격으로 이를 관리할 수 있다.

또한, 깃이 명령행 인터페이스를 제공하는데 반해 깃허브는 그래픽 인터페이스를 제공하기 때문에 사용자 입장에서 보다 편리하게 깃 저장소를 관리할 수 있다.

결론! 깃허브는 깃 이용자들에게 편의를 제공하기 위해 존재하는 하나의 플랫폼이다.



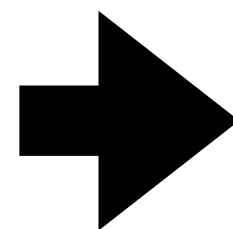
내 PC에서 혼자 작업하는, 로컬 저장소!

온라인에 존재하는, 원격 저장소!

내가 만든 프로젝트 폴더
더 나은 언제나 이곳을
보며 작업을 진행한다

나의 문서 작업은
이 곳에서 이루어진다

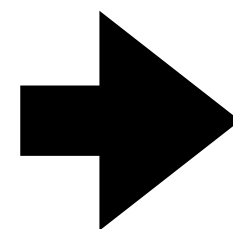
Working Directory



수정 이력을 기록할 때
일을 대기시키는 장소

깃이 관리하는 영역

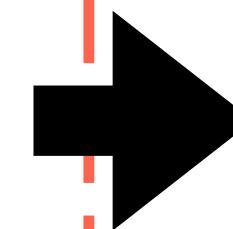
Staging Area



스테이징 영역에서 대기
중이던 파일들의 수정
이력이 최종적으로
기록되는 장소

깃이 관리하는 영역

Repository



웹 상에 존재하는
원격 저장소

로컬 저장소에서 작업한
깃 프로젝트를 공유할
수 있어 편리하다

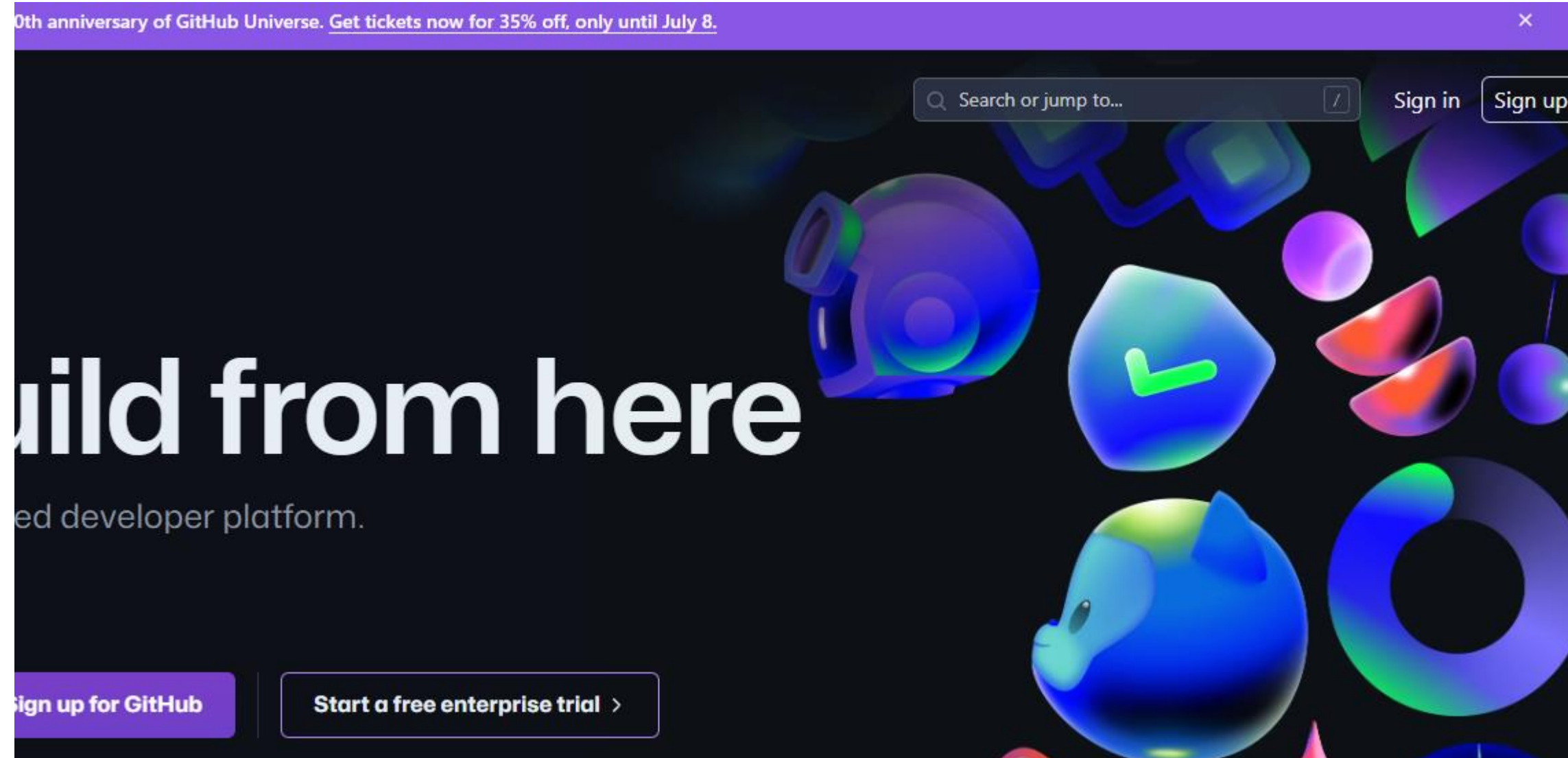
Github remote repository

로컬 저장소에서 일단 작업하고, 원격 저장소 생성하고, 원격 저장소에 로컬 저장소의 내용을 업로드하는 방식을 실습!

깃허브의 시작은 가입으로부터

깃허브는 웹 서비스이고, 계정이 있는 이용자들에게만 서비스를 제공한다. 따라서 깃허브 이용을 위해 가장 먼저 해야 할 일은 ‘회원 가입’이다!

=> <https://github.com/>



가입하고토큰만들기

1. 가입하고 토큰 만들기

1) Sign Up으로 가입 후 로그인

2) Personal access token 만들기

- 우측 상단의 프로필 - Settings

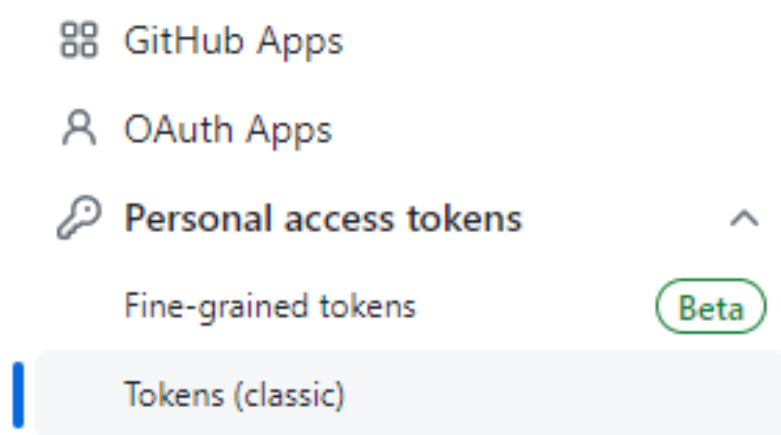
- Developer Settings

 Sponsorship log

 Developer settings

가입하고토큰만들기

- Personal access tokens - Generate new token
- repo 및 원하는 기능에 체크, 기간 설정 뒤 Generate token
- 토큰 안전한 곳에 보관해 둘 것



Personal access tokens (classic)

Need an API token for scripts or testing? [Generate a personal access token](#) for quick access to t

Personal access tokens (classic) function like ordinary OAuth access tokens. They can be used instead of a passw used to [authenticate to the API over Basic Authentication](#).

New personal access token (classic)

Personal access tokens (classic) function like ordinary OAuth access tokens. They can be used instead of a passw over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

Note

my token

What's this token for?

Expiration *

30 days

The token will expire on Thu, Aug 1 2024

Select scopes

Scopes define the access for personal tokens. [Read more about OAuth scopes](#).

- | | |
|---|--------------------------------------|
| <input checked="" type="checkbox"/> repo | Full control of private repositories |
| <input checked="" type="checkbox"/> repo:status | Access commit status |
| <input checked="" type="checkbox"/> repo_deployment | Access deployment status |
| <input checked="" type="checkbox"/> public_repo | Access public repositories |
| <input checked="" type="checkbox"/> repo:invite | Access repository invitations |
| <input checked="" type="checkbox"/> security_events | Read and write security events |

Personal access tokens (classic)

Generate new token ▼

Revoke all

Tokens you have generated that can be used to access the [GitHub API](#).

Make sure to copy your personal access token now. You won't be able to see it again!

✓ ghp_qac7wuhMx6abMumMNWeqAexfmw7EAI0HyMkG

Delete

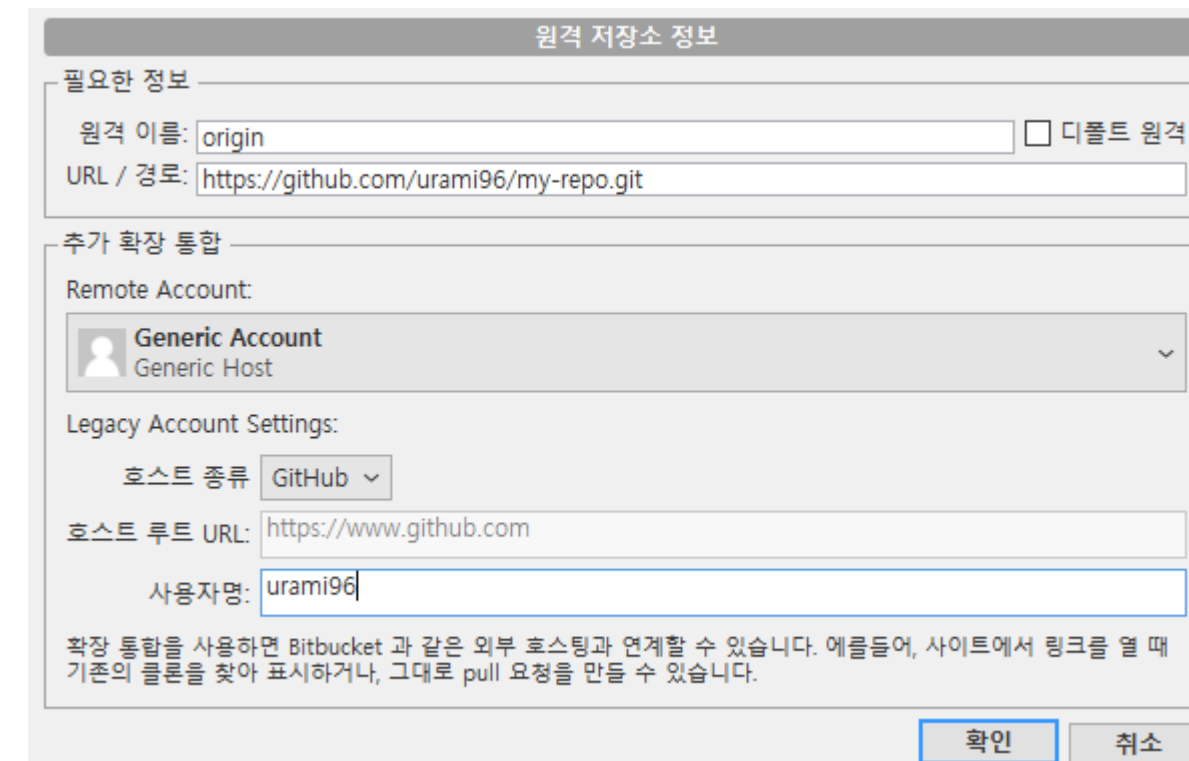
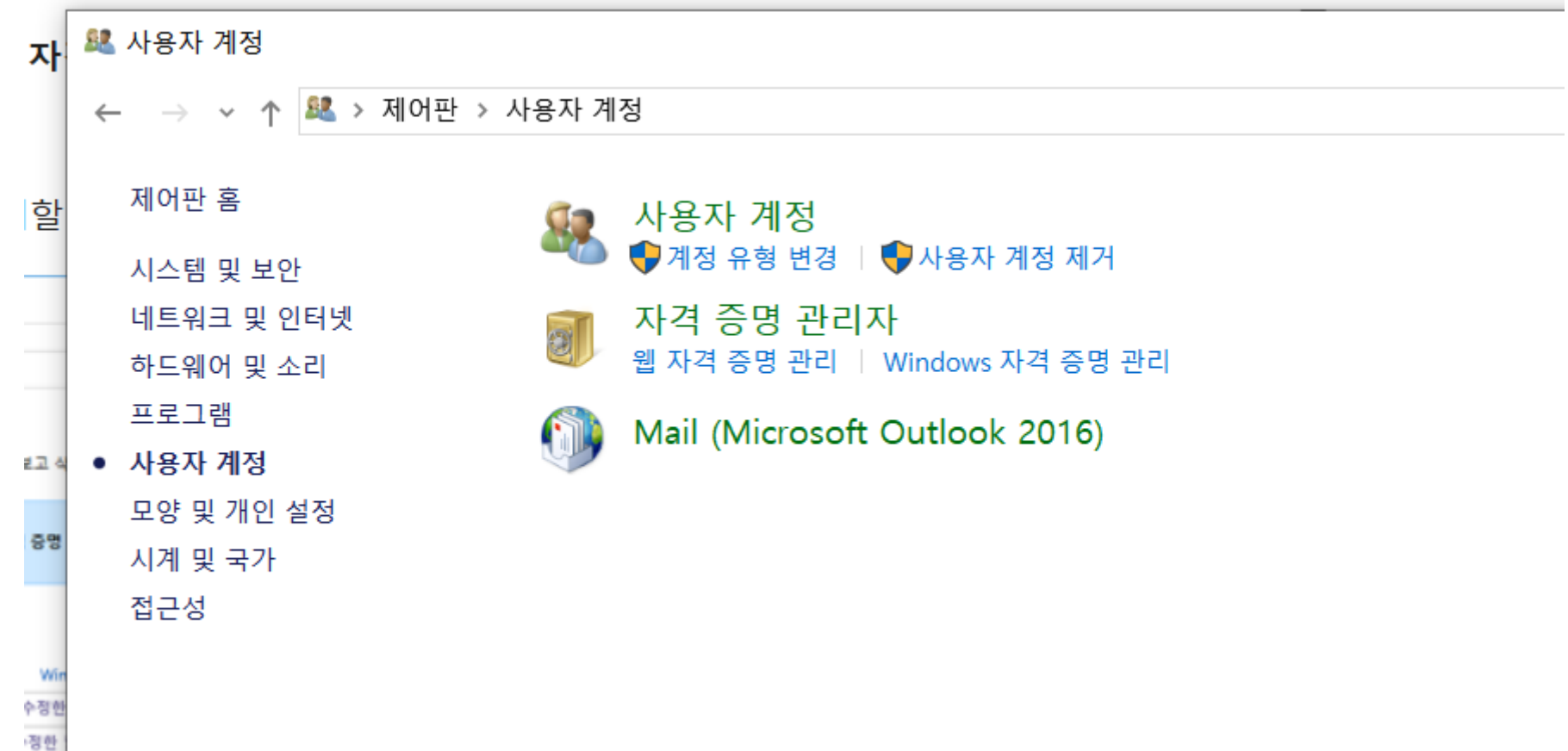
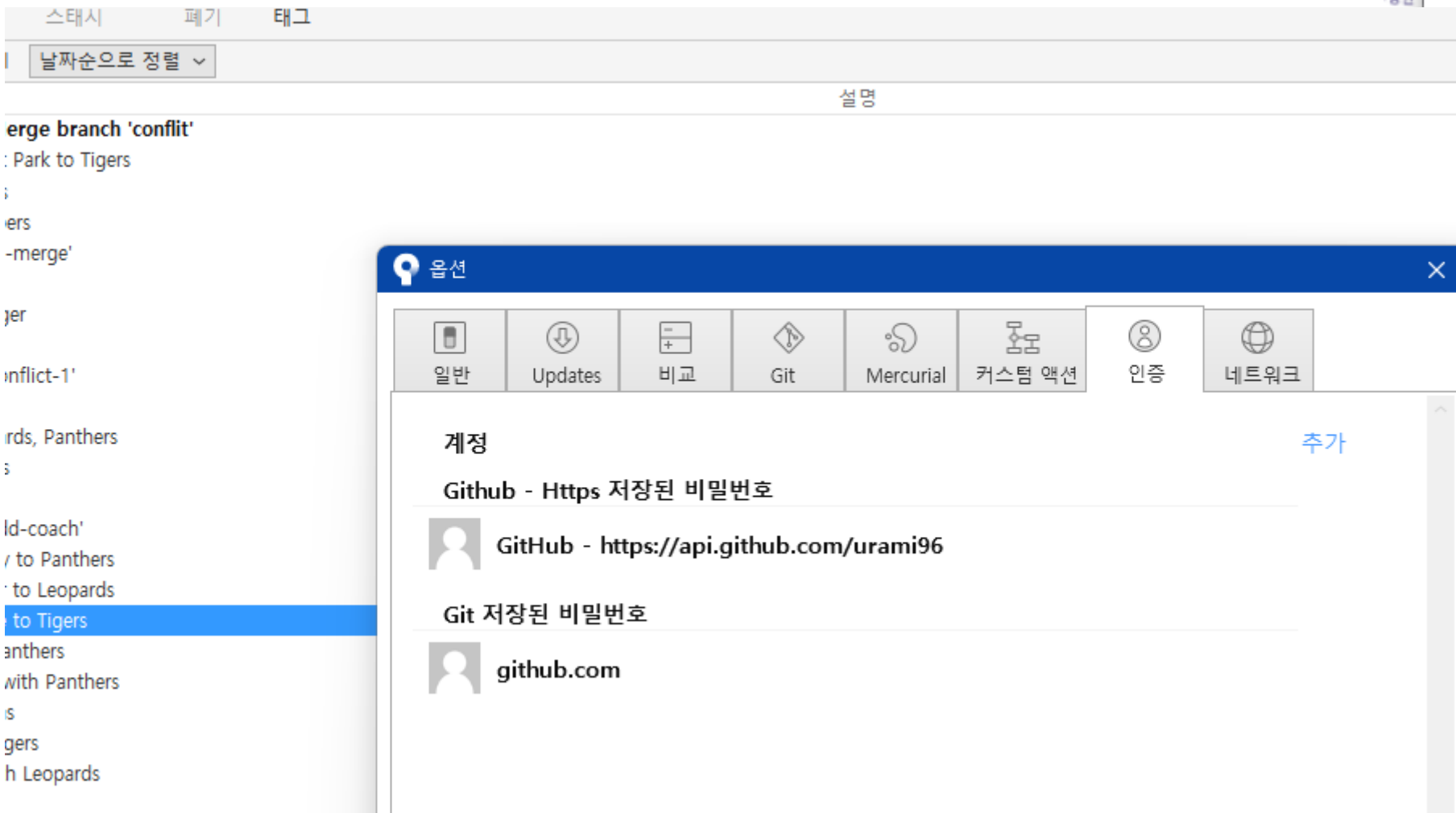
Personal access tokens (classic) function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

가입하고토큰만들기

3. 토큰 컴퓨터에 저장하기

- Windows 자격 증명 관리자
- Windows 자격 증명 선택
- git:https://github.com 자격 정보 생성
- 사용자명과 토큰 붙여넣기

4. sourcetree [도구-옵션-인증]



Repository 만들기

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere?

[Import a repository.](#)

Required fields are marked with an asterisk ().*

Owner *



Polytech12 ▾



Repository name *

git-practice

✔ git-practice is available.

Great repository names are short and memorable. Need inspiration? How about [psychic-sniffle](#) ?

Description (optional)



Public

Anyone on the internet can see this repository. You choose who can commit.



Private

You choose who can see and commit to this repository.

Collaborators

The screenshot shows the GitHub interface for managing repository collaborators. At the top, a navigation bar includes links for Actions, Projects, Wiki, Security, Insights, and Settings. On the left sidebar, the 'Collaborators' option is selected under the 'Access' section. The main content area is titled 'Manage access' and displays a modal dialog for adding a collaborator to the repository 'git-practice'.

Modal Dialog: Add a collaborator to git-practice

Search by username, full name, or email

Find people

Select a collaborator above

Main Content: Manage access

You haven't invited any collaborators yet

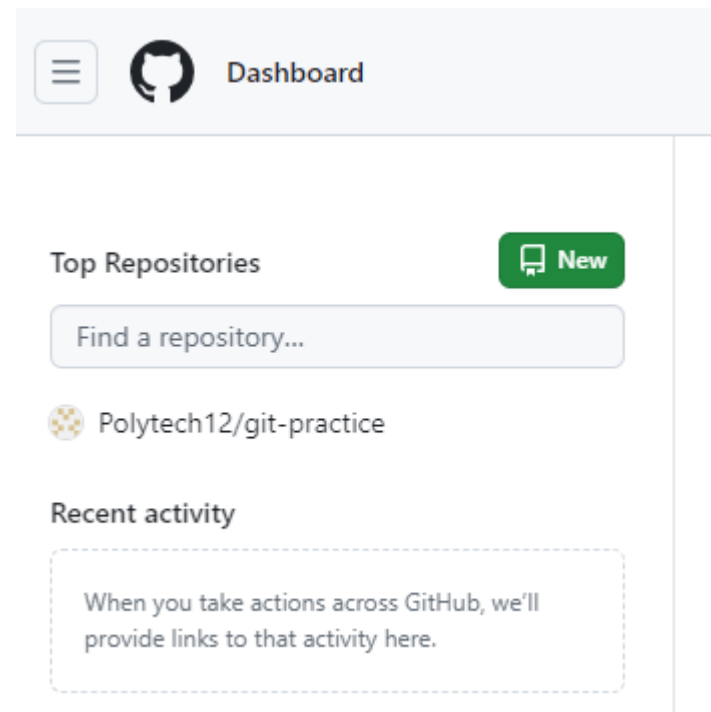
Add people

로컬에 원격 저장소 추가 후 푸시

1. GitHub 레포지토리 생성 후 복붙 명령어

1) git remote add origin (원격 저장소 주소)

- 원격저장소 click 후 ‘...or push an existing repository from the command line’ 아랫줄 copy



...or push an existing repository from the command line

```
git remote add origin https://github.com/Polytech12/git-practice.git
git branch -M main
git push -u origin main
```

- 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 브랜치명을 main으로
- git push -u origin main
 - . 로컬 저장소의 커밋 내역들 원격으로 push(업로드)
 - . -u 또는 --set-upstream : 현재 브랜치와 명시된 원격 브랜치 기본 연결

로컬에 원격 저장소 추가 후 푸시

```
stech@LAPTOP-OE1P7VO2 MINGW64 /d/Git_practise (main)
$ git remote -v
origin https://www.github.com (fetch)
origin https://www.github.com (push)
```

```
stech@LAPTOP-OE1P7VO2 MINGW64 /d/Git_practise (main)
$ git remote remove origin
```

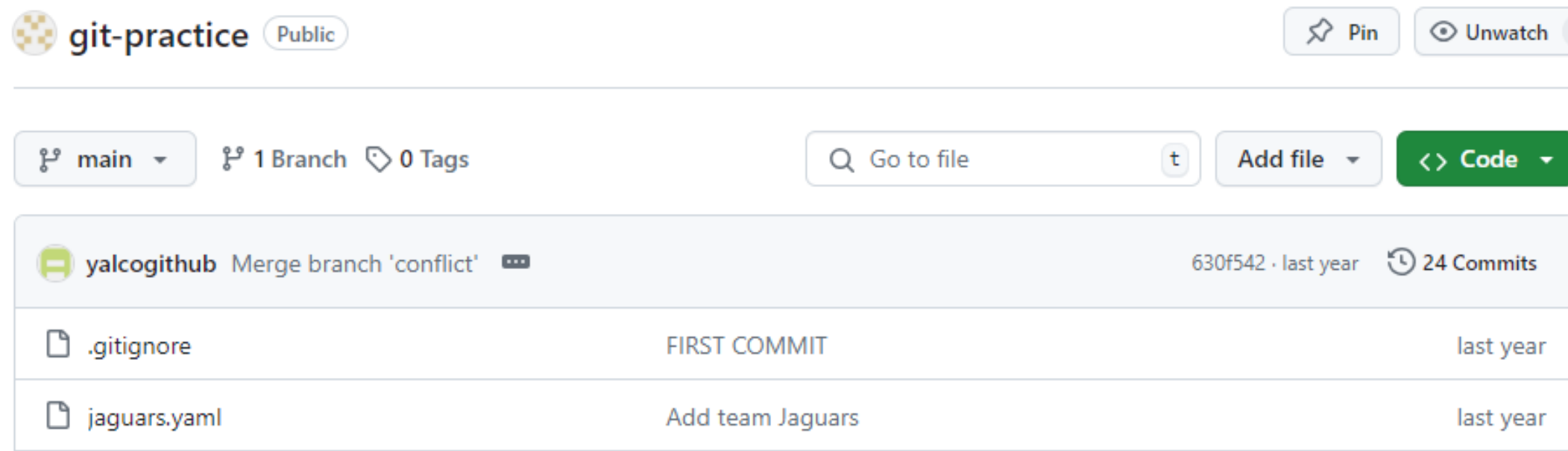
```
stech@LAPTOP-OE1P7VO2 MINGW64 /d/Git_practise (main)
$ git remote add origin https://github.com/Polytech12/git-practice.git
```

```
stech@LAPTOP-OE1P7VO2 MINGW64 /d/Git_practise (main)
$ git branch -M main
```

```
stech@LAPTOP-OE1P7VO2 MINGW64 /d/Git_practise (main)
$ git push -u origin main
Enumerating objects: 77, done.
Counting objects: 100% (77/77), done.
Delta compression using up to 2 threads
Compressing objects: 100% (76/76), done.
Writing objects: 100% (77/77), 7.11 KiB | 207.00 KiB/s, done.
Total 77 (delta 27), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (27/27), done.
To https://github.com/Polytech12/git-practice.git
 * [new branch]    main -> main
branch 'main' set up to track 'origin/main'.
```

로컬에 원격 저장소 추가 후 푸시

- 2) GitHub의 해당 레포지토리 페이지 새로고침하여 살펴보기
- 파일들 내용
 - 커밋 내역들



3) 원격 목록 보기

- git remote
- 자세히 보기: git remote -v

4) 원격 지우기 (로컬 프로젝트와의 연결만 없애는 것. GitHub의 레포지토리는 지워지지 않음)

- git remote remove (origin 등 원격 이름)

GitHub에서 프로젝트 다운받기

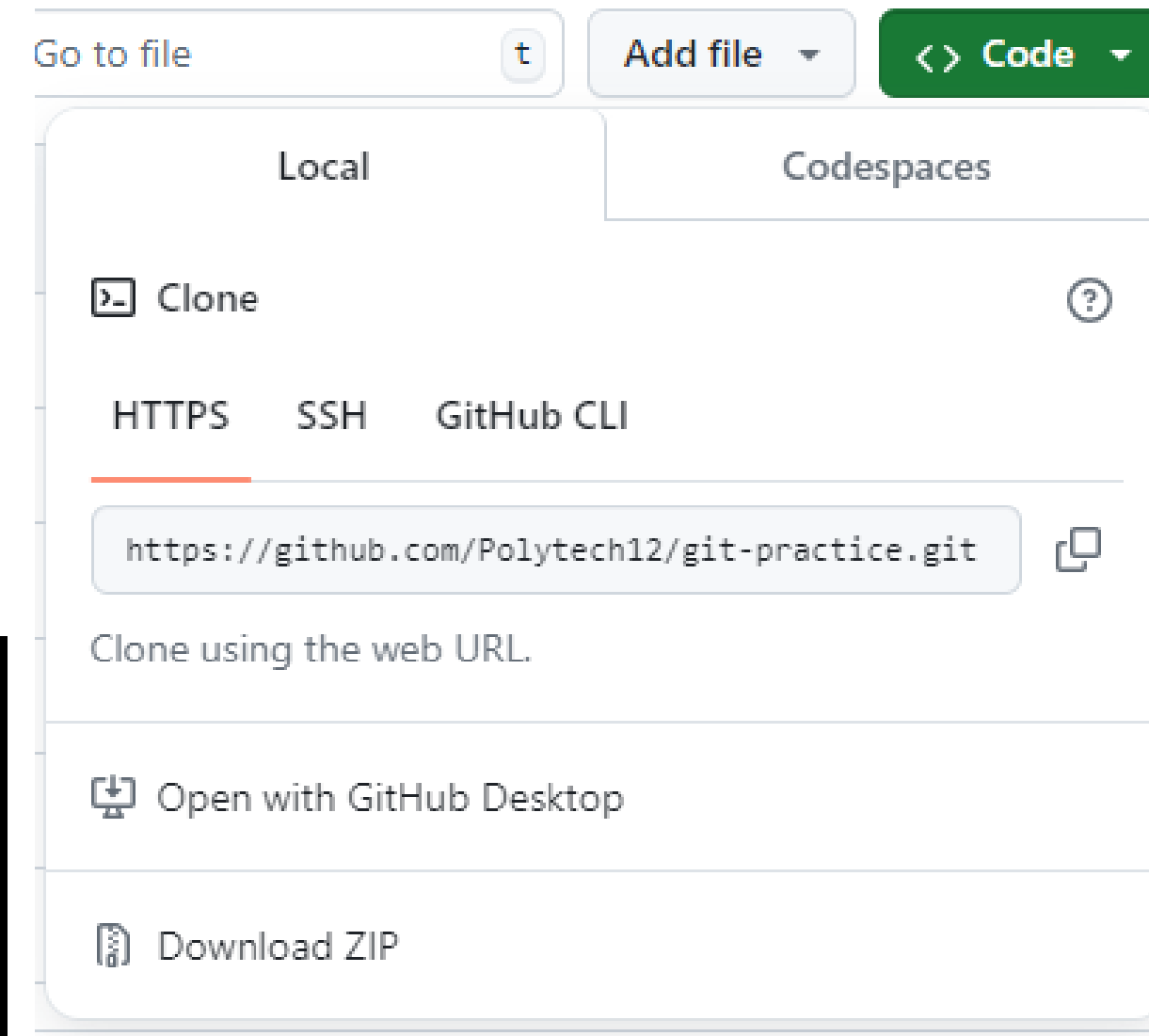
1) GitHub에서 프로젝트 다운받기

- Download ZIP: 파일들만 다운받음, Git 관리내역 제외
- Git clone: Git 관리내역 포함 다운로드

2) 터미널이나 Git Bash에서 대상 폴더 이동 후

- git clone (원격 저장소 주소)

```
stech@LAPTOP-OE1P7V02 MINGW64 /d/clone-practice
$ git clone https://github.com/Polytech12/git-practice.git
Cloning into 'git-practice'...
remote: Enumerating objects: 77, done.
remote: Counting objects: 100% (77/77), done.
remote: Compressing objects: 100% (49/49), done.
remote: Total 77 (delta 27), reused 77 (delta 27), pack-reused 0
Receiving objects: 100% (77/77), 7.11 KiB | 1.42 MiB/s, done.
Resolving deltas: 100% (27/27), done.
```



push

- 1) 원격으로 커밋 밀어올리기(push)
 - Leopards의 members에 Evie 추가
 - 커밋 메시지: Add Evie to Leopards

```
stech@LAPTOP-OE1P7V02 MINGW64 /d/Git_practise (main)
• $ git commit -am 'Add Evie to Leopards'
> '[main e6cf8b8] Add Evie to Leopards
1 file changed, 2 insertions(+), 1 deletion(-)'
```

- 2) 아래 명령어로 push

- git push

. 이미 git push -u origin main으로 대상 원격 브랜치가 지정되었기 때문에 가능

- 3) GitHub 페이지에서 확인

- GitHub의 파일들과 커밋 내역 확인

🔑 main 1 Branch 0 Tags 🔍 Go to file	
urami96 Add Evie to Leopards	
📄 .gitignore	FIRST COMMIT
📄 jaguars.yaml	Add team Jaguars
📄 leopards.yaml	Add Evie to Leopards

```
stech@LAPTOP-OE1P7V02 MINGW64 /d/Git_practise (main)
• $ git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 2 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 312 bytes | 312.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/Polytech12/git-practice.git
630f542..e6cf8b8 main -> main
```

pull

1) 원격의 커밋 당겨오기(pull)

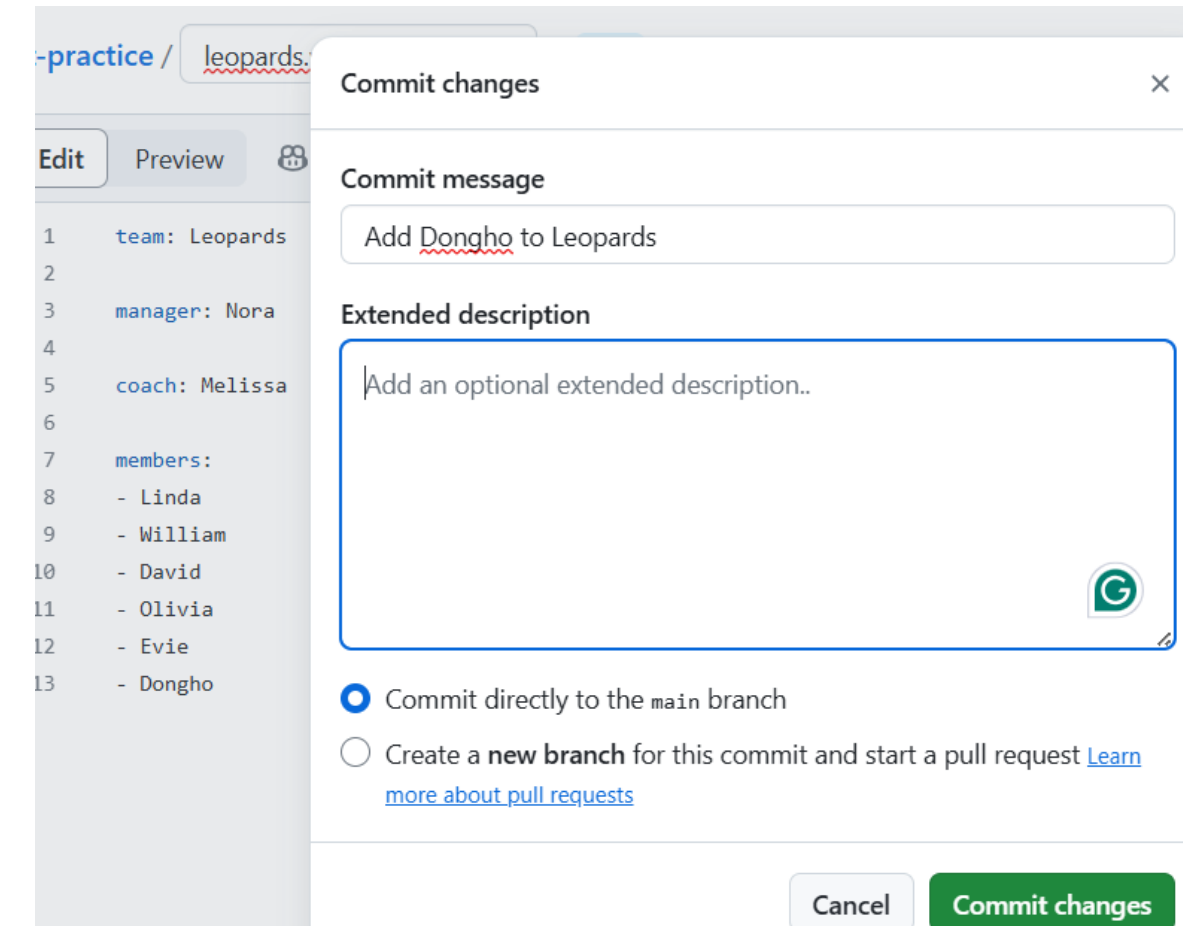
- GitHub에서 Leopards의 members에 Dongho 추가
- 커밋 메시지: Add Dongho to Leopards

2) 아래 명령어로 pull

- git pull

```
stech@LAPTOP-OE1P7V02 MINGW64 /d/Git_practise (main)
● $ git pull
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 2), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 940 bytes | 4.00 KiB/s, done.
From https://github.com/Polytech12/git-practice
   e6cfeb8..0cacaf2  main      -> origin/main
Updating e6cfeb8..0cacaf2
Fast-forward
 leopards.yaml | 3 ++-
 1 file changed, 2 insertions(+), 1 deletion(-)
```

3) 로컬에서 파일과 로그 살펴보기



pull 할 것이 있을 때 push를 하면?

- 1) 로컬에서 Leopards의 manager를 Dooli로 수정
 - GitHub에서 Leopards의 members에 Dongho 추가
 - 커밋 메시지: Edit Leopards manager
- 2) GitHub에서 Leopards의 coach를 Lupi로 수정
 - 커밋 메시지: Edit Leopards coach
- 3) push 해보기
 - 원격에 먼저 적용된 새 버전이 있으므로 적용 불가
 - pull 해서 원격의 버전을 받아온 다음 push 가능

```
stech@LAPTOP-OE1P7V02 MINGW64 /d/Git_practise (main)
$ git commit -am 'Edit Leopards manager'
[main 791ace2] Edit Leopards manager
1 file changed, 1 insertion(+), 1 deletion(-)
```

```
stech@LAPTOP-OE1P7V02 MINGW64 /d/Git_practise (main)
$ git push
To https://github.com/Polytech12/git-practice.git
! [rejected]        main -> main (fetch first)
error: failed to push some refs to 'https://github.com/Polytech12/git-practice.git'
hint: Updates were rejected because the remote contains work that you do not
hint: have locally. This is usually caused by another repository pushing to
hint: the same ref. If you want to integrate the remote changes, use
hint: 'git pull' before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

pull 할 것이 있을 때 push를 하면?

4) Push 할 것이 있을 시 pull 하는 두 가지 방법

- git pull --no-rebase [merge 방식]

- . 소스트리에서 확인해보기

- . reset으로 되돌린 다음 아래 방식도 해보기

- git pull --rebase [rebase 방식]

- . pull 상의 rebase는 다름 (협업시 사용 OK)

5) push하기

협업상 충돌 발생 해결하기

1. 로컬에서 Panthers에 Maruchi 추가
 - 커밋 메시지: Add Maruchi to Panthers
2. 원격에서 Panthers에 Arachi 추가
 - 커밋 메시지: Add Arachi to Panthers
3. pull 하여 충돌상황 마주하기
 - --no-rebase와 --rebase 모두 해 볼 것

로컬의 내역 강제 push해보기

- 1) 로컬의 내역 충돌 전으로 reset
- 2) 아래 명령어로 원격에 강제 적용
 - git push --force

1. 로컬에서 브랜치 만들어 원격에 push 해보기

- 1) from-local 브랜치 만들기
- 2) 아래 명령어로 원격에 push
 - git push
- 3) 아래 명령어로 원격의 브랜치 명시 및 기본설정
 - git push -u origin from-local
- 4) 원격저장소 Jaguars에 cheolsu 추가 [edit
- 4) 브랜치 목록 살펴보기
 - GitHub에서 목록 보기
 - 아래 명령어로 로컬과 원격의 브랜치들 확인
 - . git branch -a

2. 원격의 브랜치 로컬에 받아오기

- 1) GitHub에서 from-remote 브랜치 만들기
 - git branch -a에서 현재는 보이지 않음
- 2) 아래 명령어로 원격의 변경사항 확인
 - git fetch
- 3) 아래 명령어로 로컬에 같은 이름의 브랜치를 생성하여 연결하고 switch
 - git switch -t origin/from-remote

3. 원격의 브랜치 삭제

git push (원격 이름) --delete (원격의 브랜치명)

원격 저장소 이용 관련 명령어들

```
$ git remote -v
```

현재 깃 프로젝트에 등록된 원격저장소 확인하는 깃 명령어

```
$ git remote add 원격저장소 이름 원격저장소주소
```

현재 깃 프로젝트에 원격 저장소를 등록하고, 여기에 이름(별칭)을 붙이는 깃 명령어

```
$ git push
```

로컬 저장소의 내용을 원격 저장소에 공유할 때 사용하는 깃 명령어

```
$ git pull
```

원격 저장소의 내용을 로컬 저장소로 가져와 자동 병합하는 깃 명령어