# Git과 GitHub 고급 (2)

브랜치 분기와 원격 저장소 & 자주 사용되는 Git 명령어

# 목차

1. 이전 내용 리마인드
2. 브랜치 분기
3. 원격저장소
4. 커밋 제어하기
5. 실습

# 주차 별 활동 계획

| 주차 | 날짜 | 주제 |
|---|---|---|
| 1주차 | 9/25 (수) | Git 기초 (1): Git의 설치, Git과 Git의 Concept 소개 |
| 2주차 | 10/2 (수) | Git 기초 (2): 브랜치 분기와 원격 저장소 & 자주 사용되는 Git 명령어 |
| 3주차 | 10/16 (수) | Git 협업 (3): 브랜치 병합과 GitFlow, 그리고 GitHub Issue |
| 4주차 | 10/30 (수) | Git 협업 (4): GitHub Pull Request  & Code Review |
| 5주차 | 11/6 (수) | Git 심화 (5): 병합 전략 및 병합 충돌 해결 전략 (Git Rebase 명령 알아보기) |
| 6주차 | 11/13 (수) | Git 심화 (6): Git의 자료구조와 실수로 삭제한 커밋 복구하기 |
| 7주차 | 11/20 (수) | Git 응용 (7): GitHub Actions를 활용한 자동화 워크플로 생성 |
| 8주차 | 11/27 (수) | Git 응용 (8): Git Submodule & Git Hook (로컬에서의 자동화) |

# 이전 내용 리마인드

# Git이란? (4 Key Words)

- 깃(Git)은 컴퓨터 **파일의 변경사항**을 추적하고 **여러 명의 사용자들** 간에 해당 파일들의 작업을 조율하기 위한 **스냅샷 스트림** 기반의 **분산 버전 관리 시스템.**

**파일의 변경사항** | **여러 명의 사용자들** | **스냅샷 스트림** | **분산 버전 관리 시스템**

# 파일의 Life Cycle

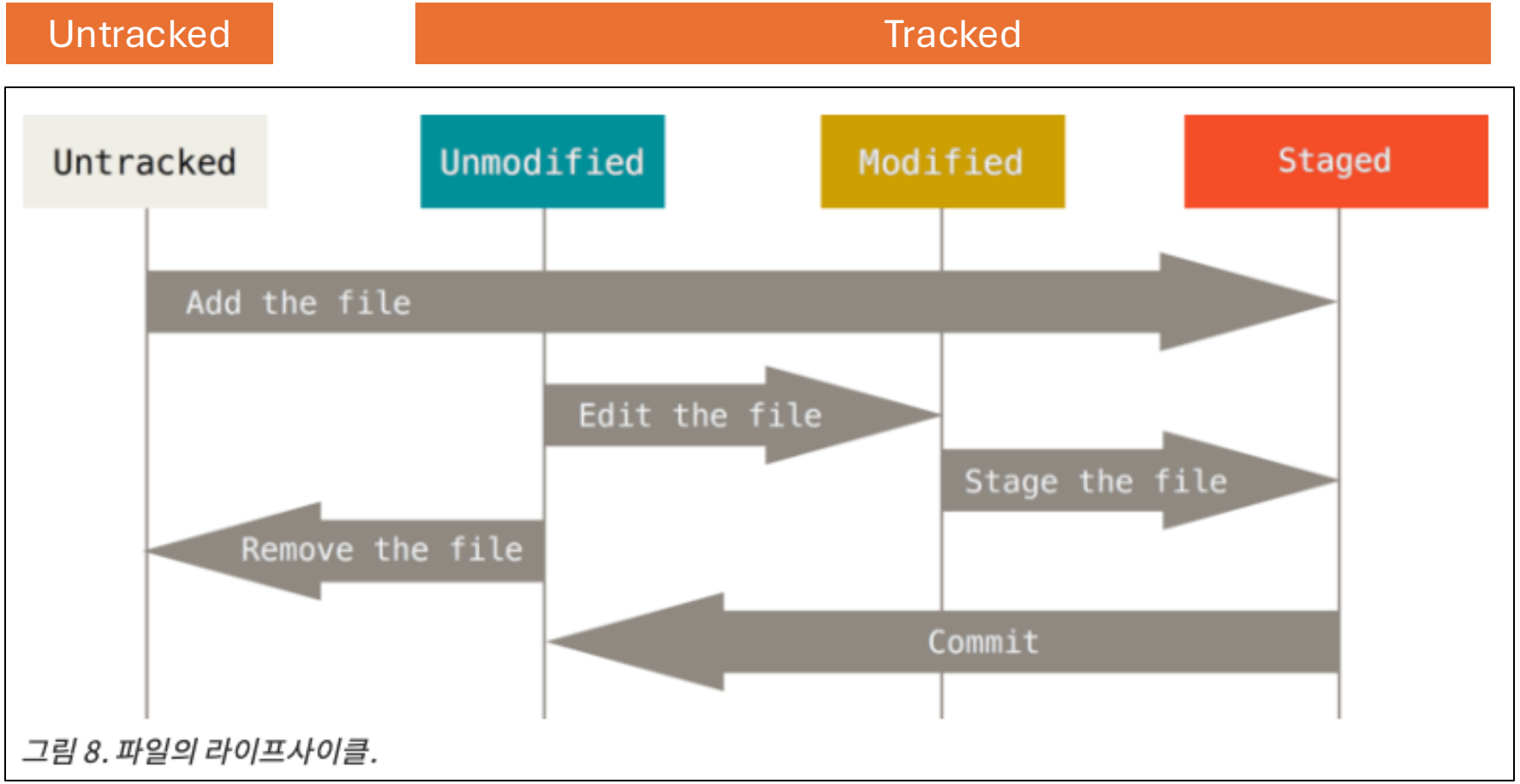



*그림 8. 파일의 라이프사이클.*

# 브랜치 분기

# Commit 을 생성하면

- 아래와 같이 Commit 객체가 생성된다.
  - Commit의 메타 데이터 (작성자, 작성일, 등)
  - 파일 트리
  - 각 파일들에 대한 BLOB

    (Binary Large OBject)

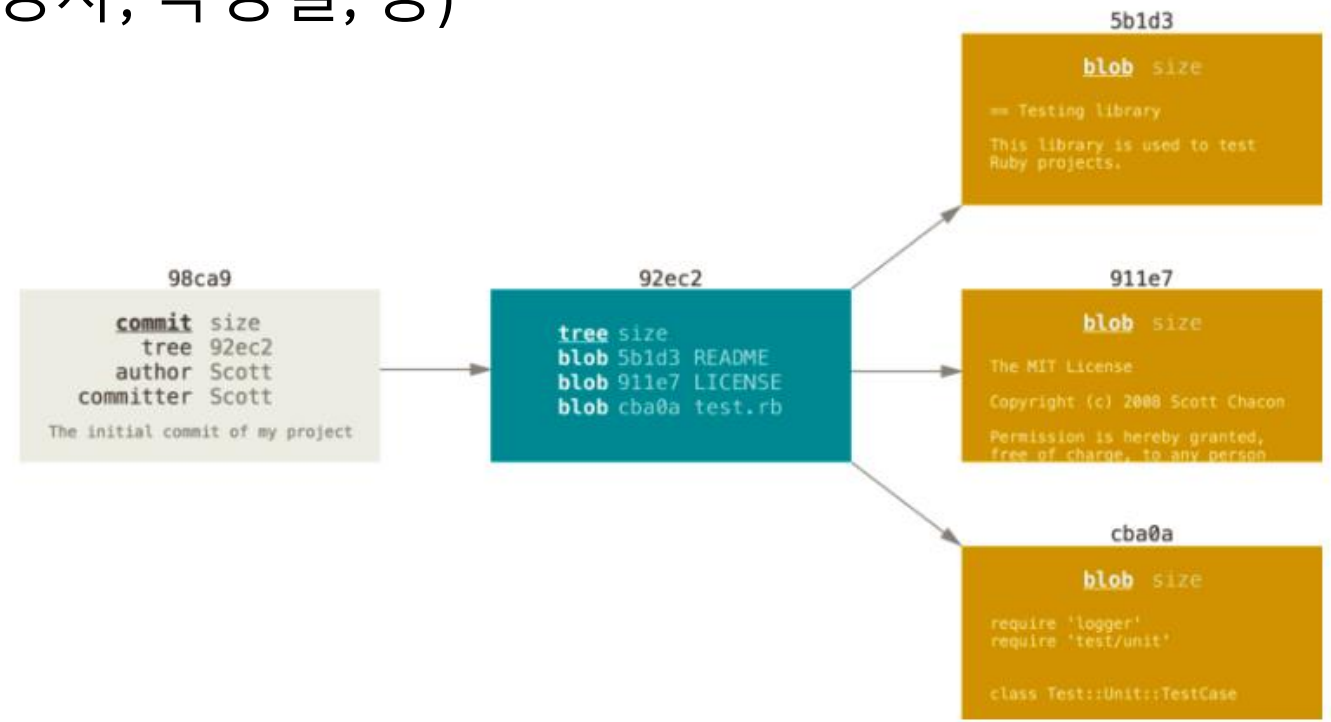



*그림 9. 커밋과 트리 데이터*

# Commit 을 추가하면

- Commit을 추가하면 이전 Commit이 무엇인지도 함께 저장한다.
- (연결리스트-likely, ... 더 나아가면 방향그래프가 됨)

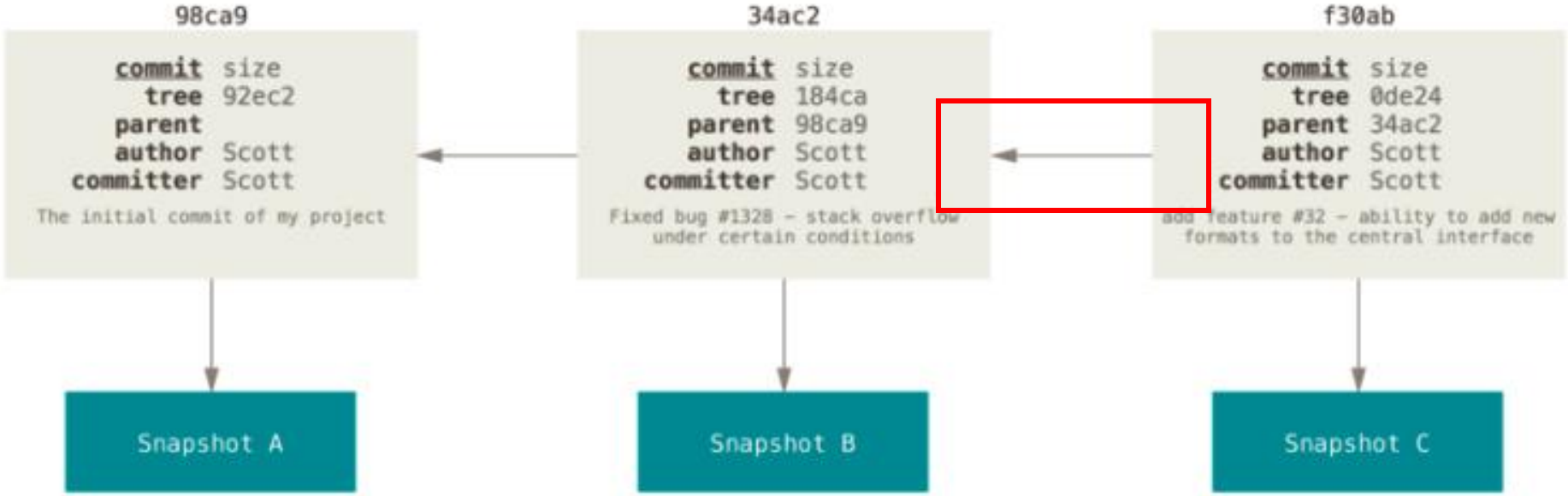



*그림 10. 커밋과 이전 커밋*

# 브랜치란

- Commit 사이를 가볍게 이동할 수 있는 어떤 **포인터** 같은 것.

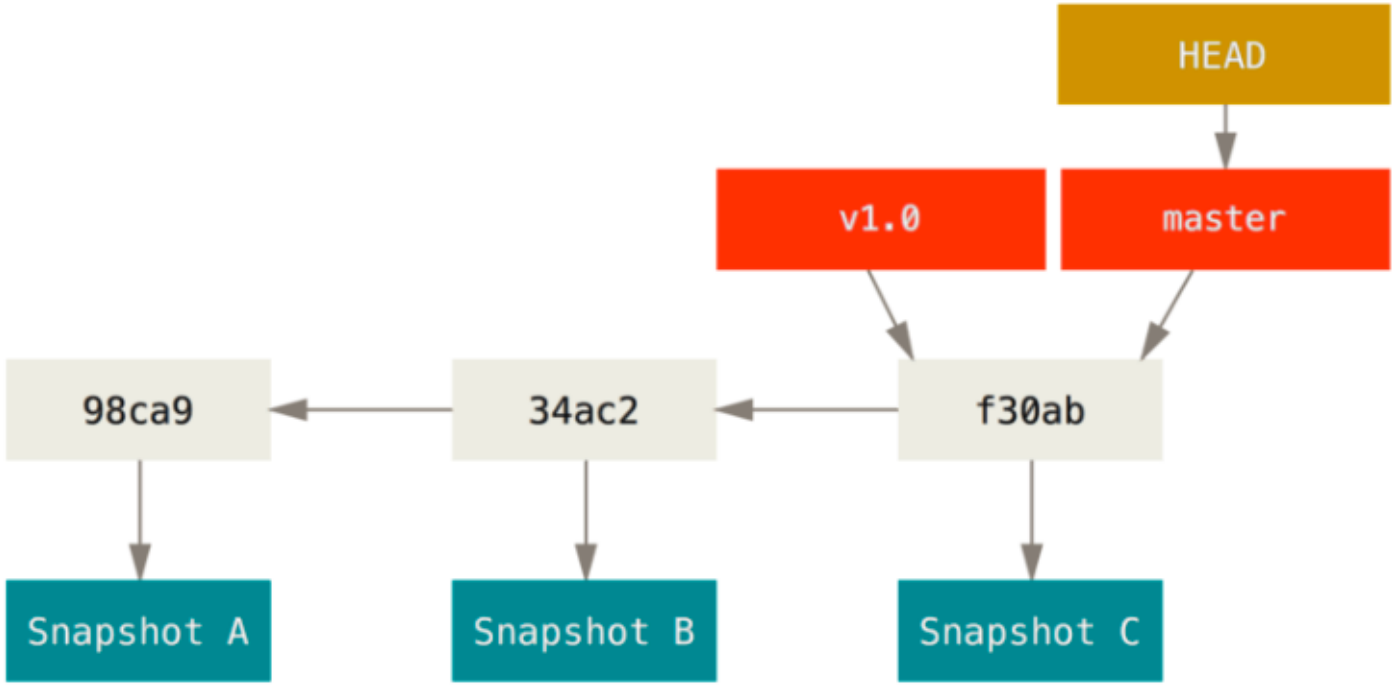



*그림 11. 브랜치와 커밋 히스토리*

# 브랜치란

- 브랜치를 이동하면 Working Directory의 파일이 변경된다.

  (해당 브랜치에서 가장 마지막에 수행항 작업 내용으로 변경)

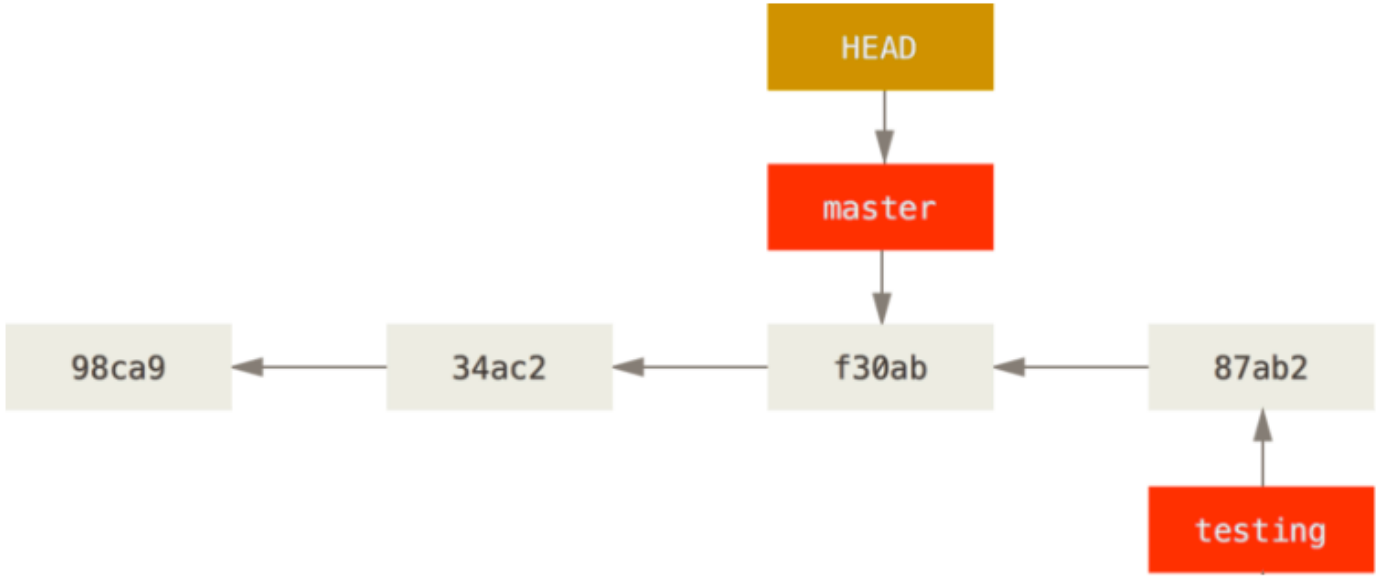
- 파일 변경시 문제가 있는 경우 브랜치 이동 명령은 거부 됨.

  (수정한 내용이 겹치거나 등)

# 브랜치를 여러 개 생성하다 보면...

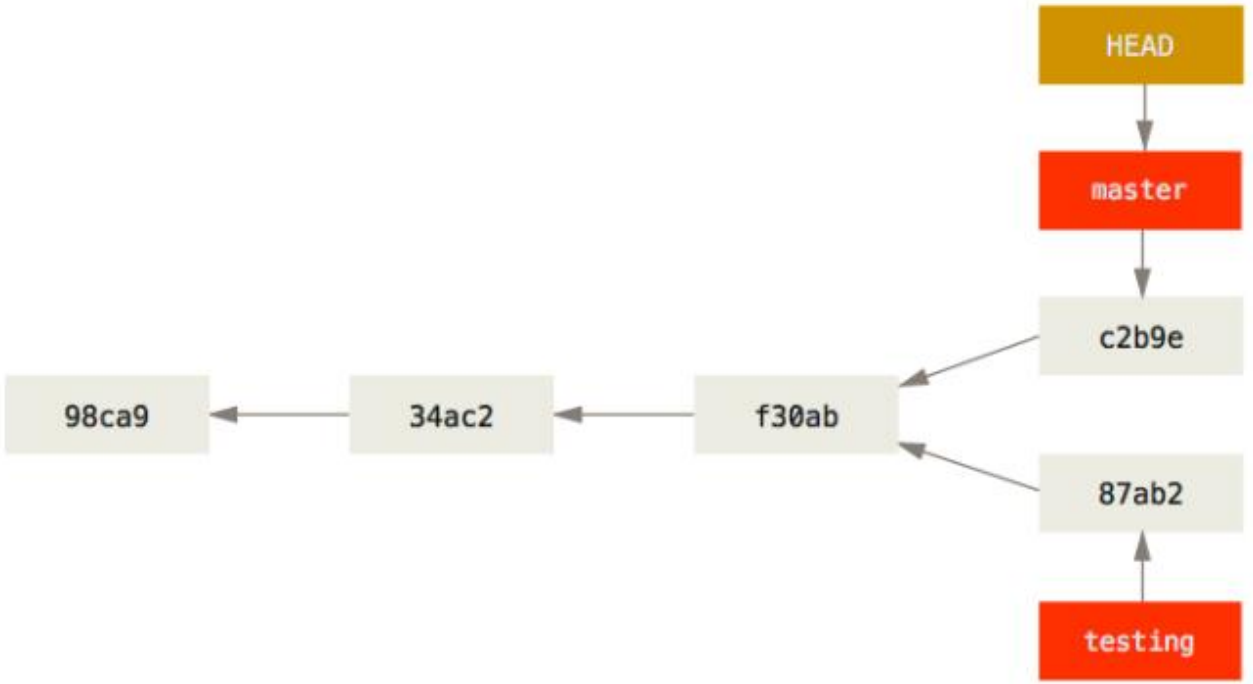
- 브랜치를 이동하면 Working Directory의 파일이 변경된다.

  => Working directory 를 표시하는 HEAD 포인터가 바뀐다



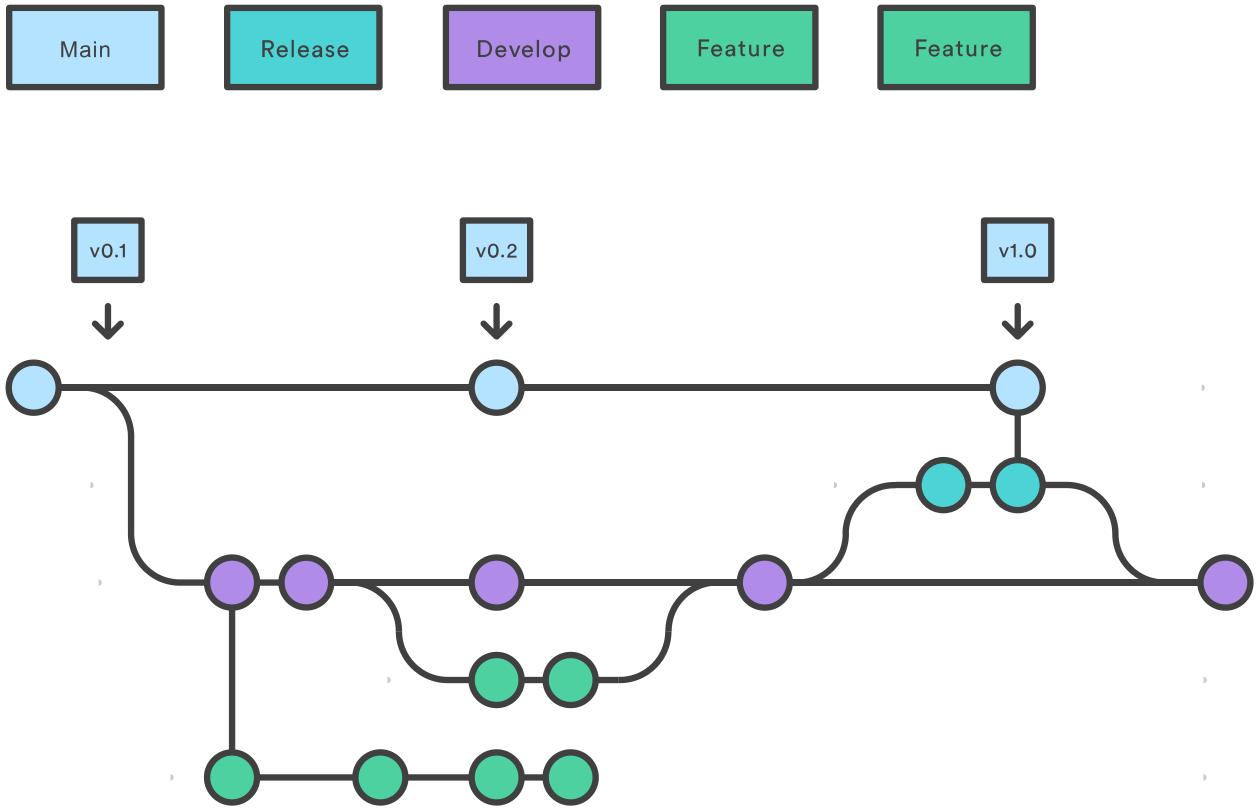
# 브랜치를 여러 개 생성하다 보면...

- HEAD 포인터가 바뀐 사진

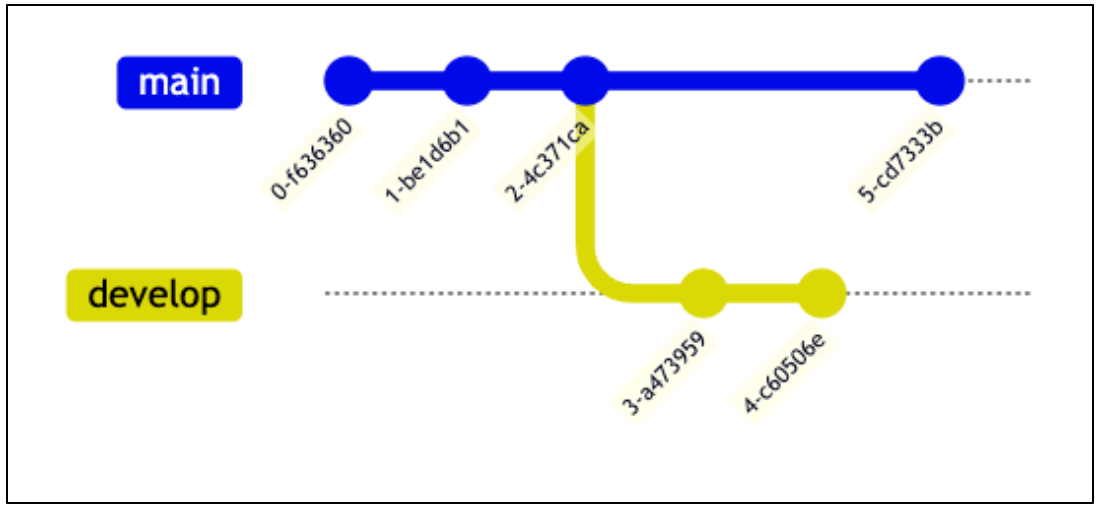


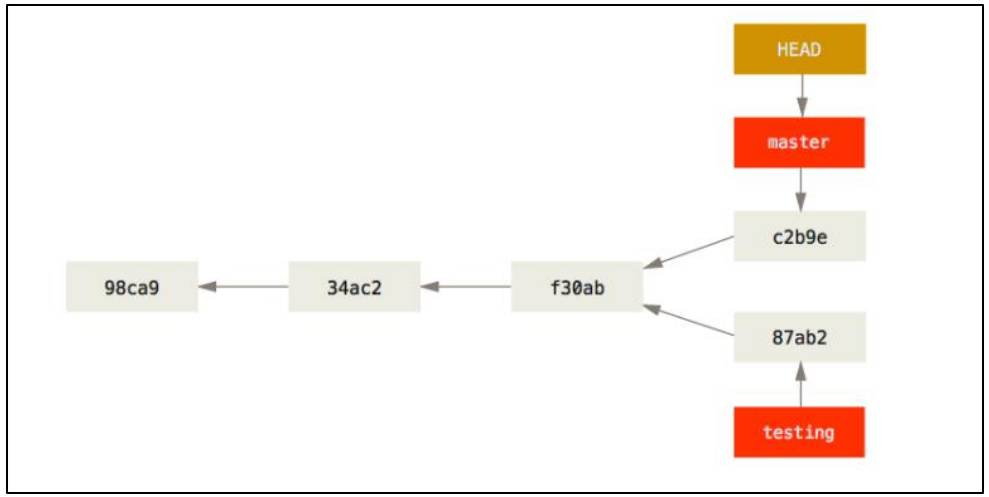
# 브랜치를 여러 개 생성하는 이유

관리를 용이하게 하기 위해서.
목적에 따라 코드를 아래 4가지 형식으로 저장 및 관리하는 경우가 많다.

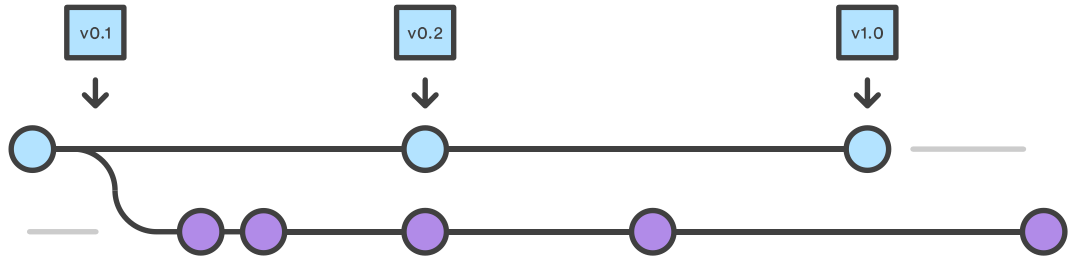


# Git Graph 를 표현하는 다양한 방법







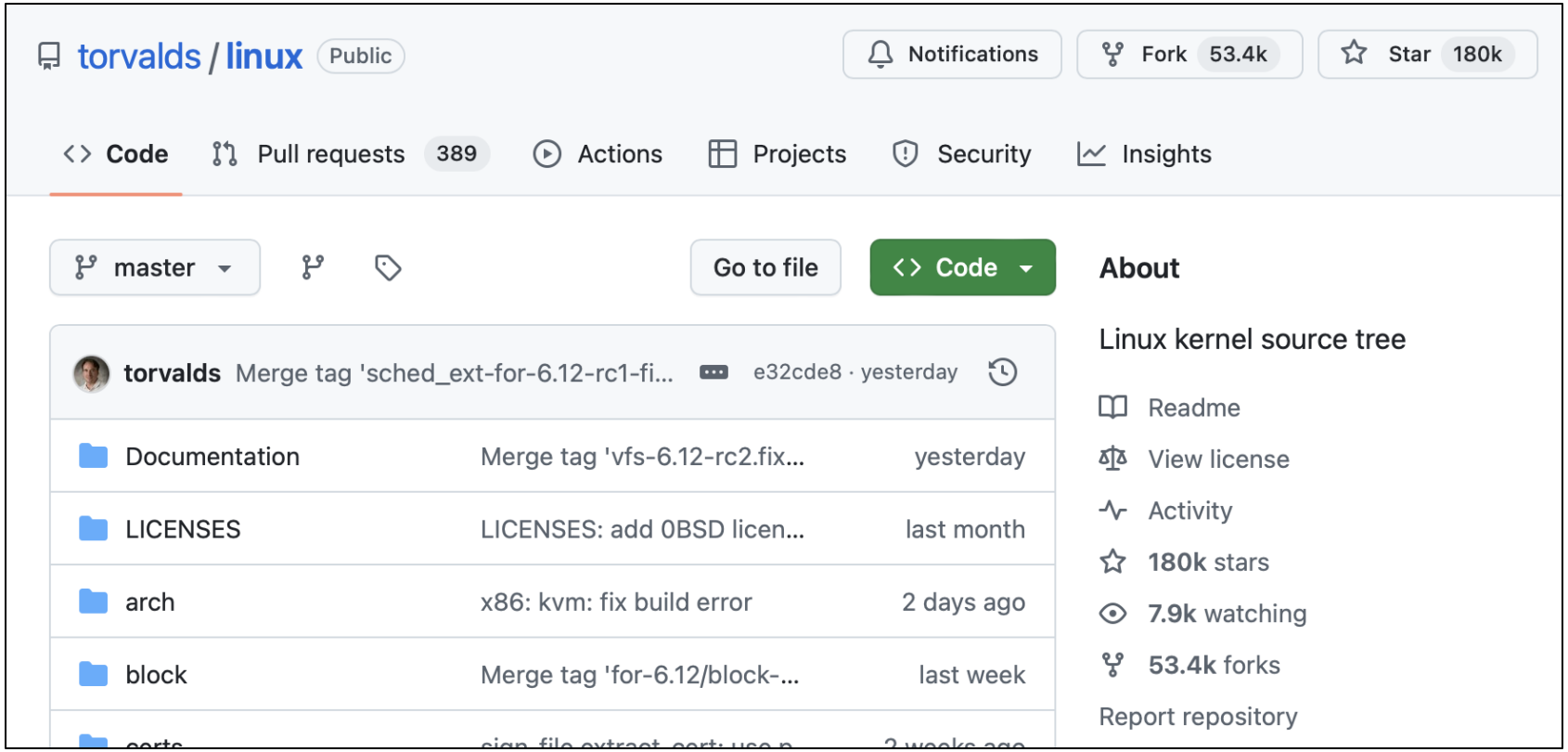
# 특별한 브랜치...? main? master?

- 그런건 없다.

Git 버전 관리 시스템에서 “master” 브랜치는 특별하지 않다. 다른 브랜치와 다른 것이 없다. 다만 모든 저장소에서 “master” 브랜치가 존재하는 이유는 `git init` 명령으로 초기화할 때 자동으로 만들어진 이 브랜치를 애써 다른 이름으로 변경하지 않기 때문이다.

# 특별한 브랜치...? main? master?

- 리눅스의 경우, master 브랜치를 사용하고 있다.



# 브랜치 관련 명령

- 내가 있는 현재 위치에 브랜치 생성하기 (브랜치를 이동하지는 않음)

```
$ git branch <브랜치 명>
```

- 특정 브랜치로 이동하기

```
$ git checkout <브랜치 명>
```

- 특정 브랜치 제거하기

```
$ git branch -d <브랜치 명>
```

# 원격 저장소

# 중앙 집중식

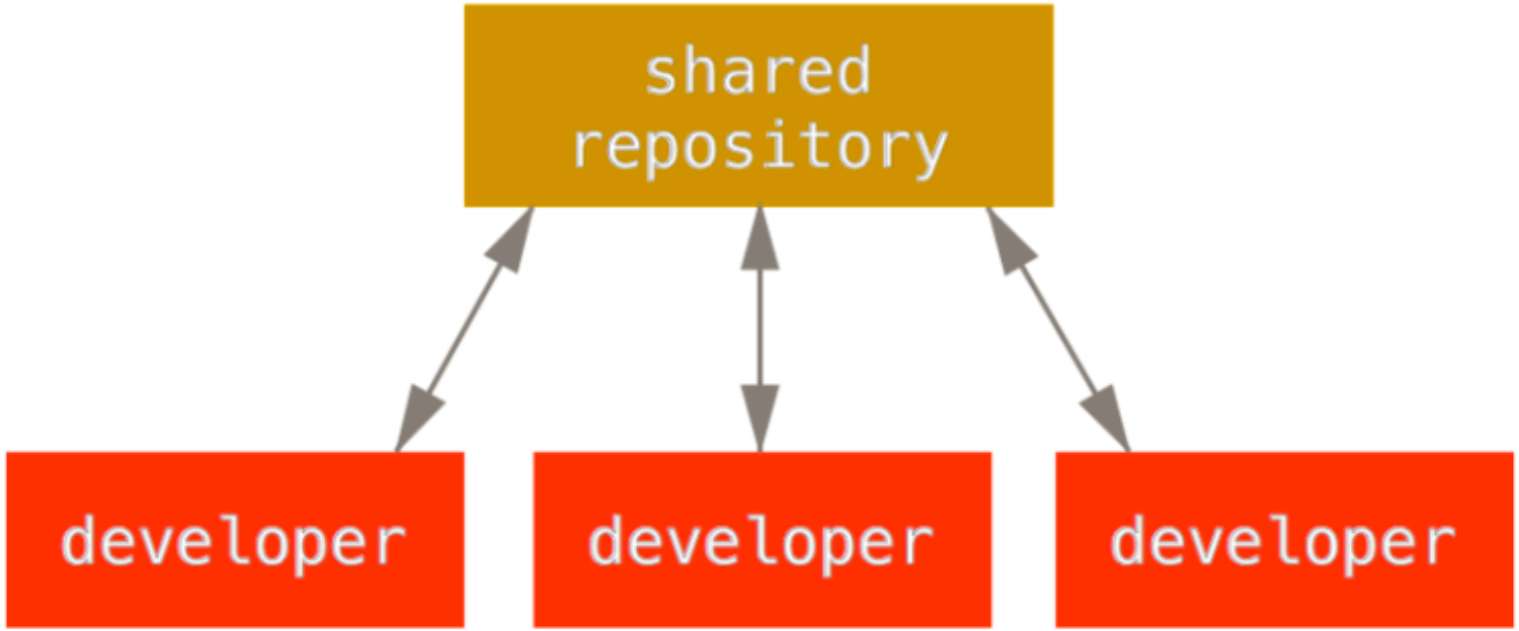



*그림 54. 중앙집중식 워크플로.*

페어 프로그래밍?

# 리모트 저장소

```
$ git remote
```

- 리모트(원격저장소)의 이름을 조회하는 명령어
    - 리모트의 이름은 일반적으로 origin, upstream 의 이름을 많이 쓴다.

# 리모트 저장소의 주소 확인하기

```
$ git remote get-url <리모트 이름>
```

```
$ git remote set-url <리모트 이름> <새로운 주소>
```

```
~/GitHub/smu-202115064/Git-and-GitHub-Advanced (main ✗) git remote
origin
~/GitHub/smu-202115064/Git-and-GitHub-Advanced (main ✗) git remote get-url origin
https://github.com/smu-202115064/Git-and-GitHub-Advanced.git
~/GitHub/smu-202115064/Git-and-GitHub-Advanced (main ✗) git remote set-url origin https://github.com/hepheir/Git-and-GitHub-Advanced.git
~/GitHub/smu-202115064/Git-and-GitHub-Advanced (main ✗) git remote get-url origin
https://github.com/hepheir/Git-and-GitHub-Advanced.git
~/GitHub/smu-202115064/Git-and-GitHub-Advanced (main ✗)
```

# 커밋 제어하기

# 상태 확인하기

```
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
nothing to commit, working directory clean
```

- Untracked = 아직 스냅샷(커밋)에 넣어지지 않은 파일

# 파일을 스테이지 하기

```
$ git add file1 [,file2 [, file3 [, ...]]]
```

- 새로운 파일(Untracked)을 추적하도록 하거나,
- 수정된(Modified) 파일을 커밋할 예정(Staged)으로 변경

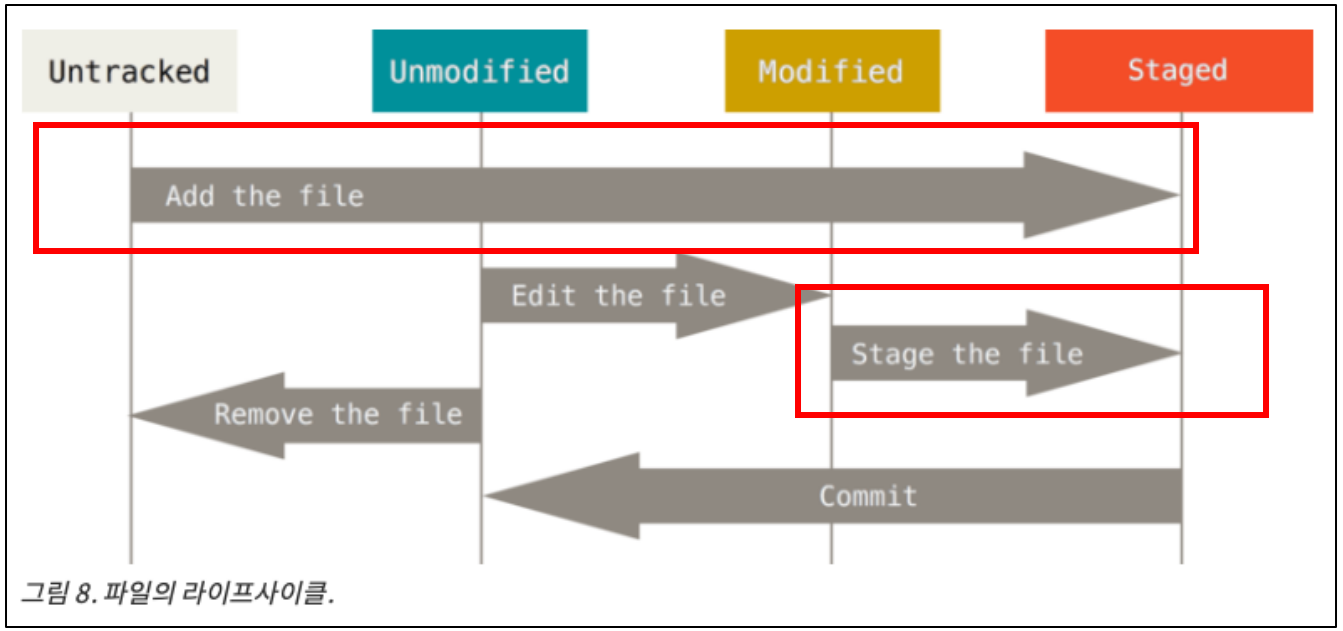
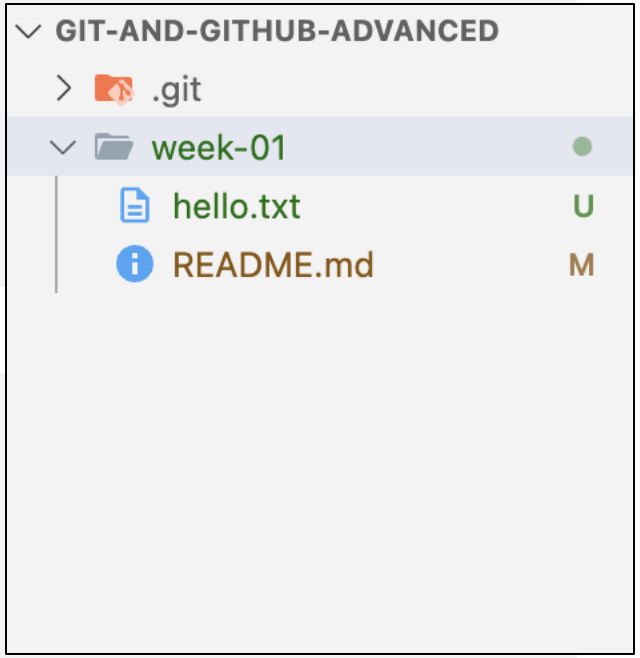



*그림 8. 파일의 라이프사이클.*

# 파일을 스테이지 하기

```
$ git add ./week-01/README.md ./week-01/hello.txt
```




```
$ git add ./week-01/*
```

- 인자로 여러 파일을 입력하거나, Glob 패턴을 사용하여 여러 개의 파일을 지정 가능

# 스테이징을 생략하고 커밋하기

```
$ git commit -a file1 [,file2 [, file3 [, ...]]]
```

- -a 옵션을 추가하면 **Tracked 상태의 파일**을 자동으로 Staging Area에 넣는다.
  `git add` 명령을 입력하는 수고를 덜 수 있다.

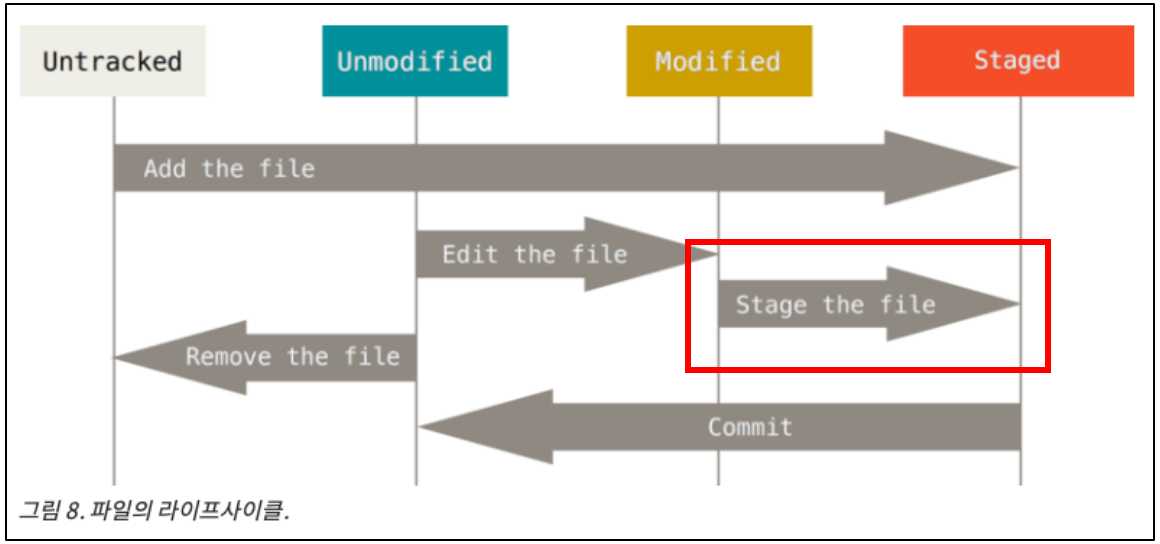


그림 8. 파일의 라이프사이클.

# 파일을 추적하지 않기 (삭제하기)

```
$ git rm file1 [,file2 [, file3 [, ...]]]
```

- -a 옵션을 추가하면 **Tracked 상태의 파일**을 자동으로 Staging Area에 넣는다.
  `git add` 명령을 입력하는 수고를 덜 수 있다.

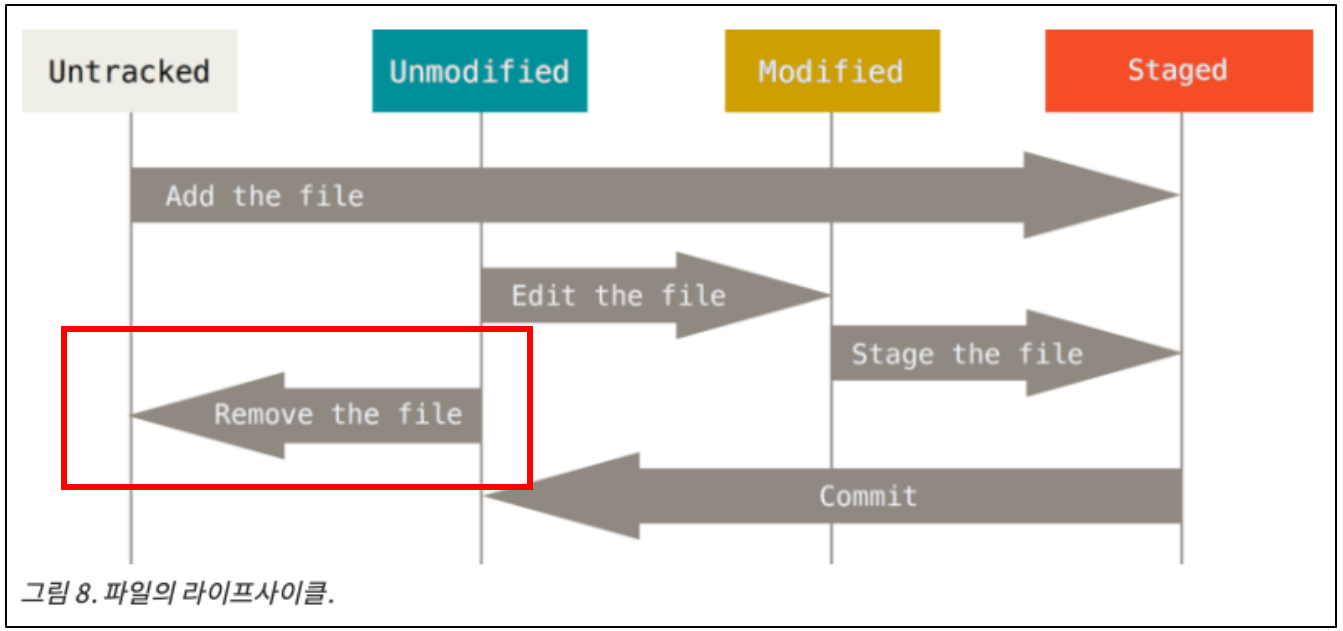



*그림 8. 파일의 라이프사이클.*

# 상태 확인하기

```
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
nothing to commit, working directory clean
```

- Untracked = 아직 스냅샷(커밋)에 넣어지지 않은 파일

# 커밋 수정하기

```
$ git commit --amend
```

- --amend 인자를 붙이면 현재 브랜치의 가장 마지막 commit에 현재 Stage 된 내용을 적용함
  기존 커밋을 수정하는 꼴.

```
$ git commit --amend --no-edit
```

- --no-edit 인자를 붙여 메시지 수정을 생략 할 수 있음.

# 실습

# [지난 과제] 백준 온라인 저지의 문제 1개 풀이.

- 풀이할 문제 선택과 풀이에 사용할 프로그래밍 언어는 자유.
- 제출한 문제 풀이 소스코드를 저장소에 커밋할 것.
  - Commit 시 파일 명은 `/이름/BOJ_문제번호.*` 으로 통일.
    - 예) 김동주, 1000번 문제를 Python 언어로 풀이 시, `/김동주/BOJ_1000.py`
    - 예) 서동혁, 2444번 문제를 C언어로 풀이 시 , `/서동혁/BOJ_2444.c`
  - 문제 풀이 소스코드 1개 외의 파일은 커밋하지 말 것.