GILLILAB · TECH

lazygit 으로 배우는 Git 완전 정복

GitHub 레포 생성부터 FastAPI 프로젝트 운영까지

gillilab

TECH SERIES

Contents

1

들어가며

Git 명령을 못 외워서 막히는 개발자는 생각보다 적습니다. `add`, `commit`, `push` 는 대부분 손에 익어 있습니다. 그런데도 사고가 나는 이유는 다른 데 있습니다. **지금 저장소가 어떤 상태인지가 눈에 보이지 않기 때문**입니다.

인덱스에 무엇이 올라가 있는지, 리베이스 도중 내가 몇 번째 커밋에 서 있는지, 스태시에 무엇을 넣어뒀는지—전부 `git status` 와 `git log` 를 반복해 치면서 머릿속으로 그려야 합니다. 그리고 그 그림이 실제와 어긋나는 순간, 지우면 안 되는 것을 지웁니다.

lazygit 은 그 그림을 화면에 띄워 두는 터미널 UI 입니다. 이 책은 lazygit 사용 설명서가 아니라, **lazygit 화면을 교재 삼아 Git 자체를 익히는 실습서**입니다. 그래서 이 책의 모든 단축키 표에는 대응하는 git 명령이 같은 줄에 실려 있습니다. `<space>` 를 누르는 일이 곧 `git add` 라는 사실을 계속 눈으로 확인하게 만드는 것이 목적입니다.

1.1 이 책이 전제하는 것

전제는 두 가지뿐입니다.

- 터미널에서 명령을 실행하고 파일을 편집할 수 있다
- `git add` · `git commit` · `git push` 를 한 번이라도 써 봤다

그 외에는 전제하지 않습니다. 인덱스와 작업 트리의 차이, 리베이스가 실제로 하는 일, 리플로그가 무엇을 보관하는지는 필요한 자리에서 lazygit 패널과 짝지어 다시 설명합니다. lazygit 을 써 본 적이 없어도 되고, 실습 프로젝트로 쓰는 FastAPI 를 몰라도 됩니다.

반대로 이 책이 맞지 않는 경우도 적어 둡니다. 버전 관리라는 개념 자체가 처음이라면 `git init` 부터 설명하는 입문서를 한 권 먼저 보고 오는 편이 낫습니다. 이 책은"명령은 아는데 상태가 안 그려진다"는 지점을 겨냥합니다.

1.2 다 읽고 나면 할 수 있는 것

- 파일이 아니라 **줄 단위로 골라** 커밋을 쪼갤 수 있습니다.
- `wip` 커밋이 잔뜩 쌓인 브랜치를 인터랙티브 리베이스로 정리한 뒤 PR 을 올릴 수 있습니다.
- 충돌 마커를 직접 만지지 않고 헝크를 골라 충돌을 해결할 수 있습니다.
- `reset --hard` 로 날린 커밋을 리플로그에서 되살릴 수 있습니다.
- 어느 커밋에서 깨졌는지 모를 때 bisect 로 범위를 좁힐 수 있습니다.
- 자주 하는 작업을 `customCommands` 로 단축키에 얹어 자기 도구로 만들 수 있습니다.

그리고 그 하나하나가 어떤 git 명령이었는지 말할 수 있게 됩니다. 이쪽이 사실 더 중요합니다. lazygit 이 깔려 있지 않은 원격 서버에 접속했을 때 손이 멈추면 안 되니까요.

1.3 왜 lazygit 인가

Git 을 다루는 방법은 크게 셋입니다.

CLI 는 정확하지만 상태를 보여주지 않습니다. 무엇을 하든 결과를 다시 물어봐야 알 수 있고, 물어보는 명령 (`status`, `log`, `diff`, `stash list`, `reflog`) 이 또 제각각입니다.

GUI 클라이언트는 상태를 보여주지만 Git 의 어휘를 자기 어휘로 갈아 끼웁니다. "Sync"버튼 하나가 fetch 인지 pull 인지 push 까지인지 화면만 봐서는 알 수 없습니다. 그래서 도구에는 익숙해지지만 Git 에는 익숙해지지 않고, 도구를 바꾸면 처음부터 다시 배웁니다.

lazygit 은 그 사이에 있습니다. 터미널 안에서 돌기 때문에 SSH 로 들어간 서버에서도 그대로 쓸 수 있고, 화면 구성은 Git 개념 (작업 트리·인덱스·브랜치·히스토리·스태시) 그대로이며, `@` 키를 누르면 방금 실행된 **진짜 git 명령**이 로그로 나옵니다. 배우는 내내 정답지를 옆에 두고 있는 셈입니다.

이 책이 lazygit 을 고른 이유가 여기 있습니다. 편해서가 아니라, 배우는 과정이 그대로 Git 학습이 되기 때문입니다.

1.4 무엇을 만들며 배우는가

FastAPI 로 작은 Task API 를 하나 만듭니다. 할 일을 만들고 조회하고 수정하고 지우는, 엔드포인트 다섯 개짜리 서비스입니다.

굳이 실제 프로젝트를 쓰는 이유가 있습니다. 파일 하나에 커밋 두 개짜리 연습용 저장소에서는 배울 것이 생기지 않습니다. 한 파일에 두 가지 변경이 섞이지 않으면 라인 단위 스테이징을 할 이유가 없고, 브랜치가 하나면 충돌이 나지 않으며, 커밋이 셋뿐이면 리베이스로 정리할 것도 없습니다. Git 의 어려운 기능들은 전부 **코드가 어느 정도 쌓인 뒤에야 필요해지는** 기능입니다.

그래서 이 책은 `gh repo create` 로 저장소를 만드는 것부터 시작해 라우터를 붙이고, 브랜치를 따고, PR 을 올리고, 일부러 충돌을 내고, 히스토리를 정리하고, 태그를 붙이는 데까지 저장소 하나를 끝까지 끌고 갑니다. 대부분의 장이 앞 장의 결

과 위에서 시작합니다.

FastAPI 자체는 가르치지 않습니다. 필요한 코드는 본문에 전부 실려 있으니 그대로 옮겨 적으면 되고, 코드가 무슨 뜻인지 몰라도 Git 실습에는 지장이 없습니다. 이 책에서 코드는"커밋할 거리"입니다.

1.5 이 책을 읽는 법

- **순서대로 읽으세요.** 3 장에서 만든 환경, 4 장에서 만든 저장소, 5 장에서 쓴 코드가 뒤 장의 재료가 됩니다. 건너뛰면 실습할 대상이 없습니다.
- **표는 두 열을 같이 보세요.** 왼쪽 단축키만 외우면 lazygit 사용자가 되고, 오른쪽의 git 명령을 함께 보면 Git 을 배우게 됩니다. 이 책은 후자를 노립니다.
- **손으로 따라 하세요.** 읽기만 해도 이해는 됩니다. 하지만 충돌 해결과 리플로그 복구는 한 번 해 보지 않으면 정작 필요한 순간에 손이 나가지 않습니다.
- **단축키 표기는 lazygit 문서와 같습니다.** `<space>`·`<enter>`·`<ctrl+f>` 는 그 키 자체를 뜻하고, 대문자는 shift 조합입니다 (`A` 는 shift+a).
- **뒤의 세 장은 참조용입니다.** 트러블슈팅 (12 장)·핵심 단축키 빠른 참조 (13 장)·마치며 (14 장) 는 순서대로 읽는 장이라기보다 막혔을 때 펼치는 장입니다.

1.6 이 책이 검증된 환경

> **기준 버전**: lazygit 0.63.1 (2026-07-15) / gh 2.96.0 / git 2.50.1 / uv 0.11.28 / FastAPI 0.140.0 — 2026-07 확인

본문의 단축키·명령·출력은 모두 이 조합에서 확인한 것입니다. 버전이 더 높아도 대부분 그대로 동작합니다. 만약 단축키가 달라졌더라도 `?`(현재 패널의 단축키 메뉴) 와 `@`(실행된 git 명령 로그) 두 개면 직접 확인할 수 있습니다. 그래서 이 두 키를 3 장에서 가장 먼저 익힙니다.

2

시작하기 전에

이 장은 두 가지를 합니다. 실습을 시작하기 전에 갖춰야 할 것을 확인하고, 앞으로 책이 어떤 경로로 움직이는지를 미리 보여줍니다. 지금 전부 이해할 필요는 없고, 나중에"이게 왜 필요했더라"싶을 때 돌아와 보는 지도 정도로 생각하면 됩니다.

2.1 갖춰야 할 것

- ☐ **터미널과 에디터** —명령을 실행하고 파일을 편집할 수 있으면 무엇이든 됩니다. 본문의 셸 예시는 macOS 기준으로 적혀 있지만, 단축키와 화면 구성은 OS 와 무관하게 같습니다.
- ☐ **git** —이미 깔려 있을 확률이 높습니다. `git --version` 으로 확인합니다.
- ☐ **lazygit 과 GitHub CLI** —`brew install lazygit gh`. macOS 가 아니면 3 장에 apt 와 Go 툴체인 설치 방법도 함께 실었습니다.
- ☐ **GitHub 계정과 인증** —`gh auth login --web --git-protocol ssh`. 4 장에서 실습 저장소를 GitHub 에 직접 만들기 때문에 계정이 필요합니다.
- ☐ **uv** —실습 프로젝트를 만들 Python 패키지 매니저입니다. 설치 방법은 14 장 참고 자료의 uv 공식 문서에 있습니다.

앞의 셋만 있으면 3 장까지는 그대로 진행할 수 있습니다. GitHub 계정과 uv 는 4 장·5 장에 들어가기 전까지 준비하면 됩니다.

주의

실습 저장소는 `--public` 으로 만듭니다. 회사 계정이나 조직 계정으로 로그인한 상태라면 개인 계정인지 먼저 확인하세요.

2.2 이 책이 지나갈 길

장	하는 일
3	lazygit 을 설치하고 5 개 패널이 각각 어떤 Git 개념인지 익힙니다
4	`gh repo create` 로 실습 저장소를 만들고 원격을 연결합니다
5	FastAPI 코드를 넣고 **라인 단위로 골라** 첫 커밋을 만듭니다
6	브랜치를 따서 엔드포인트를 추가하고 PR 까지 올립니다
7	쌓인 `wip` 커밋을 인터랙티브 리베이스와 fixup 으로 정리합니다
8	커밋을 옮기고 (cherry-pick) 되돌리고 (revert) 일부만 떼어냅니다
9	일부러 충돌을 내고 화면에서 헝크를 골라 해결합니다
10	스태시·reset·reflog·undo·bisect 로 사고 난 저장소를 되살립니다
11	worktree·submodule·tag 를 다루고 `config.yml` 로 도구를 자기 손에 맞춥니다

3 장부터 11 장까지는 저장소 하나를 계속 이어서 씁니다. 중간에 며칠 쉬었다가 돌아와도 저장소만 그대로 두면 이어집니다.

2.3 막혔을 때

- 화면이 깨지거나 명령이 실패하면 **12 장 트러블슈팅**의 원인·해결 표를 먼저 봅니다. 실제로 자주 걸리는 항목만 모았습니다.
- 단축키가 기억나지 않으면 **13 장 핵심 단축키 빠른 참조**를 펼치거나, lazygit 안에서 `?` 를 누릅니다. `?` 는 지금 포커스된 패널 기준으로 목록을 보여줍니다.
- 방금 누른 키가 무슨 일을 했는지 모르겠으면 `@` 를 눌러 실행된 git 명령을 확인합니다. 이 습관 하나가 이 책 전체에서 가장 오래 남습니다.

3

설치·CLI 플래그·화면 구조

도구를 깔고 화면과 친해지는 장입니다. 앞으로 나올 모든 실습이 여기서 익힌 다섯 개 패널 위에서 벌어지므로, 이 장만큼은 단축키를 외우기보다 **어느 패널이 Git 의 무엇에 해당하는지**를 잡는 데 집중하세요. 그 대응 하나가 이 책 전체의 뼈대입니다.

설치 방법은 셋입니다. 하나만 골라 실행하면 됩니다. macOS 라면 Homebrew, Ubuntu/Debian 이라면 apt, 그 외 환경이거나 최신 버전을 직접 받고 싶다면 Go 툴체인을 씁니다.

```
# macOS (Homebrew)
brew install lazygit gh

# Ubuntu / Debian
sudo apt install lazygit gh

# Go 툴체인 (모든 플랫폼)
go install github.com/jesseduffield/lazygit@latest

# 설치 확인
lazygit --version
gh --version
git --version
```

예상 출력:

```
commit=, build date=, build source=Homebrew, version=0.63.1, os=darwin, arch=arm64,
  ↪ git version=2.50.1 (Apple Git-155)
```

```
gh version 2.96.0 (2026-07-02)
git version 2.50.1 (Apple Git-155)
```

세 줄이 다 나오면 준비가 끝난 것입니다. 버전 숫자가 이보다 높아도 상관없습니다. 낮다면—특히 lazygit 이 0.5x 대라면—뒤에 나오는 단축키 몇 개가 다를 수 있으니 업데이트를 권합니다.

3.1 CLI 플래그 전체

lazygit 은 저장소 안에서 인자 없이 `lazygit` 만 쳐도 됩니다. 아래 플래그들은 알아두면 편한 것들인데, 실제로 매일 쓰게 되는 건 `-p`(다른 저장소 열기), `--filter`(파일 하나의 히스토리만 보기), `--print-config-dir`(설정 파일 위치 찾기) 정도입니다. 나머지는 필요해질 때 이 표로 돌아오면 됩니다.

```
lazygit --help                    # 전체 도움말
lazygit log                       # 포커스 패널 지정 (status|branch|log|stash)
lazygit -p ~/work/other-repo      # 다른 저장소 열기
lazygit --filter app/main.py      # 특정 파일 히스토리만 필터링
lazygit --screen-mode half        # 시작 화면 모드 (normal|half|full)
lazygit --print-config-dir        # 설정 디렉토리 출력
lazygit --config                  # 기본 설정 전체 출력
lazygit -ucf ~/dotfiles/work.yml  # 커스텀 설정 파일 (쉼표로 복수)
lazygit -w ~/proj -g ~/proj/.git  # work-tree / git-dir 명시
lazygit completion zsh            # 셸 자동완성 생성 (bash|zsh)
lazygit --debug                   # 디버그 모드 (별도 탭에서 --logs)
```

3.2 5 개 패널과 Git 개념 대응

저장소에서 `lazygit` 을 실행하면 왼쪽에 세로로 다섯 개의 패널이, 오른쪽에 그 선택 항목의 상세 (주로 diff) 가 나옵니다. 이 다섯 개는 임의로 나눈 UI 가 아니라 **Git 이 관리하는 다섯 가지 상태를 그대로 하나씩 맡고 있습니다.**

번호	패널	Git 대응 개념	대응 명령어
1	Status	저장소·브랜치 요약	`git status -sb`
2	Files	작업 트리 + 인덱스	`git status`
3	Branches	로컬/원격 브랜치, 태그	`git branch -a`, `git tag`
4	Commits	커밋 히스토리, reflog	`git log`, `git reflog`
5	Stash	스태시 목록	`git stash list`

오른쪽 열의 명령들을 평소에 따로따로 쳐 왔다면, lazygit 은 그 다섯 개를 항상 동시에 띄워 두는 도구라고 이해하면 정확합

니다. 그래서"지금 상태가 어떻더라"를 확인하려고 명령을 칠 일이 사라집니다.

3.3 내비게이션 단축키

여기 있는 표를 다 외울 필요는 없습니다. 당장 필요한 건 `1~5`(패널 점프), `j/k`(항목 이동), `<enter>/<esc>`(진입·복귀), 그리고 `?`(지금 패널의 단축키 전부) 다섯 개뿐입니다. 나머지는 화면을 돌아다니다 보면 자연스럽게 손에 붙습니다.

단축키	기능
`1 ~ 5`	해당 번호 패널로 즉시 점프
`<tab>` / `<backtab>`	다음 / 이전 패널
`h l` 또는 ← →	이전 / 다음 패널
`j k` 또는 ↓ ↑	항목 아래 / 위 이동
`]` / `[`	패널 내 다음 / 이전 탭
`0`	메인 뷰 포커스
`<enter>` / `<esc>`	진입 / 뒤로
`v`, `<shift+↓>`, `<shift+↑>`	범위 선택
`/`, `n`, `N`	검색, 다음/이전 결과
`,` / `.`	이전 / 다음 페이지
`<` / `>`	맨 위 / 맨 아래
`H` / `L`	좌 / 우 스크롤
`+` / `_`	다음 / 이전 화면 모드
`?`	현재 패널 전체 단축키 메뉴
`q` / `Q`	종료 / 디렉토리 변경 없이 종료

주의

`?` 는 **현재 포커스된 패널 기준**으로 단축키를 보여줍니다. 커밋 패널의 `?` 와 파일 패널의 `?` 는 다른 목록입니다.

3.4 지금부터 들일 습관 하나

`@` 를 누르면 lazygit 이 방금 실행한 **실제 git 명령**이 로그로 나옵니다. 이 책의 표들이 단축키 옆에 git 명령을 나란히 실어 둔 이유가 여기 있습니다. 표를 외우는 대신, 단축키를 누른 뒤 `@` 로 무슨 명령이 나갔는지 확인하는 편이 훨씬 빨리 붙습니다.

앞으로 처음 보는 단축키를 누를 때마다 `@` 를 한 번씩 눌러 보세요. 이 습관 하나가 이 책에서 얻어 가는 것 중 가장 오래 남습니다.
