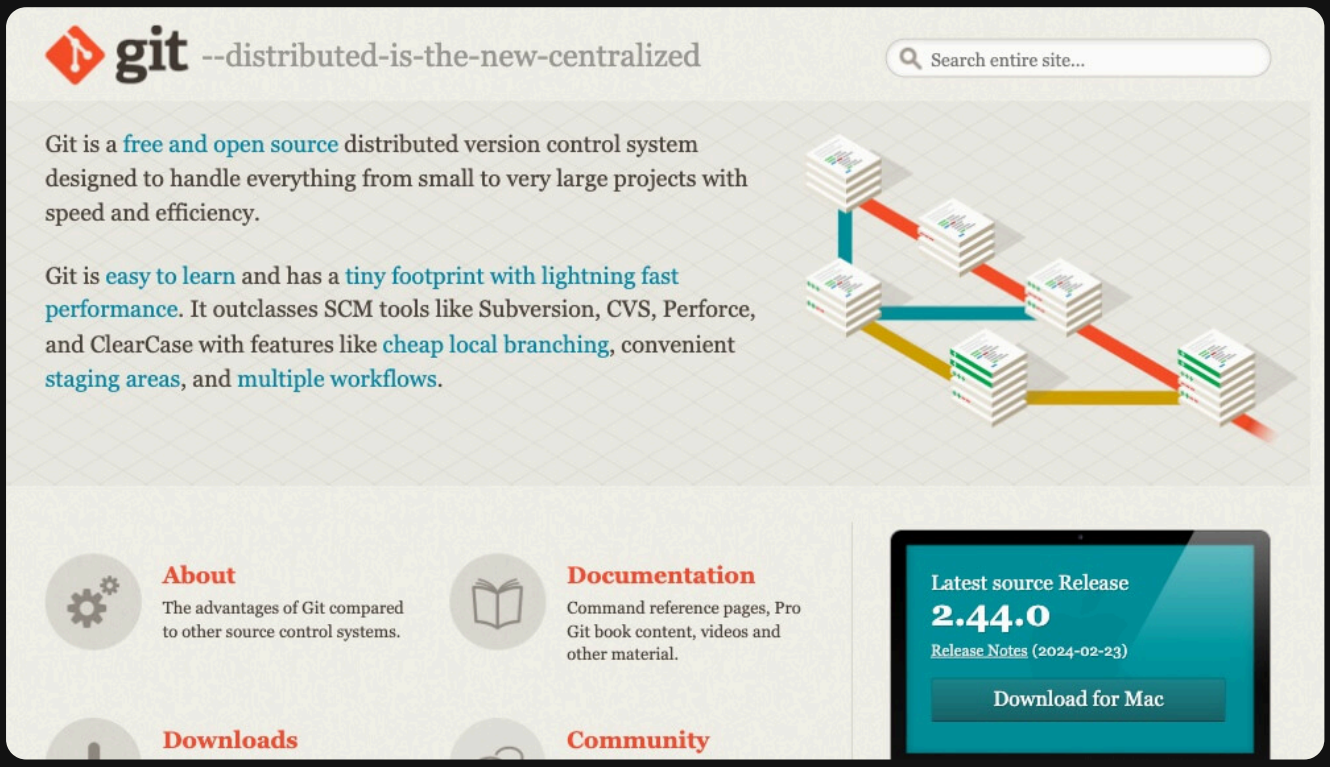
Git 핵심 명령어 모음



버전 관리 시스템(VCS) Git에서 주로 사용하는 명령을 빠르게 정리합니다.

구성 (Config)

구성(Config)은 운영체제 단위의 Git 환경 설정입니다.
개행 문자(Newline) 구성은 Windows와 Unix 계열 운영체제(macOS) 간의 줄바굼 호환성 문제를 방지하기 위한 설정입니다.
이름과 이메일은 버전 생성 시 작성자 정보를 표시하기 위함으로, GitHub 등의 서비스 사용자 정보와 달라도 무방하나 되도록 같게 작성하는 것이 좋습니다.

명령	설명
`git -v`	Git 버전 확인

`git config --global core.autocrlf true`	개행 문자 설정 (Windows)
`git config --global user.name '<이름>'`	사용자 이름 설정
`git config --global user.email '<이메일>'`	사용자 이메일 설정
`git config --global init.defaultBranch main`	Git v2.28 미만인 경우, 메인 브랜치 이름을 `main` 으
`git config --global pull.rebase true`	`pull` 명령어 실행 시 리베이스를 기본 동작으로 설정
`git config --global --list`	구성 목록 확인
`git config --global --unset <항목이름>`	구성 항목 삭제

초기화 (Init)

초기화(Init)는 프로젝트 단위로 Git 버전 관리를 시작하는 기능입니다.
원격 별칭(Remote Alias)은 원격 저장소를 지칭하는 이름으로, 단일 원격인 경우 `origin` 을 사용하는 것을 추천합니다.

명령	설명	
`git init`	프로젝트 버전 관리 시작	
`git remote`	원격 저장소 목록 확인	
`git remote -v`	원격 저장소 URL 확인	
`git remote add <원격별칭> <URL>`	원격 저장소 추가	`git remote add origin https`
`git clone <URL>`	원격 저장소 복제	`git clone https://github.c`
`rm -rf .git`	버전 관리 초기화 (macOS)	
`rmdir /s .git`	버전 관리 초기화 (Windows)	

추적 (Track)

명령	설명	예시
`git status`	현재 브랜치의 변경사항 확인	
`git add <파일>`	특정 파일 추적 및 스테이징	`git add ./src/main.js`
`git add .`	모든 파일 추적 및 스테이징	
`git mv <파일A> <파일B>`	스테이징된 파일 이름 변경	`git mv ./mnin.js ./ma`
`git rm -r --cached <파일>`	추적 목록에서 제거 (`.gitignore` 갱신)	`git rm -r --cached ./`
`git clean -fdn`	삭제 가능한 추적되지 않은 파일 목록 확인	
`git clean -fd`	추적되지 않은 파일 삭제	
`git restore --staged <파일>`	특정 파일 언스테이징 (v2.23)	`git restore --staged`
`git restore --staged .`	모든 파일 언스테이징 (v2.23)	

버전 생성 (Commit)

버전 생성(Commit)은 현재 작업 내용을 하나의 버전으로 기록(생성)하는 것을 말합니다.

명령	설명
`git commit -m '<메시지>'`	버전 생성 (따옴표 닫기 전에는 메시지 줄바꿈 가능)
`git commit -am '<메시지>'`	추적 파일 스테이징 및 버전 생성
`git commit` > `i` > 메시지 입력 > `esc` > `:wq`	Vim 에디터로 메시지 작성 및 버전 생성
`git commit --amend`	직전 커밋을 현재 커밋으로 덮어쓰기, Empty Commit (이후 경

확인 (Log)

확인(Log)은 생성한 버전 내용이나 내역, 변경 사항, 작업자 등을 확인하는 것을 말합니다.

`git log`	현재 브랜치의 버전 내역을 확인	
`git log -<숫자>`	숫자만큼만 최신 버전 내역 확인	`git log -2`
`git log --all`	모든 브랜치 내역 확인	
`git log --oneline`	간략한 버전 내역 확인	
`git log --graph`	그래프 형태로 버전 내역 확인	
`git reflog`	로컬의 모든 버전 관리 내역 확인	
`git show`	현재 브랜치의 최신 버전 확인	
`git show <브랜치>`	특정 브랜치의 최신 버전 확인	`git show dev`
`git show <해시>`	특정 버전 확인	`git show 1a2b`
`git blame <파일>`	특정 파일의 작업자 확인	`git blame ./s`
`git blame -L <시작>,<종료> <파일>`	특정 파일의 시작부터 종료 줄까지 작업자 확인	`git blame -L`
`git blame -L <시작> <파일>`	특정 파일의 시작부터 마지막 줄까지 작업자 확인	`git blame -L`
`git blame -L ,<종료> <파일>`	특정 파일의 처음부터 종료 줄까지 작업자 확인	`git blame -L`

브랜치 (Branch)

브랜치(Branch)는 프로젝트에서 여러 작업을 나눠 병렬로 진행할 수 있는, 버전 관리의 각 분기점을 의미합니다.

명령	설명	예시
`git branch`	로컬 브랜치 목록 확인	
`git branch -r`	원격 브랜치 목록 확인	
`git branch -a`	로컬 및 원격 브랜치 목록 확인	
`git branch <브랜치>`	브랜치 생성	`git branch dev`
`git branch -D <브랜치>`	브랜치 삭제	`git branch -D dev`

`git checkout <브랜치>`	브랜치 전환	`git checkout dev`
`git checkout -b <브랜치>`	브랜치 생성 및 전환	`git checkout -b dev`
`git checkout <해시>`	특정 버전 체크아웃	`git checkout 1a2b3c4d`
`git switch <브랜치>`	브랜치 전환 (v2.23)	`git switch dev`
`git swtich -c <브랜치>`	브랜치 생성 및 전환 (v2.23)	`git switch -c dev`

밀어내기 (Push)

밀어내기(Push)는 로컬 저장소의 버전 내역을 원격 저장소로 업로드하는 기능입니다.
강제 플래그(`--force`, `-f`)는 충돌을 무시하고 원격 저장소를 덮어쓰므로, 확실한 경우에만 사용해야 합니다.

명령	설명	예시
`git push <원격별칭> <브랜치>`	원격 저장소로 밀어내기	`git push origin dev`
`git push <원격별칭> --all`	원격 저장소로 모든 브랜치 밀어내기	`git push origin --all`
`git push <원격별칭> <브랜치> -f`	원격 저장소로 강제(Force) 밀어내기	`git push origin dev -f`
`git push <원격별칭> <브랜치> -u`	원격 저장소로 밀어내기 후 생략 가능	`git push origin dev -u`

당겨오기 (Pull)

당겨오기(Pull)는 원격 저장소의 버전 내역을 로컬 저장소로 다운로드하는 기능입니다.

명령	설명	
`git pull <원격별칭> <브랜치>`	원격 저장소에서 브랜치 당겨오기	`git pull or`
`git pull <원격별칭> --all`	원격 저장소에서 모든 브랜치 당겨오기	`git pull or`

가져오기 (Fetch)

가져오기(Fetch)는 원격 저장소의 최신 내역을 로컬의 원격 내역과 동기화하는 기능으로, 로컬 브랜치에는 영향을 주지 않습니다.

명령	설명	C
`git fetch`	현재 원격 저장소의 브랜치 목록 가져오기	
`git fetch <원격별칭>`	특정 원격 저장소의 브랜치 목록 가져오기	`git fet`
`git fetch -all`	모든 원격 저장소의 브랜치 목록 가져오기	
`git fetch --prune`	원격 저장소에서 브랜치 목록 가져와서 로컬의 원격 브랜치 목록 덮어쓰기	

비교 (Diff)

비교(Diff)는 두 개의 버전이나 파일 등의 차이를 서로 비교하는 기능입니다.

명령	설명	
`git diff <파일>`	파일의 수정 내용 확인	
`git diff <파일A> <파일B>`	A와 B 파일 비교	
`git diff <브랜치>`	특정 브랜치와 현재 브랜치 비교	`git diff dev`
`git diff <브랜치A> <브랜치B>`	A와 B 브랜치 비교	`git diff main dev`
`git diff <브랜치A>:<파일> <브랜치B>:<파일>`	A와 B 브랜치의 파일 비교	`git diff main:src/`
`git diff <해시>`	현재 버전과 특정 버전 비교	`git diff 1a2b3c4d`
`git diff <해시A> <해시B>`	A와 B 버전 비교	`git diff 1a2b3c4d`

작업 취소 (Rollback)

롤백(Rollback)은 현재 작업 중인 변경 사항을 모두 취소하고 버리는 것을 말합니다.

명령	설명	예시
`git checkout HEAD -- <파일>`	특정 파일 롤백	`git checkout HEAD -- ./src/main.js`
`git restore <파일>`	특정 파일 롤백 (v2.23)	`git restore ./src/main.js`
`git restore .`	모든 파일 롤백 (v2.23)	
`git reset --hard HEAD`	모든 파일 롤백	

초기화 (Reset)

초기화(Reset)는 특정 버전으로 이동하고 그 이후 버전 내역을 제거하는 기능입니다.

명령	설명	예시
`git reset --hard HEAD~<번호>`	번호만큼 이전 버전으로 리셋	`git reset --hard HEAD~2`
`git reset --hard HEAD~1`	직전 버전으로 리셋 (`1` 버전 전으로)	
`git reset --hard HEAD~`	직전 버전으로 리셋 (`1` 생략)	
`git reset --hard <해시>`	특정 버전으로 리셋	`git reset --hard 1a2b3c4`
`git reset --hard HEAD^`	마지막 버전을 삭제	
`git reset --hard`	수정 내용을 버림	
`git reset --soft`	수정 내용을 스테이징	
`git reset --mixed`	수정 내용을 스테이징하지 않음	

리돌리기 (Revert)

되돌리기(Revert)는 특정 버전을 취소하고 취소한 새로운 버전을 생성하는 기능입니다.

명령	설명	예시
`git revert <해시>`	특정 버전을 취소하고 새로운 버전 생성	`git revert 1a2b3c4d`

임시 저장 (Stash)

임시 저장(Stash)는 작업 중인 변경사항을 버전으로 생성하지 않고 별도로 저장하는 기능입니다.

명령	설명	예시
`git stash list`	임시 저장된 작업 목록 확인	
`git stash`	현재 작업을 임시 저장	
`git stash -a`	미추적 파일 포함, 임시 저장	
`git stash -m '<메시지>'`	메시지와 함께 현재 작업을 임시 저장	
`git stash -am '<메시지>'`	미추적 파일 포함, 메시지와 함께 현재 작업을 임시 저장	
`git stash show <번호>`	특정 번호의 임시 저장 내용 보기	`git stash show`
`git stash apply`	가장 최신의 임시 저장 내용을 현재 브랜치에 적용	
`git stash apply <번호>`	특정 번호의 임시 저장 내용을 현재 브랜치에 적용	`git stash apply`
`git stash drop`	가장 최신의 임시 저장 내용 삭제	
`git stash drop <번호>`	특정 번호의 임시 저장 내용 삭제	`git stash drop`
`git stash pop`	가장 최신의 임시 저장을 적용하고 목록에서 삭제	
`git stash clean`	모든 임시 저장 목록 삭제	

병합 (Merge)

병합(Merge)은 두 개의 브랜치를 하나로 합치는 기능입니다.

명령	설명	예시
`git merge <브랜치>`	현재 브랜치에 특정 브랜치 병합	`git merge dev`
`git merge --abort`	충돌 시, 병합 과정 중단	

병합을 통해 두 브랜치의 내용이 달라 충돌(Conflict)이 발생하는 경우, 충돌을 해결하고 다시 커밋해야 합니다.
'현재 변경 사항'은 현재 브랜치(`main`)의 작업 내용, '수신 변경 사항'은 병합할 브랜치(`dev`)의 작업 내용을 의미합니다.
충돌 해결 후 수정된 파일을 스테이징(`git add`)하고 병합 버전을 생성(`git commit`)해야 합니다.

```
<<<<<< HEAD (현재 변경 사항)
main / abc
=======
dev / xyz
>>>>>> dev (수신 변경 사항)
```

재배치 (Rebase)

재배치(Rebase)는 현재 브랜치의 내역을 대상 브랜치의 최신 버전 다음으로 배치(이동)하는 기능입니다.

```
(main)-- A - B - C
          \
(dev)----- D - E
```

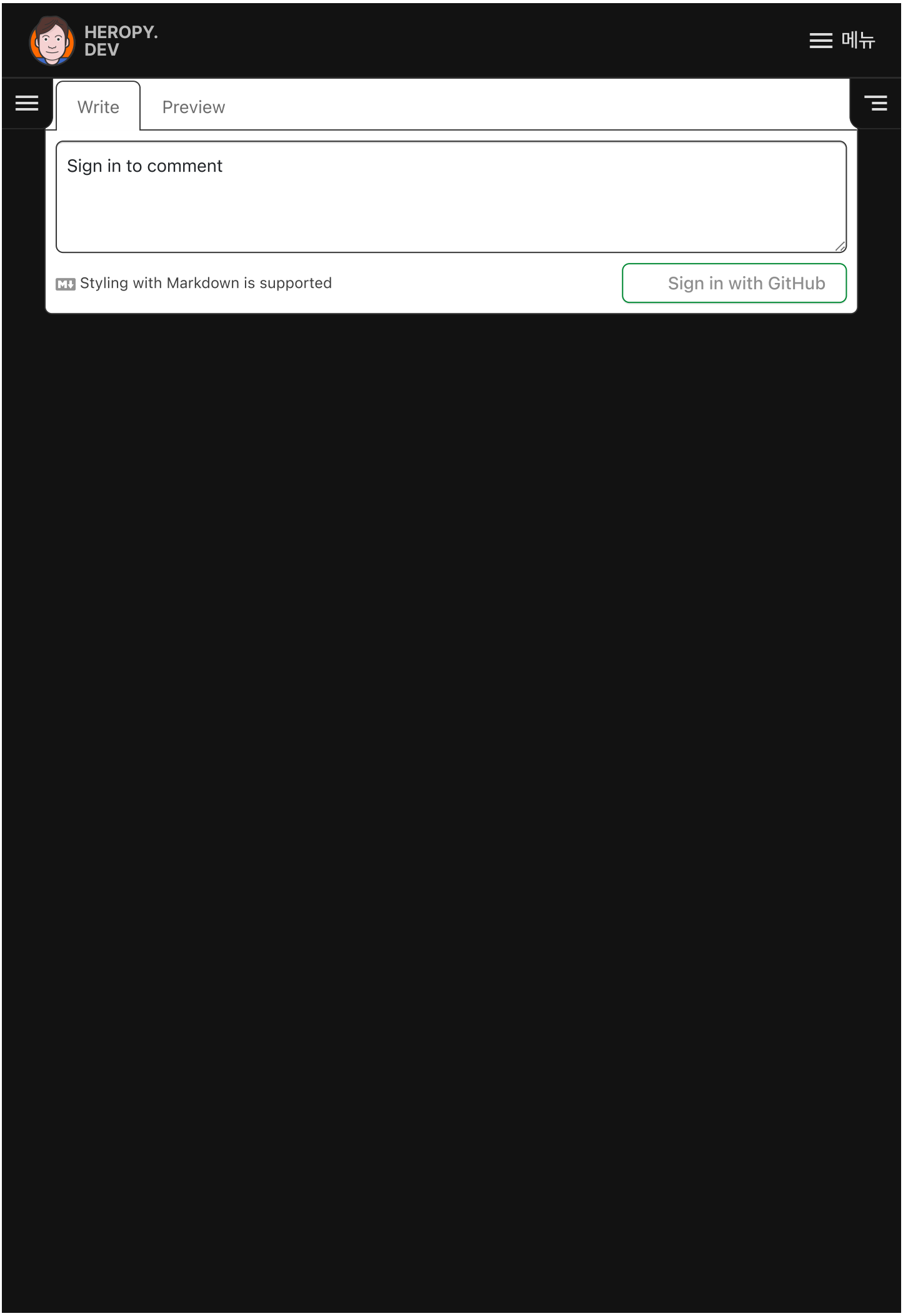
재배치 전 (Before)

배치 후 (After)

명령	설명	예시
`git rebase <브랜치>`	현재 브랜치를 대상 브랜치로 재배치	`git rebase main`
`git rebase --continue`	재배치 계속 진행	
`git rebase --abort`	재배치 과정 중단	

다음은 `dev` 브랜치를 `main` 브랜치로 재배치하는 과정입니다.

1. `git checkout dev`: 재배치할 브랜치로 전환.
2. `git rebase main`: 현재 브랜치(`dev`)를 대상 브랜치(`main`)로 재배치 시작.
3. 충돌(Conflict) 발생 시 해결.
4. `git add .`: 충돌 해결 후 스테이징.
5. `git rebase --continue`: 재배치 계속 진행.
6. 버전 메시지 수정 및 저장(`:wq`).
7. 3~6번 과정 반복 및 재배치 완료!


HEROPY.
DEV
메뉴
Write
Preview
Sign in to comment
Styling with Markdown is supported
Sign in with GitHub