9. 태그와 릴리즈

버전 태그 달기 (`git tag`)

✅ 1. 태그(Tag)란?

> 특정 커밋(commit)에 **버전 이름(스냅샷 라벨)** 을 붙여서
> 그 시점을 **고정된 기준점**으로 삼는 기능.

- 릴리즈용으로 자주 사용됨
- 주로 `v1.0.0`, `v2.1.3` 같은 형태
- lightweight 태그 vs annotated 태그

✅ 2. 왜 태그를 쓰는가?

목적	설명
✅ 배포 버전 관리	`v1.0.0` 같은 버전 마킹
✅ 릴리즈 기준점 기록	이전 릴리즈와의 비교 가능
✅ CI/CD 트리거	특정 태그에만 배포 설정
✅ 긴 SHA 대신 가독성 있는 이름 제공	`a9b3d9c` 대신 `v2.0.0`

✅ 3. 태그 종류

유형	설명	GitHub Release와 연동
Lightweight 태그	단순 커밋 포인터 (이름만 있음)	❌ 불가
Annotated 태그	이름 + 메타데이터 + 서명 가능	✅ 가능

✅ 4. 태그 달기

🟡 Lightweight 태그

```
git tag v1.0.0
```

- 가장 간단한 방식
- 커밋에 이름만 붙임

🟢 Annotated 태그 (권장)

```
git tag -a v1.0.0 -m "Release version 1.0.0"
```

- `-a`: annotated
- `-m`: 태그 메시지
- 태그 작성자, 날짜 등 포함

🔹 과거 커밋에 태그 달기

```
git log --oneline
git tag -a v1.0.0 <커밋해시> -m "Initial release"
```

✅ 5. 태그 목록 확인

```
git tag
# 또는 정렬:
git tag --sort=creatordate
```

✅ 6. 태그 삭제

```
git tag -d v1.0.0            # 로컬에서 삭제
git push origin :refs/tags/v1.0.0  # 원격에서 삭제
```

✅ 7. 태그 푸시 / 가져오기

🔼 태그 푸시

```
git push origin v1.0.0      # 개별 푸시
git push origin --tags      # 전체 태그 푸시
```

> 주의: `git push`만으로는 태그가 자동 전송되지 않음

🔽 태그 가져오기

```
git fetch --tags
```

✅ 8. GitHub Release와 태그 연결

1. GitHub → [Releases] 탭
2. `Draft a new release`
3. 기존 태그 선택 또는 새 태그 작성
4. 릴리즈 노트 작성
5. 배포 아티팩트 (.zip, .tar 등) 업로드 가능

✅ 9. 태그 기반 체크아웃

```
git checkout v1.0.0
```

- 해당 시점으로 소스 코드 상태 전환
- ⚠️ Detached HEAD 상태임

✅ 10. 태그로 CI 배포 자동화 예시 (GitHub Actions)

```
on:
  push:
    tags:
      - 'v*.*.*'
```

- 태그 푸시 시 CI/CD 자동 실행

🧠 요약

명령	설명
`git tag`	태그 목록 보기
`git tag v1.0.0`	lightweight 태그
`git tag -a v1.0.0 -m "메시지"`	annotated 태그
`git push origin v1.0.0`	태그 푸시
`git push origin --tags`	전체 태그 푸시
`git tag -d v1.0.0`	태그 삭제
`git fetch --tags`	원격 태그 가져오기
`git checkout v1.0.0`	특정 태그로 이동

📦 릴리즈를 명확하게 정의하고,
🚀 배포 자동화를 트리거하고,
🔍 코드 히스토리를 구조화하는 데 있어 태그는 **사실상 필수 도구**이다.

lightweight vs annotated tag

✅ 1. 개념 비교

구분	Lightweight Tag	Annotated Tag
📌 정의	단순히 커밋에 이름 붙이기	이름 + 메타데이터 + 메시지 포함
📎 구조	이름 → 커밋 객체만 가리킴	별도 태그 객체(`tag object`) 생성
📝 설명 메시지	❌ 없음	✅ 가능 (`-m "메시지"`)
👤 작성자 정보	❌ 없음	✅ 포함됨 (작성자, 날짜)
🔐 GPG 서명	❌ 불가	✅ 가능 (`-s`)
GitHub Release 연동	❌ 불가	✅ 가능
용도	임시 마킹, 로컬 테스트용	릴리즈, 배포 기록, 공식 버전 태깅

✅ 2. 명령어 비교

🔹 Lightweight

```
git tag v1.0.0
```

> 그냥 이름만 붙임 (메시지나 작성자 없음)

🔹 Annotated

```
git tag -a v1.0.0 -m "Release version 1.0.0"
```

> `-a`는 annotated, `-m`으로 메시지 추가

✅ 3. 저장 구조 (깊이 있는 차이)

🔸 Lightweight

```
refs/tags/v1.0.0 → commit
```

- 단순 포인터처럼 커밋을 가리킴
- 태그 그 자체는 **별도 객체 아님**

🔸 Annotated

```
refs/tags/v1.0.0 → tag object → commit
```

- 태그 객체(tag object) 내부에:
  - 태그 이름
  - 작성자
  - 날짜
  - 메시지
  - GPG 서명 (선택)

✅ 4. GitHub에서의 태그 차이

항목	Lightweight	Annotated
GitHub Releases 탭 표시	❌ 안 보임	✅ 릴리즈 생성 가능
릴리즈 아카이브 다운로드	❌ 불가	✅ zip, tar.gz 제공
자동 릴리즈 노트	❌	✅ 메시지를 기반으로 생성됨

✅ 5. 언제 어떤 태그를 써야 하나?

상황	추천 태그
로컬에서 임시 테스트, 특정 시점 체크	Lightweight
버전 릴리즈, CI/CD 배포 트리거, 기록 보관	Annotated (권장)
GitHub Release 생성	Annotated (필수)
태그 서명 보안 필요	Annotated (`-s` 사용)

✅ 6. 태그 유형 변환

- lightweight → annotated는 **변환 불가**
- 삭제하고 다시 만들 것:

```
git tag -d v1.0.0
git tag -a v1.0.0 -m "annotated로 재작성"
```

🧠 요약 정리표

항목	Lightweight	Annotated
커밋 포인터	✅	✅
설명 메시지	❌	✅
작성자, 날짜	❌	✅
GPG 서명	❌	✅ (`-s`)
GitHub Release	❌	✅
릴리즈 용도	❌	✅
생성 명령	`git tag v1.0.0`	`git tag -a v1.0.0 -m "..."`

📌 **실전에서는 무조건 annotated 태그를 기본값처럼 사용**하고,
단순한 실험이나 테스트 상황에서만 lightweight를 쓰는 게 좋다.

이전 커밋에 태그 추가

✅ 1. 먼저 커밋 해시 확인

```
git log --oneline
```

예시 출력:

```
a3c1e2f (HEAD -> main) Fix typo in README
9bd72f0 Add user profile page
3f1ad84 Initial commit
```

✅ 2. Lightweight 태그로 이전 커밋에 태그 달기

```
git tag v1.0.0 9bd72f0
```

- 단순히 이름만 붙는 lightweight 태그

✅ 3. Annotated 태그로 이전 커밋에 태그 달기 (권장)

```
git tag -a v1.0.0 9bd72f0 -m "Release version 1.0.0"
```

- `-a` : annotated
- `-m` : 태그 메시지
- `9bd72f0` : 태그를 달 커밋 해시

✅ 4. 태그 확인

```
git show v1.0.0
```

✅ 5. 태그를 원격 저장소(GitHub 등)에 푸시

```
git push origin v1.0.0
```

> 여러 개 푸시할 경우:

```
git push origin --tags
```

🧠 요약

명령어	설명
`git log --oneline`	커밋 해시 확인
`git tag v1.0.0 <해시>`	lightweight 태그
`git tag -a v1.0.0 <해시> -m "..."`	annotated 태그
`git push origin v1.0.0`	태그 푸시

GitHub 릴리즈 페이지 연동

✅ 1. 릴리스란 무엇인가?

> GitHub에서 **버전 태그(`git tag`)에 설명, 파일, 노트 등을 덧붙여**
> 사용자에게 공식 릴리스를 제공하는 기능.

- 배포 대상 버전을 명확히 함 (`v1.0.0`)
- 변경사항 요약, 파일 첨부 가능
- Git 태그와 연결되어 있음
- `Releases` 탭에 따로 표시됨

✅ 2. 릴리즈 페이지로 가는 법

방법 1: GitHub 저장소 우측 상단

👉 `Code`, `Issues`, `Pull requests` 옆에
🔽 `Releases` 탭 클릭 → `Draft a new release`

방법 2: 직접 URL 이동

```
https://github.com/<사용자명>/<저장소명>/releases
```

✅ 3. 릴리즈 생성 절차

1. `Draft a new release` 클릭
2. 아래 항목 입력:

항목	설명
🚀 Tag version	기존 태그 선택 or 새 태그 입력
🪄 Target	병합 기준 커밋 (보통 main)
📝 Release title	사용자에게 보일 릴리즈 제목
📄 Description	주요 변경 사항, 사용법 등
📎 Attach binaries	실행파일, zip, tar.gz 등 업로드
✅ 옵션	Pre-release 여부, Latest release 표시 여부

1. `Publish release` 버튼 클릭

✅ 4. Git에서 푸시한 태그로 릴리즈 만들기

1. 먼저 태그 생성 및 푸시

```
git tag -a v1.2.0 -m "Release version 1.2.0"
git push origin v1.2.0
```

2. GitHub → Releases → `Draft new release`

→ Tag로 `v1.2.0` 선택 → 설명 작성 → Publish

> 💡 푸시된 태그가 있어야 Tag 목록에서 선택 가능

✅ 5. 릴리즈 자동화: GitHub Actions 예시

```yaml
on:
  push:
    tags:
      - 'v*.*.*'

jobs:
  release:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - run: echo "릴리즈 버전입니다: ${{ github.ref }}"
```

- 태그 푸시 시 자동 릴리즈 트리거 가능
- `actions/create-release` 같은 GitHub 액션 사용 가능

✅ 6. 릴리즈와 태그의 관계

태그	릴리즈
Git 내부의 커밋 포인터	GitHub UI 상의 태그 설명/파일
로컬에서도 사용 가능	GitHub에서만 보임
`git tag`로 생성	웹에서 작성하거나 연결 가능
메시지 없음도 가능	메시지 필수 (릴리즈 노트)

✅ 7. 릴리즈 예시

- https://github.com/nodejs/node/releases
- https://github.com/vercel/next.js/releases

✅ 8. 자주 묻는 질문 (FAQ)

질문	답변
태그는 있는데 릴리즈가 안 보여요	태그만 있고 릴리즈를 수동으로 안 만들었기 때문
릴리즈 삭제 시 태그도 사라지나요?	❌ 태그는 Git에 남아 있음. 별도 삭제 필요
binary 파일 배포도 가능하나요?	✅ `.zip`, `.exe`, `.tar.gz` 등 업로드 가능

🧠 요약

단계	설명
① Git 태그 생성	`git tag -a v1.0.0 -m "..."`
② 태그 푸시	`git push origin v1.0.0`
③ GitHub → Releases → Draft	태그 선택 후 릴리즈 설명 작성
④ 릴리즈 게시	`Publish Release` 버튼
⑤ (선택) 자동화	GitHub Actions로 자동 릴리즈 설정 가능

`changelog` 작성 전략

✅ 1. `CHANGELOG`란?

> 프로젝트의 **버전별 변경 이력**을 **명확하게 문서화**한 파일.
> 일반적으로 루트 디렉토리에 `CHANGELOG.md` 또는 `CHANGES.md`로 존재.

✅ 2. 왜 필요한가?

이유	설명
👀 사용자에게 변경 내용 전달	무엇이 변경/추가/삭제/수정되었는지 한눈에
💉 QA와 테스트 기준 제공	릴리즈마다 테스트 범위 파악
📦 배포 프로세스 자동화와 연계	릴리즈 노트 자동 생성 기반
🧑‍🤝‍🧑 오픈소스 협업 가이드라인	타 개발자와의 정보 공유, 리뷰 기준 제공
✅ 버전 태그와 연동	Git 태그, GitHub Release와 일관성 유지

✅ 3. 작성 포맷 예시: [Keep a Changelog]

```
# Changelog

All notable changes to this project will be documented in this file.

The format is based on [Keep a Changelog](https://keepachangelog.com)
and this project adheres to [Semantic Versioning](https://semver.org).

## [Unreleased]
### Added
- 로그인 기능에 2FA 추가

### Changed
```

```
- 회원가입 시 이메일 인증 방식 개선

### Fixed
- iOS Safari에서 팝업 창 안 닫히는 오류 수정

---

## [1.2.0] - 2024-11-15
### Added
- 프로필 페이지에 다크 모드 지원 추가

### Fixed
- 일부 이미지 로딩 오류 해결

## [1.1.0] - 2024-09-01
### Added
- 소셜 로그인(Google, Facebook) 기능

## [1.0.0] - 2024-06-01
### Added
- 기본 로그인/회원가입
- 게시판 CRUD 기능
```

✅ 4. 분류 항목 제안 (기본)

항목	의미
🔹 `Added`	새 기능 추가
🔸 `Changed`	기존 기능 변경
🔧 `Fixed`	버그 수정
❌ `Removed`	삭제된 기능
⚠️ `Deprecated`	앞으로 제거 예정 기능
💉 `Security`	보안 관련 패치

✅ 5. 좋은 CHANGELOG의 조건

항목	설명
✅ **버전별 구분**	`## [1.2.0] - YYYY-MM-DD` 형태
✅ **항목별 분류**	`Added`, `Changed` 등 그룹화
✅ **명확한 설명**	한 문장으로 핵심 요약
✅ **작성자 관점 X**	내부 구현보다 사용자 입장에서

항목	설명
✅ **자동화 가능**	Conventional Commits 기반 자동 생성도 고려

✅ 6. 커밋 메시지 → CHANGELOG 자동 생성 전략

💡 Conventional Commits 사용 시 자동화 가능

커밋 메시지 예시:

```
feat(login): add OAuth2 support
fix(signup): fix email regex
chore: update dependencies
```

→ `release-please`, `standard-version`, `semantic-release` 등의 툴로 자동 changelog 생성 가능

✅ 7. 릴리즈와 연결하는 방법

- Git 태그 (`git tag v1.2.0`)와 같은 버전명을 사용
- GitHub Release에 CHANGELOG 내용을 그대로 붙여 사용
- CI에서 자동 릴리즈 노트로 생성 (예: GitHub Actions)

✅ 8. 추천 전략

상황	전략
OSS, 팀 프로젝트	`Keep a Changelog` 스타일 명시적 구분
빠른 릴리즈 환경	Conventional Commits + 자동 생성
내부 서비스	이슈 트래커(예: Jira)와 changelog 자동 연동
긴 릴리즈 주기	수동으로 요약 정리 + 릴리즈마다 리뷰

✅ 9. `.github/CHANGELOG_TEMPLATE.md`로 템플릿 관리 (선택)

```
## [VERSION] - YYYY-MM-DD

### Added
-

### Changed
-

### Fixed
-

### Removed
-
```

자동화 도구에서 사용 가능하게 템플릿 지정

🧠 요약

항목	설명
`CHANGELOG.md`	변경사항 이력 파일
포맷	버전 + 날짜 + 항목별 구분
전략	사용자 중심 요약, 태그 연동
자동화	Conventional Commits 기반 가능
툴	`semantic-release`, `release-please` 등