

깃 설정과 기본 명령어 정리

Version Control and
Git & Github basics

강환수 교수



AI Experts
Who Lead
The Future

01

깃 설정 명령어

- 맥(lf)과 윈도우(crlf) 간의 자동 변환
 - `git config --global core.autocrlf true`
- 사용자 설정
 - `$ git config --global user.name "John Doe"`
 - `$ git config --global user.email johndoe@example.com`
- 뉴라인 경고 발생 없애기(옵션)
 - `git config --global core.safecrlf false`

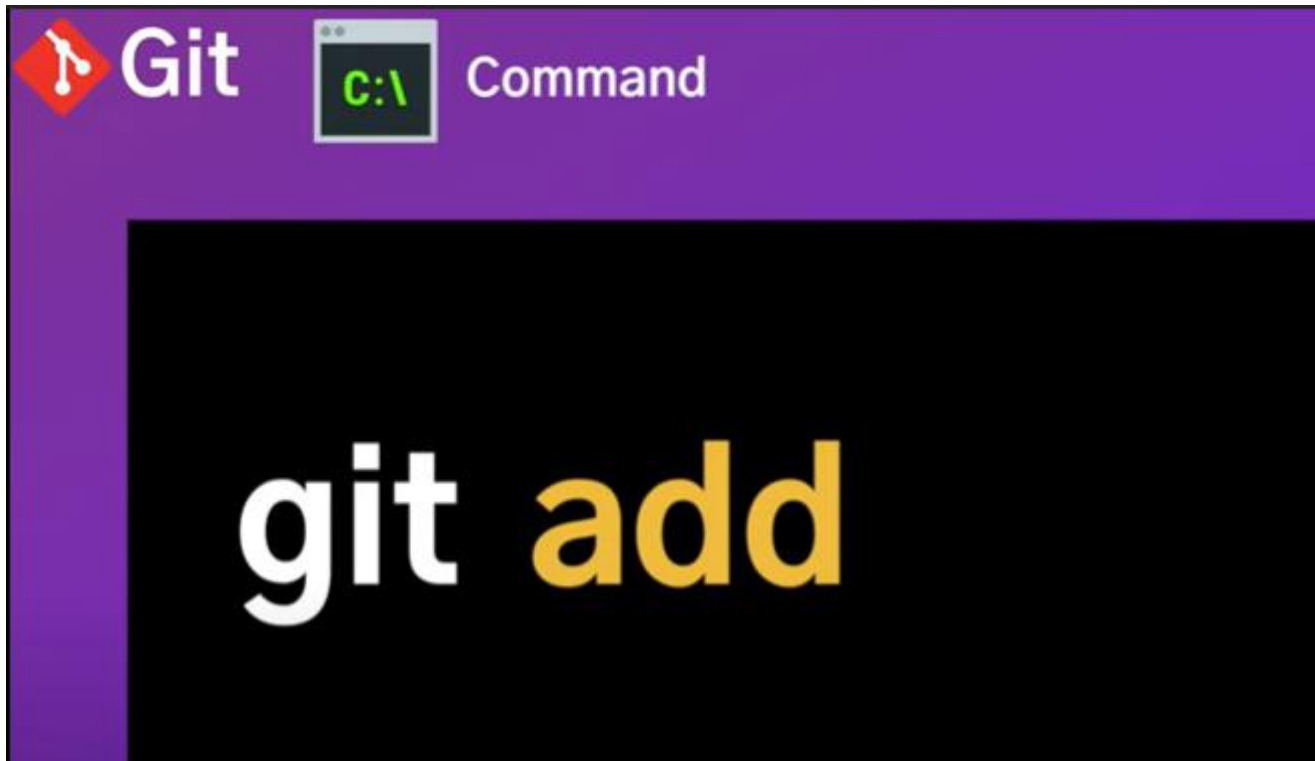
- `git config core.editor code`
- `git config core.editor "code --wait"`
- `git config core.editor notepad`
- `git config --global core.editor code`

AI Experts
Who Lead
The Future

02

깃 저장소 생성과 삭제

- 맨 앞에 항상 git
 - \$ git add
 - \$ git commit



- "git init" 명령어 실행
 - 프로젝트(소스코드들이 있는 디렉토리)를 git repository로 만들기 위해서 사용
 - 디렉토리를 git repository로 만들어야 git으로 버전 관리가 가능

- **\$ git init [project-name]**
 - 현재 하부 폴더에 새로운 저장소 [project-name]를 생성
 - [project-name] 없다면, 현재 폴더를 저장소로 생성, 다음과같은 의미
 - **\$ git init .**
- **새로운 로컬 저장소를 생성하고 이름을 정합니다.**
 - Initialized empty Git repository in [저장소전체경로]/.git/

- Git을 사용한다면 설치 후 꼭 미리 설정해두기를 권장
 - 커밋을 할 때 사용할 이름과 이메일을 지정
 - 저장소에 변경 사항을 추가하는 Commit 작업을 할 때 누구의 작업인지를 기록하는 것이 매우 중요
 - GitHub의 사용자를 연결할 때도 사용
- 저장소로 이동 환경설정
 - 현재 로컬 저장소에 적용할 사용자 정보를 설정합니다
 - `$ git config user.name "[name]"`
 - 자신이 생성한 커밋(commit)에 들어갈 이름을 설정합니다
 - `$ git config user.email "[email address]"`
 - 자신이 생성한 커밋에 들어갈 이메일 주소를 설정합니다

- [지역저장소]/.git/config 파일

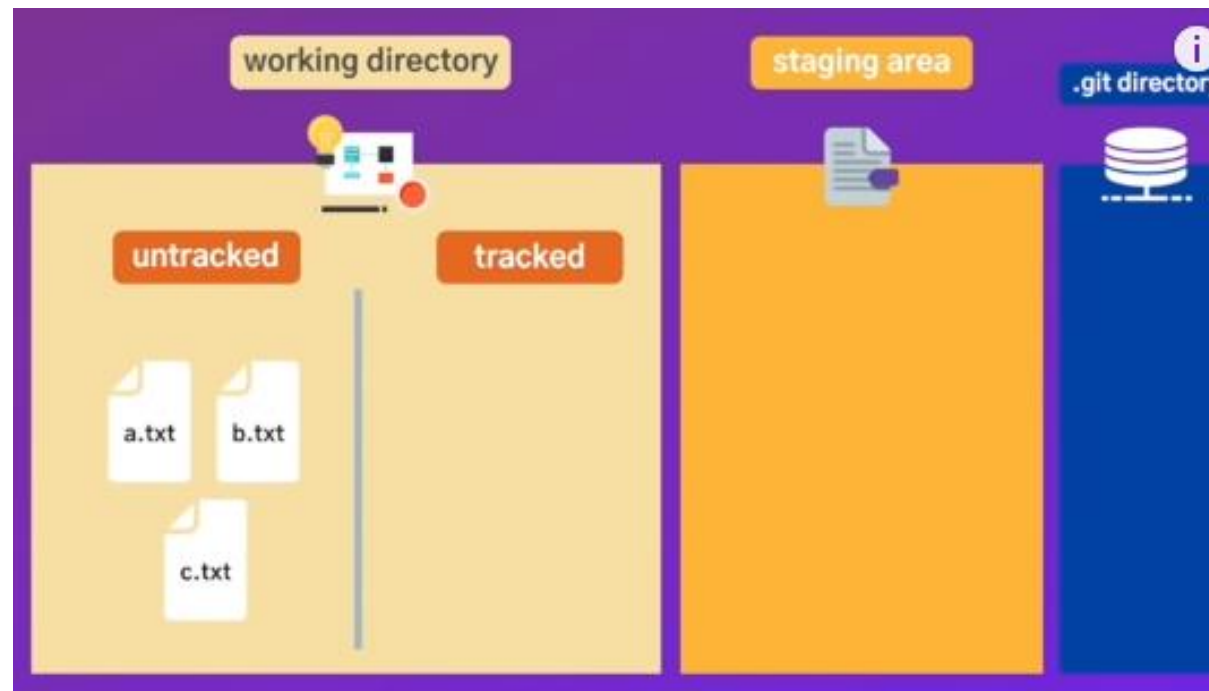
```
PC@DESKTOP-482NOAB MINGW64 /c/oss/git/02w/basic (main)
$ cat .git/config
[core]
    repositoryformatversion = 0
    filemode = false
    bare = false
    logallrefupdates = true
    symlinks = false
    ignorecase = true
[user]
    name = hskang
    email = hskang
```

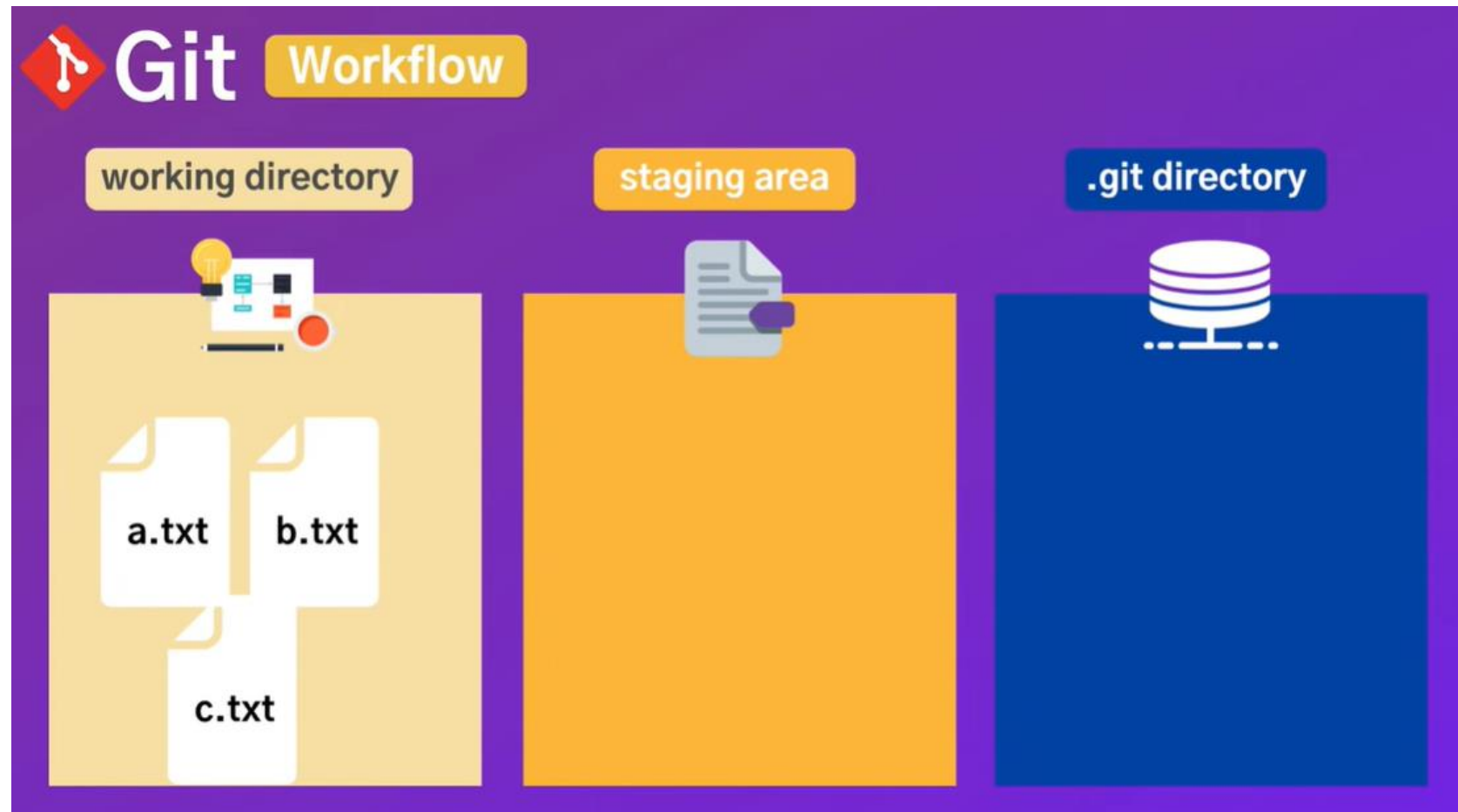
AI Experts
Who Lead
The Future

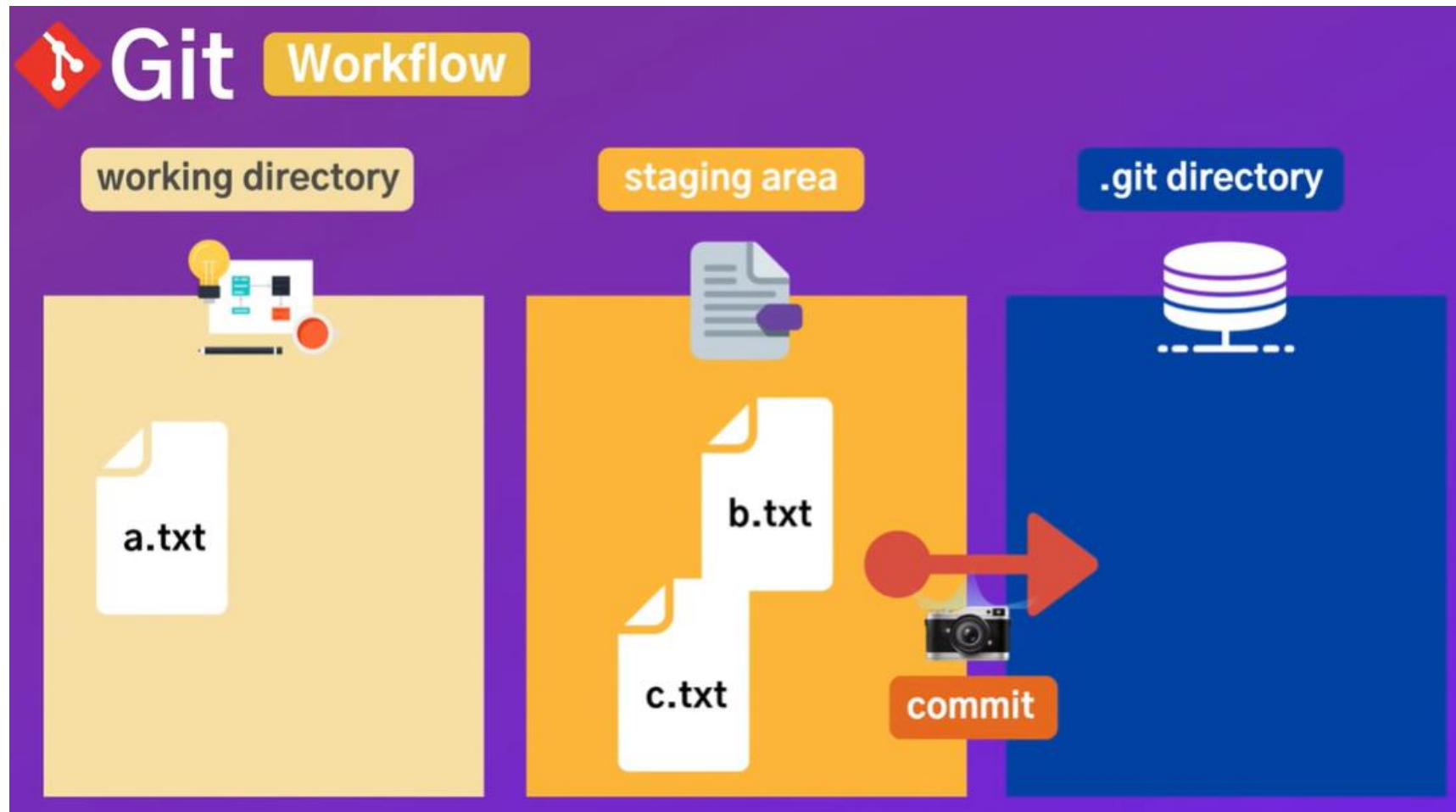
03

깃 커밋 명령어

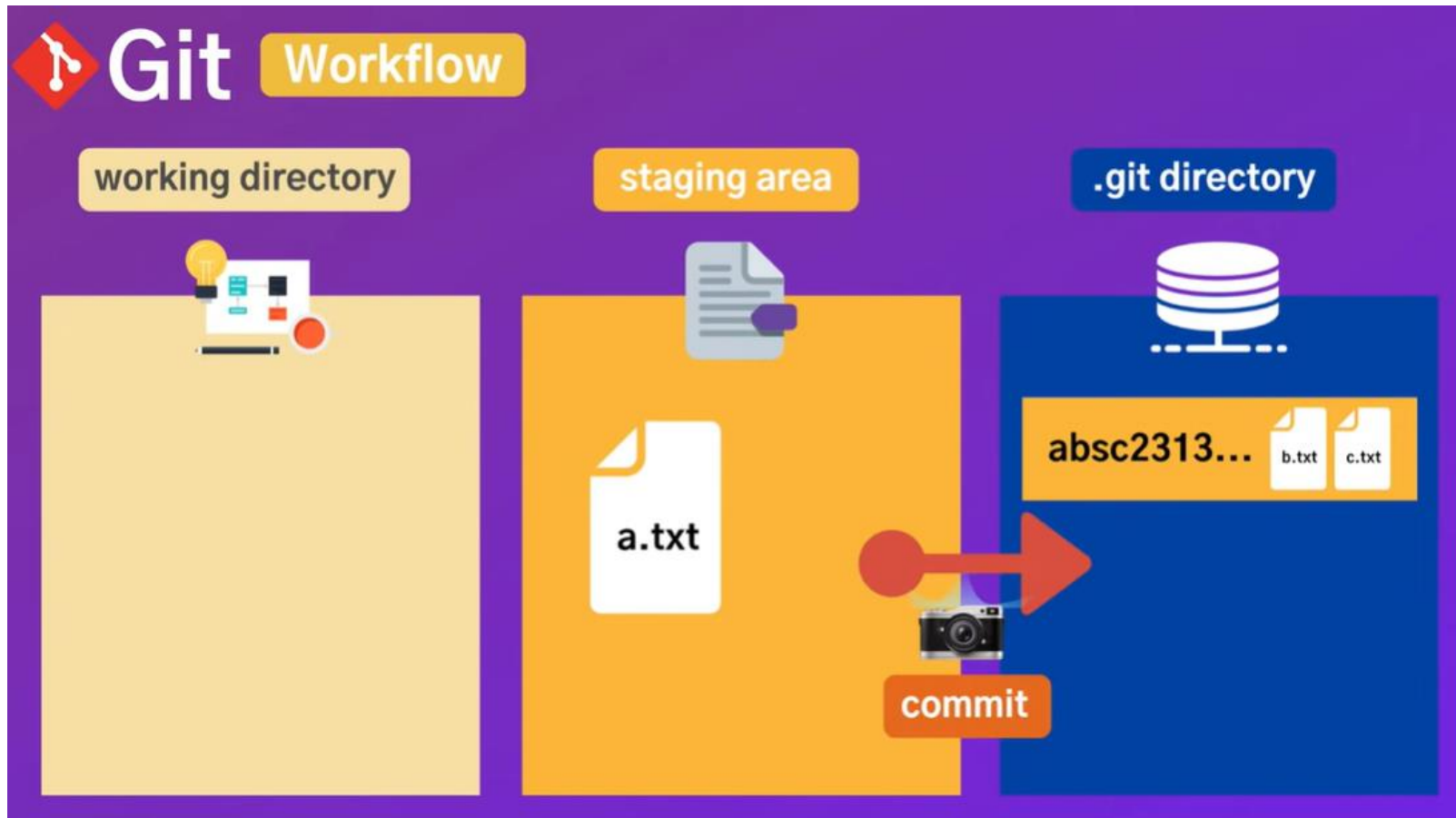
- **Untracked vs tracked**
 - 깃의 관리 대상 인지 아닌지
 - 형상(버전) 관리의 대상인지 아닌지
 - **Untracked**
 - 처음 생성만 해 놓은 파일
 - **Tracked로 이동**
 - `$ git add [file]`







- 버전 관리를 위한 git 디렉토리에 반영

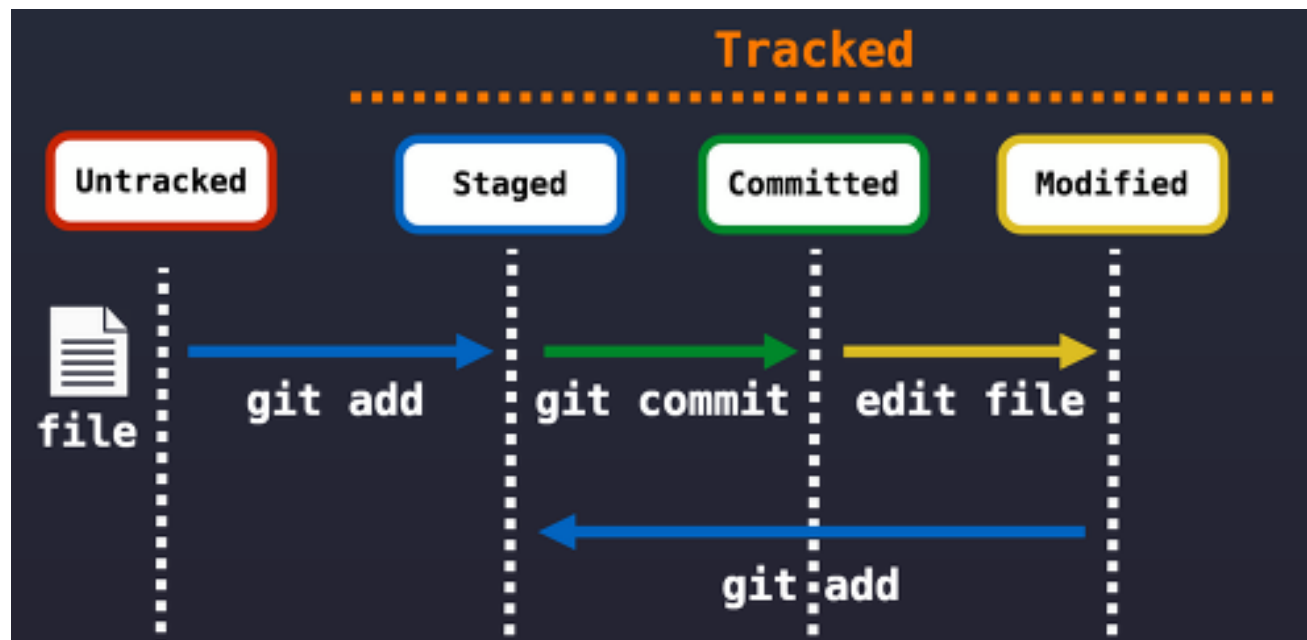




- **commit history 확인**
 - 마지막 커밋인 head부터 이전 모든 이력 로그 표시
 - Author 영역의 이름과 이메일 주소
 - git config 명령을 통해 세팅했던 user.name / user.email 값이 표기

Untracked vs Tracked

깃과 깃허브 Python language



AI Experts
Who Lead
The Future

03

깃 저장소 간략한 상태 정보

- `$ git status -s`
 - 옵션 `--short [file]`
 - 파일 상태를 짧막하게 확인하기
 - File 있으면 file에 대한 것만 표시, 없으면 모든 파일 표시

```
$ git status -s
```

Staging Status	Working Status	
M		README
M	M	Rakefile
A		lib/git.rb
M		lib/simplegit.rb
?	?	LICENSE.txt

? = Not tracked
A = Added
M = Modified

- ??
 - 아직 추적하지 않는(untracked) 새 파일 앞에 표시
- A□ 표시
 - 새로 생성해 add해 Staged된 파일 의미
- M□
 - 수정한 파일을 add한 상태
- □M README
 - 커밋한 이후에 아직 Staged 상태로 추가하지는 않았고 내용은 수정되었다는 의미
 - Add 한 이후에 파일 수정하고, 커밋한 상태
 - 결국 add, commit이 필요하다는 의미
- M□ lib/simplegit.rb
 - 파일은 내용을 변경하고 Staged 상태로 추가까지 한 상태
 - 최초 생성이 아닌 파일로서 수정한 파일, 커밋 준비가 완료된 상태
- MM Rakefile
 - 변경하고 Staged 상태로 추가한 후 또 내용을 변경해서 Staged 이면서 Unstaged 상태인 파일

왼쪽에는 Staging Area
에서의 상태 표시

바로 이전에 커밋해서
staging area는 깨끗한
상태: 빈문자

```
$ git status -s
M README
MM Rakefile
A lib/git.rb
M lib/simplegit.rb
?? LICENSE.txt
```

WD를 바로 add해서
WD가 깨끗한 상태

오른쪽에는 Working Tree
에서의 상태를 표시

- **?? => A□**
 - Untracked에서 SA로 add된 상태의 변화
- **□M => M□**
 - WD에서 수정된 파일을 add한 상태의 변화
- **□□ => □M**
 - 커밋한 이후에 파일 수정 시 상태의 변화
- **M□ => MM**
 - Add한 이후에 다시 파일 수정한 상태의 변화
- **MM => □M**
 - MM에서 커밋한 상태의 변화

AI Experts
Who Lead
The Future

04

깃 저장소 파일 수정과 staging 취소

- 깃 버전 2.23부터
- 스테이지 파일의 수정 후 취소
 - 수정 후
 - Modified
 - WD의 수정을 취소 하려면
 - `$ git restore fname`
- 스테이지 파일로 add한 것을 취소
 - Add 후
 - SA로 올린 것을 취소(HEAD에 있는 내용으로 수정) 하려면
 - `$ git restore --staged fname`