

# 깃 설정과 기본 명령어

Version Control and  
Git & Github basics

강환수 교수

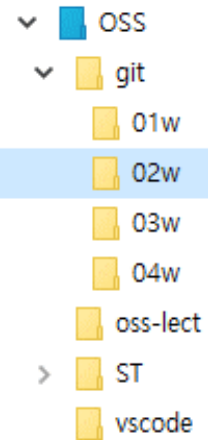


AI Experts  
Who Lead  
The Future

# 01

## 깃 설정 명령어

- 지역 저장소 생성 기준
  - 처음, 기준을 정하고 하부에 만들어 보도록
    - 폴더 git/02w 하부에



- 4가지 실행 방법
  - 메뉴 이용
  - 원하는 폴더에서 팝업 메뉴 실행
  - 탐색기에서 git bash 명령어 입력
  - 윈도우 검색에서 git bash 명령어 입력
- 실습 절차
  - 해당 폴더에서 마우스 우클릭 > git bash 실행
- 'Bash(배시)'
  - 'Bourne Again Shell'
  - 스티브 본(Steve Bourne)이라는 사람이 개발한 최초의 유닉스 '셸 프로그램'인 sh의 확장판이라는 의미

- Command Line Interface
- 현재 폴더 확인
  - \$ pwd
- 도움말 보기
  - \$ git help
  - \$ git --help
  - \$ git --help -a
  - \$ git help --all
  - \$ git add -h
    - 명령어 add에 대한 도움말
- 버전 보기
  - \$ git version
  - \$ git --version

- 폴더

- C:\Users\PC
- 파일

PC: 윈도우 로그인 사용자 계정

- .gitconfig

- 텍스 파일 내용 보기 cat

- \$ cat ~/.gitconfig
- ~

- Git bash 기본 폴더

\$ ~

bash: /c/Users/PC: Is a directory

PC@DESKTOP-482NOAB MINGW64 /c/[git tutorial]

\$ pwd

/c/[git tutorial]

# \$ cat ~/.gitconfig

```
PC@DESKTOP-482NOAB MINGW64 /c/[git tutorial]
```

```
$ cat ~/.gitconfig
```

```
[filter "lfs"]
    clean = git-lfs clean -- %f
    smudge = git-lfs smudge -- %f
    process = git-lfs filter-process
    required = true

[user]
    name = Python Kang
    email = ai7dnn@gmail.com

[credential]
    helper = manager-core

[difftool "sourcetree"]
    cmd = ' ' \"$LOCAL\" \"$REMOTE\"

[mergetool "sourcetree"]
    cmd = ' ' ' '
    trustExitCode = true

[core]
    longpaths = true
    autocrlf = true
```

```
PC@DESKTOP-482NOAB MINGW64 /c/[git tutorial]
```

```
$
```

- 맥(lf)과 윈도우(crlf) 간의 자동 변환
  - `git config --global core.autocrlf true`
- 뉴라인 경고 발생 없애기
  - `git config --global core.safecrlf false`

```
[filter "lfs"]
    clean = git-lfs clean -- %f
    smudge = git-lfs smudge -- %f
    process = git-lfs filter-process
    required = true

[user]
    name = Python Kang
    email = ai7dnn@gmail.com

[credential]
    helper = manager-core

[difftool "sourcetree"]
    cmd = "W"$LOCALW" W"$REMOTEW"

[mergetool "sourcetree"]
    cmd = ""
    trustExitCode = true

[core]
    longpaths = true
    autocrlf = true
```



- **깃 전체(global) 사용자 설정**
  - 사용자 설정 전체
    - `$ git config --global user.name "John Doe"`
    - `$ git config --global user.email johndoe@example.com`
  - 로컬 시스템에서 Git 커밋을 하면 항상 이 정보가 기본적으로 사용
- **전체 설정 확인**
  - `$ git config --list`
- **부분 설정 확인**
  - `$ git config user.name`
  - `$ git config user.email`

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

```
PC@DESKTOP-482NOAB MINGW64 /c/[git tutorial]/git-cmd (master)
$ git config core.editor
notepad
```

```
PC@DESKTOP-482NOAB MINGW64 /c/[git tutorial]/git-cmd (master)
$ git config core.editor code
```

```
PC@DESKTOP-482NOAB MINGW64 /c/[git tutorial]/git-cmd (master)
$ git config core.editor
code
```

```
PC@DESKTOP-482NOAB MINGW64 /c/[git tutorial]/git-cmd (master)
$ git config -e
```

```
PC@DESKTOP-482NOAB MINGW64 /c/[git tutorial]/git-cmd (master)
$ git config core.editor notepad
```

```
PC@DESKTOP-482NOAB MINGW64 /c/[git tutorial]/git-cmd (master)
$ git config core.editor
notepad
```

```
PC@DESKTOP-482NOAB MINGW64 /c/[git tutorial]/git-cmd (master)
$ git config -e
hint: waiting for your editor to close the file... unix2dos: converting file C:/[git
tutorial]/git-cmd/.git/config to DOS format...
dos2unix: converting file C:/[git tutorial]/git-cmd/.git/config to Unix format..
```

- **\$ git config --list**

- 마지막 : 이후에 'q' 입력

```
PC@DESKTOP-482NOAB MINGW64 /c/[git tutorial]
$ git config --list
diff.astextplain.textconv=astextplain
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
http.sslbackend=openssl
http.sslcainfo=C:/Program Files/Git/mingw64/ssl/certs/ca-bundle.crt
core.autocrlf=true
core.fscache=true
core.symlinks=false
core.editor=notepad
pull.rebase=false
credential.helper=manager-core
credential.https://dev.azure.com.usehttppath=true
init.defaultbranch=master
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
user.name=Python Kang
user.email=ai7dnn@gmail.com
credential.helper=manager-core
difftool.sourcetree.cmd='' "$LOCAL" "$REMOTE"
```

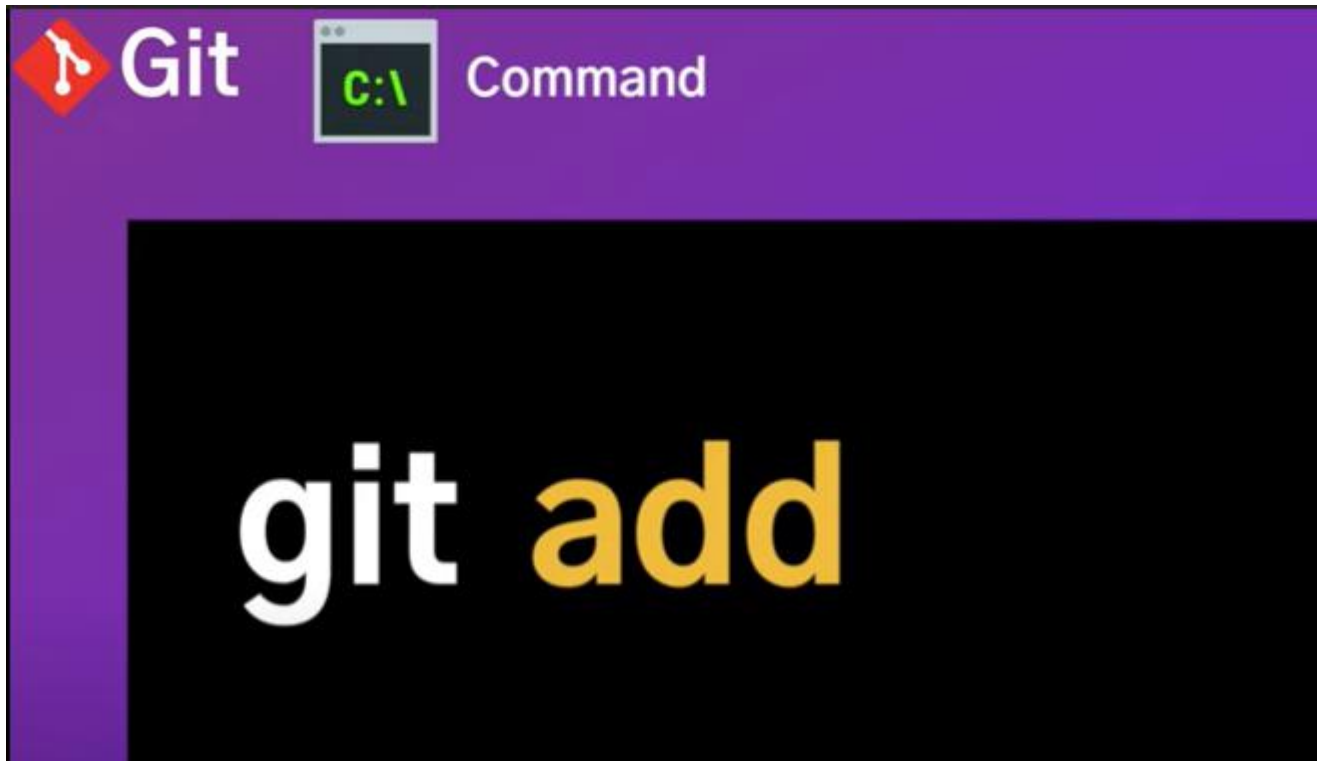
- 현재 폴더(디렉토리) 확인, print working dicrectory, linux 명령어
  - \$ pwd
- 버전 확인
  - \$ git --version
- 현재 설정 확인
  - \$ git config --list
- 사용자 이름과 이메일 설정
  - \$ git config --global user.name "John Doe"
  - \$ git config --global user.email johndoe@example.com
- 매번 사용자 이름과 이메일 설정이 다르다면
  - \$ git config user.name "John Doe"
  - \$ git config user.email johndoe@example.com

AI Experts  
Who Lead  
The Future

## 02

### 깃 저장소 생성과 삭제

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



- 폴더
  - C:\WossWgit\W02w

- "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/

- **\$ git init basic**
  - 폴더 basic을 저장소로 생성 사용
- **\$ cd basic**
- **\$ ls -al**
  - 하부 폴더 확인

- 전체 폴더를 삭제하거나 하부 폴더 .git 삭제
  - `$ rm -rf .git`
    - `-f` : 강제로 파일이나 디렉토리를 삭제
    - `-r` : 디렉토리 내부의 모든 내용을 삭제

- 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

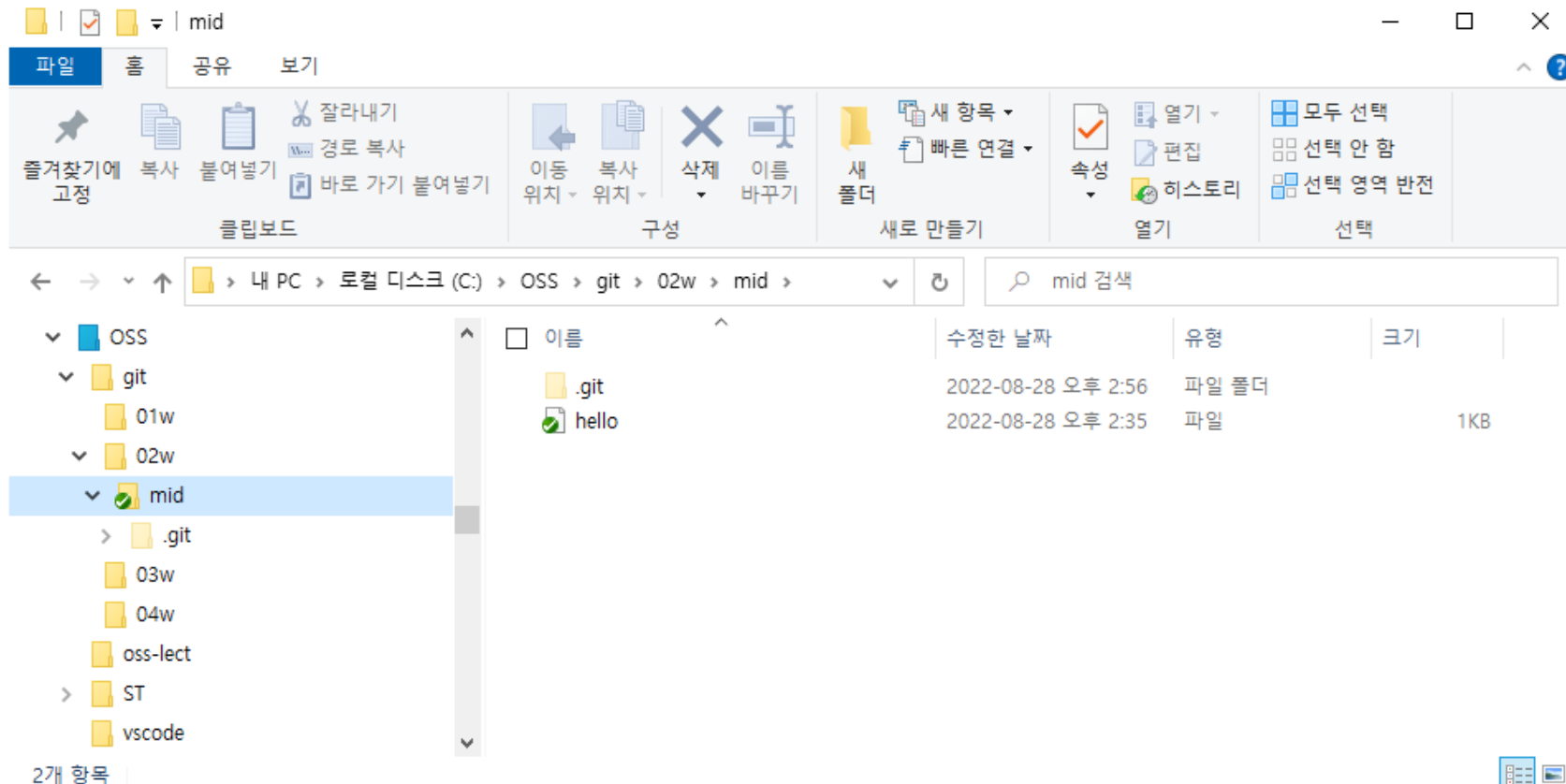
```
PC@DESKTOP-482NOAB MINGW64 /c/oss/git/02w/basic (main)
$ cat .git/config
[core]
    repositoryformatversion = 0
    filemode = false
    bare = false
    logallrefupdates = true
    symlinks = false
    ignorecase = true
[user]
    name = hskang
    email = hskang
```

- 사용자 설정 현재 저장소
  - 이 설정은 저장소 별로 저장되며 global 옵션으로 설정한 정보보다 우선적으로 사용
    - `$ git config user.name ai7dnn`
    - `$ git config user.email ai7dnn@gmail.com`
  - 사용자 설정 전체
    - `$ git config --global user.name "John Doe"`
    - `$ git config --global user.email johndoe@example.com`
- 설정 확인
  - `$ git config --list`

- # 전역 설정을 삭제
  - \$ git config --global --unset user.name
  - \$ git config --global --unset user.email
- # 개별 저장소의 설정을 삭제
  - \$ git config --unset user.name
  - \$ git config --unset user.email
- 개별을 삭제하더라도 전역에 설정한 것이 있다면 전역 설정 값으로 보임

## • 마스터 저장소

- Initialized empty Git repository in [경로] 이렇게 한 줄이 찍히고 .git이라는 숨겨진 폴더가 생성





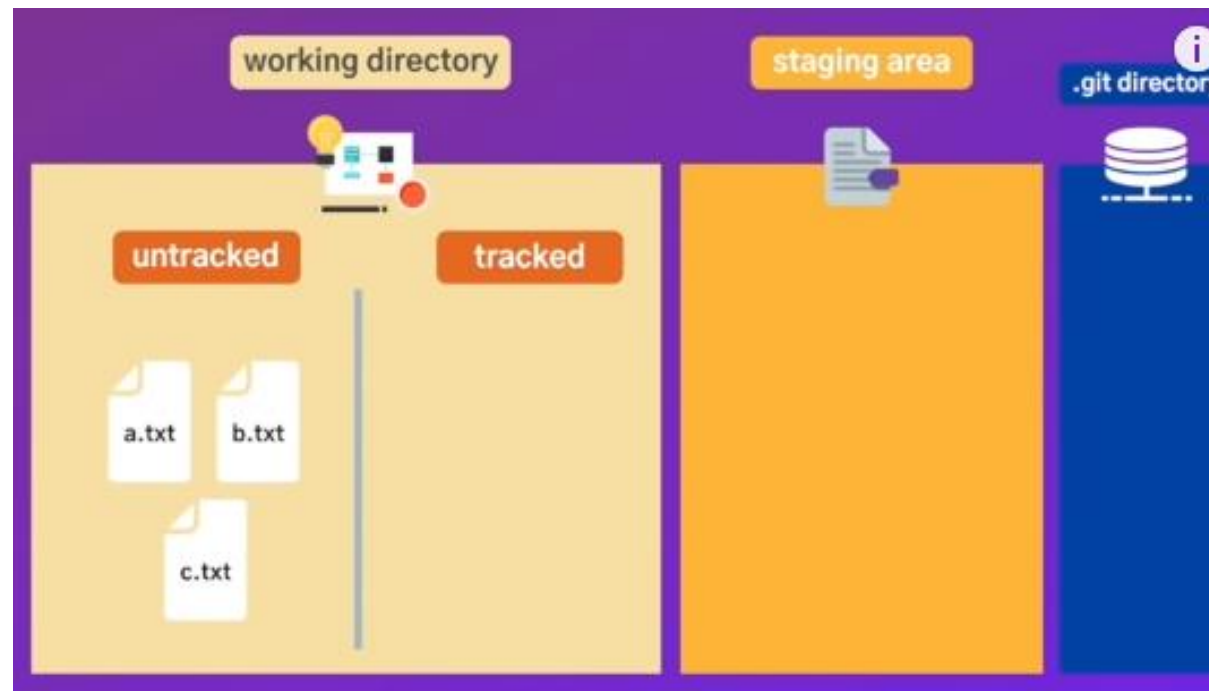
AI Experts  
Who Lead  
The Future

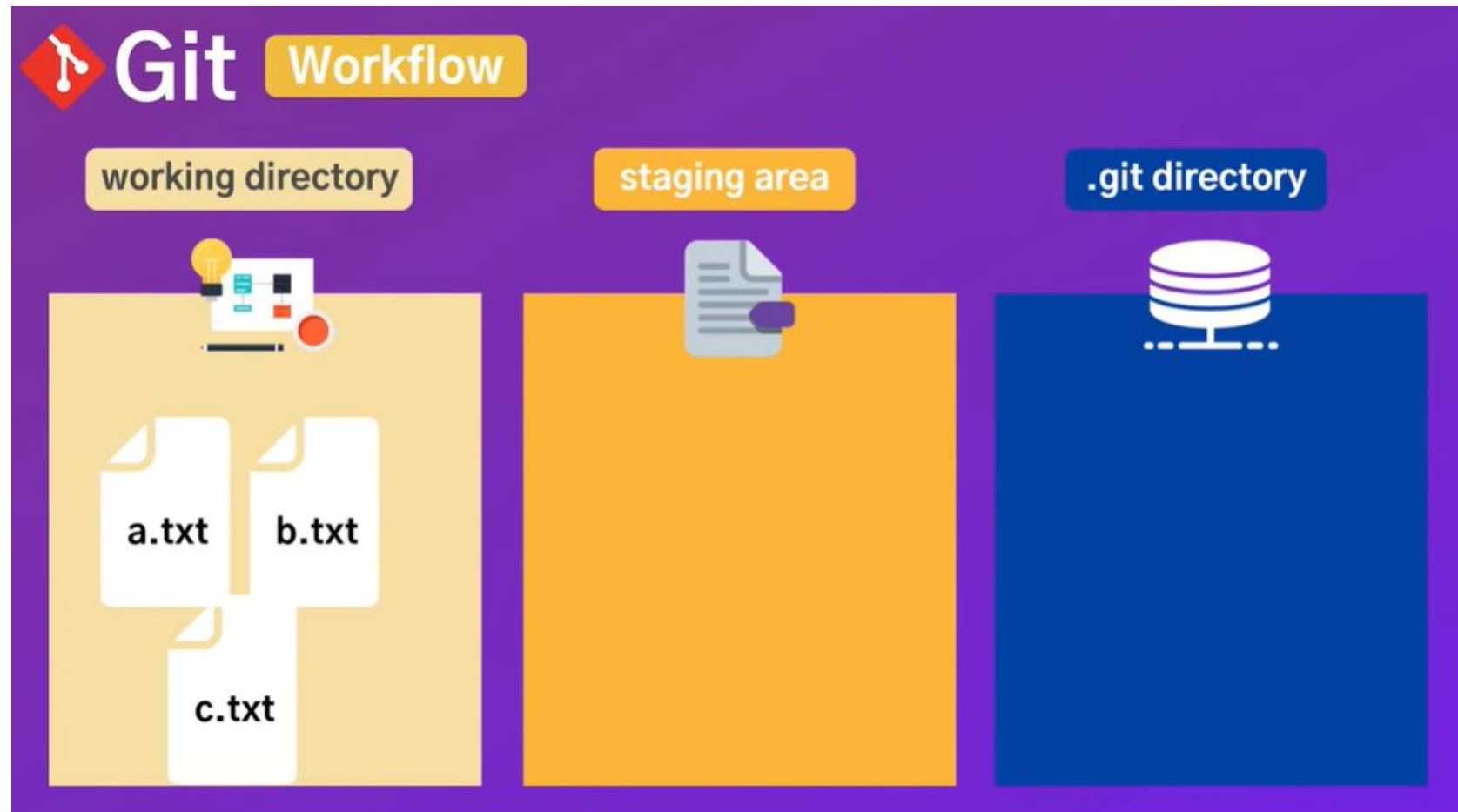
# 03

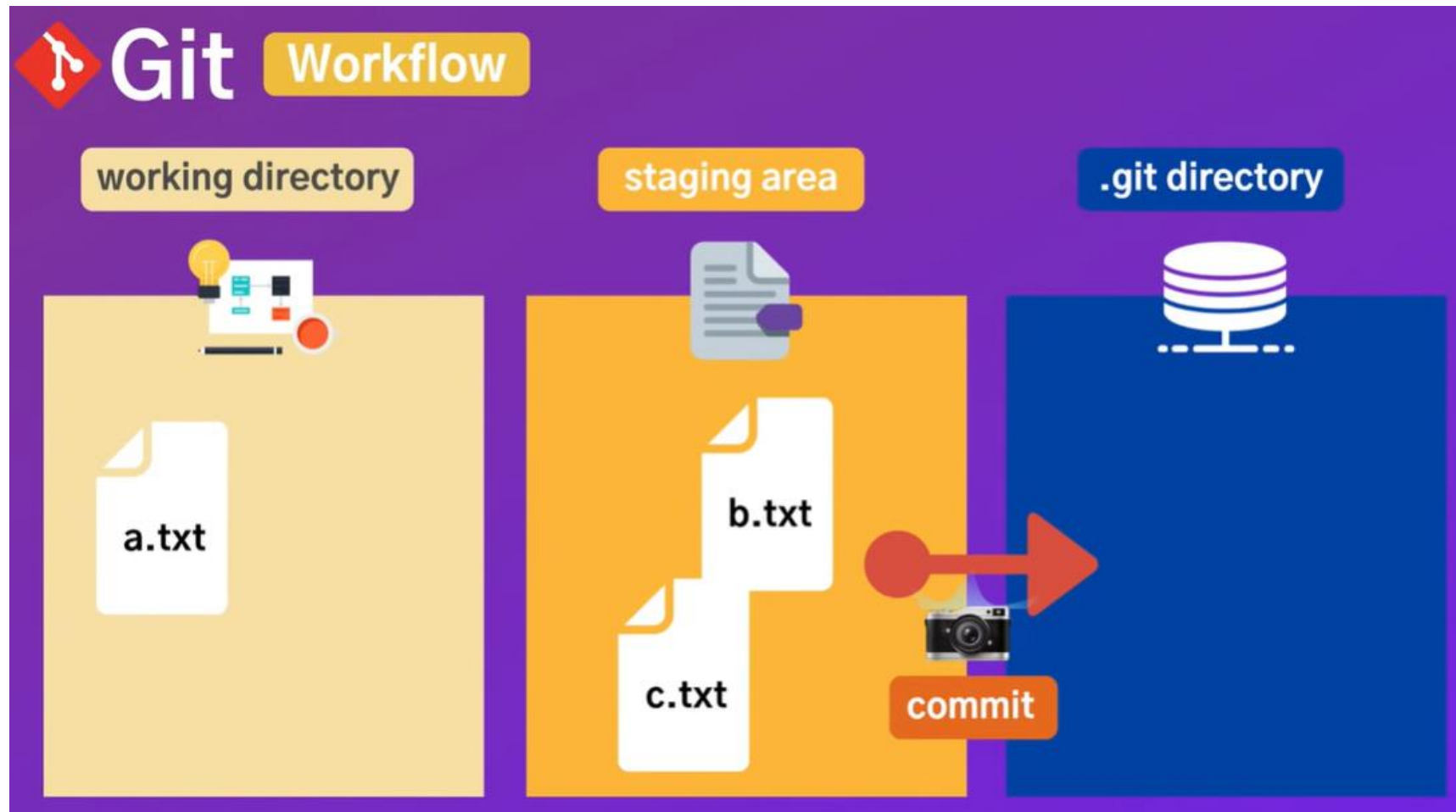
## 깃 커밋 명령어

- `$ git config --global user.name "[name]"`
  - 자신이 생성한 커밋(commit)에 들어갈 이름을 설정합니다
- `$ git config --global user.email "[email address]"`
  - 자신이 생성한 커밋에 들어갈 이메일 주소를 설정합니다
- `$ git config --list`
- `$ git config --global -e`
  - 현재 전역 설정을 (현재 설정된 편집기로 )파일을 열어 편집할 수 있음
- `$ git config -e`
  - 현재 저장소 설정을 (현재 설정된 편집기로 )파일을 열어 편집할 수 있음
- `$ git config --global core.editor "notepad"`
  - 자동 편집기 설정
- `$ git config --global init.defaultBranch main`
  - 기본 브랜치 이름설정

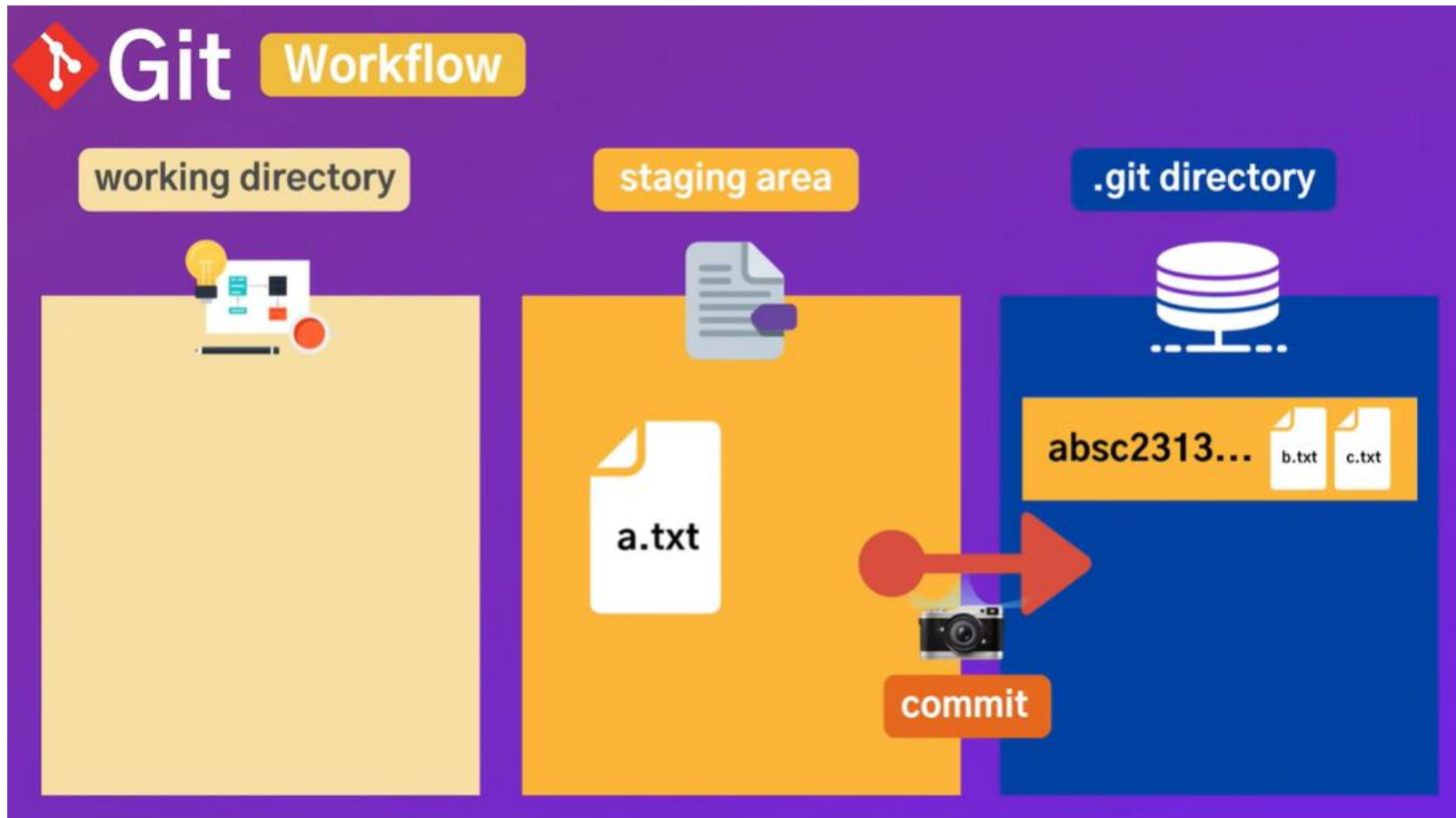
- **Untracked vs tracked**
  - 깃의 관리 대상 인지 아닌지
    - 형상(버전) 관리의 대상인지 아닌지
  - **Untracked**
    - 처음 생성만 해 놓은 파일
    - **Tracked로 이동**
      - `$ git add [file]`





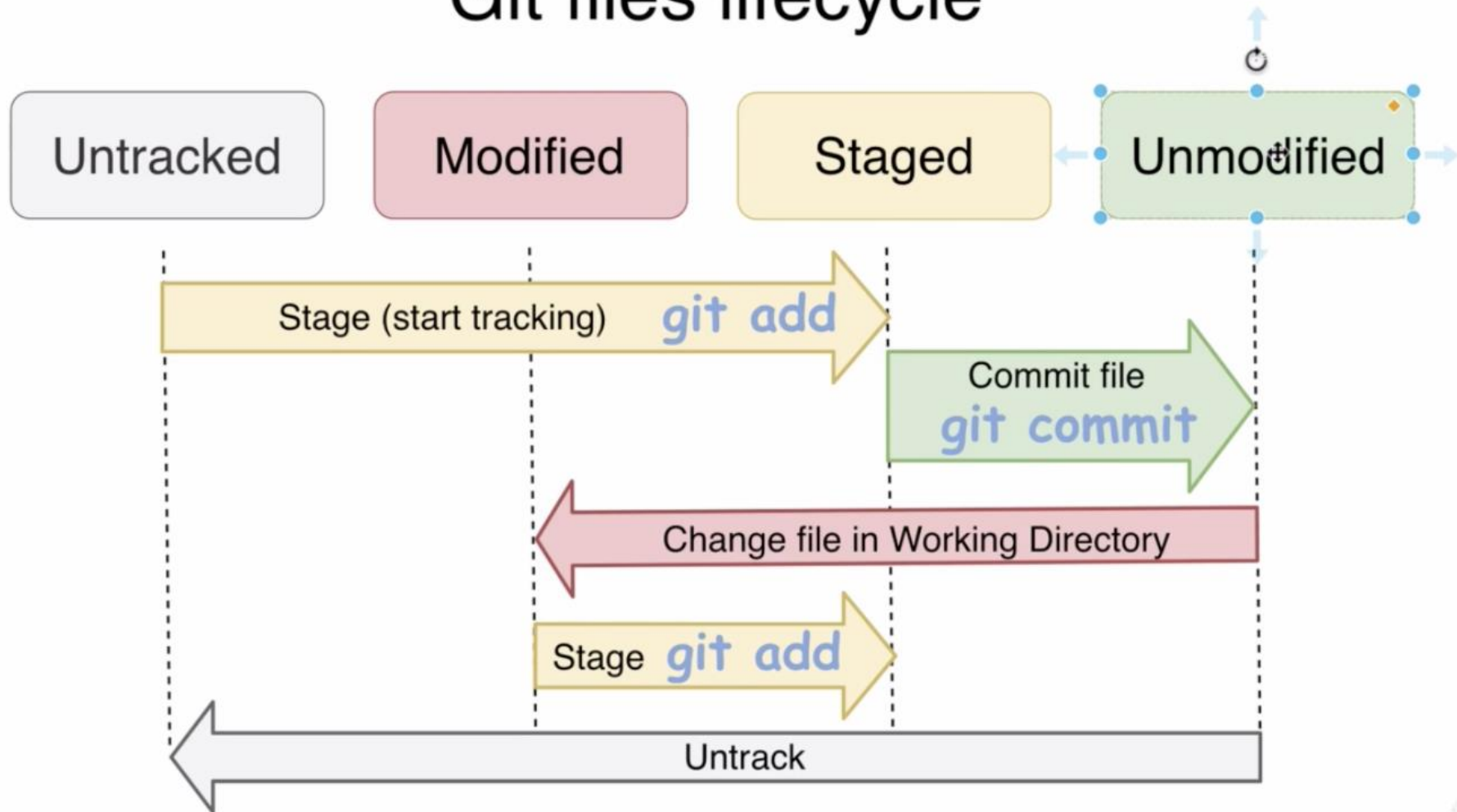


- 버전 관리를 위한 git 디렉토리에 반영

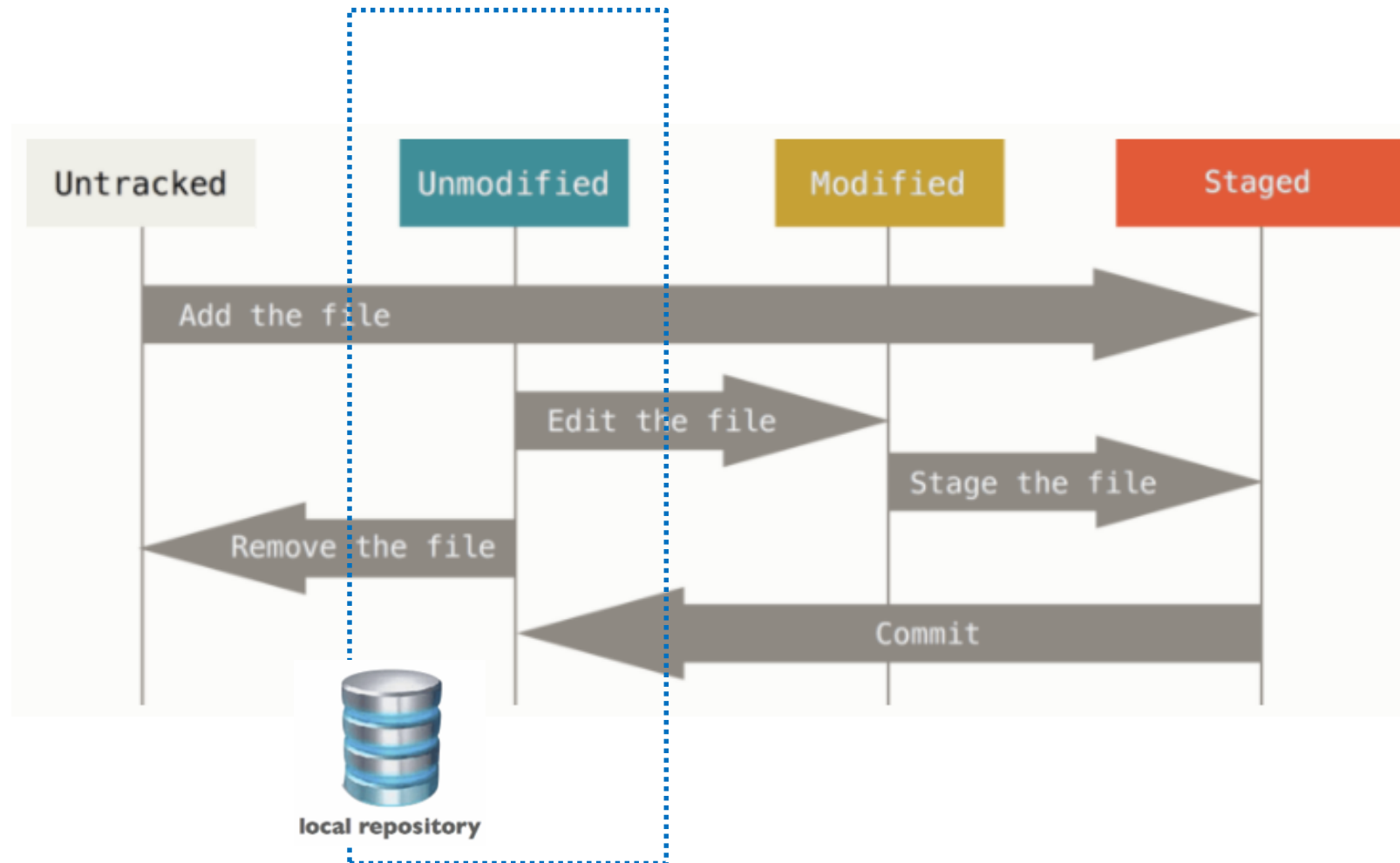




## Git files lifecycle







- hello 파일 생성 후 상태 확인
  - \$ echo 'A' > hello
  - \$ git status
    - On branch master
    - Your branch is up-to-date with 'origin/master'.
    - Untracked files:
    - (use "git add <file>..." to include in what will be committed)
  - hello
  - nothing added to commit but untracked files present (use "git add" to track)
- Git은 Untracked 파일
  - 아직 스냅샷(커밋)에 넣어지지 않은 파일
  - 파일이 Tracked 상태가 되기 전까지는 파일 커밋 불가능

- hello 파일을 추가해서 직접 Tracked 상태
  - \$ git add hello
  - \$ git status
- Tracked 상태이면서 커밋에 추가될 Staged 상태라는 것을 확인
- `Changes to be committed`: '커밋할 변경 사항' 이 준비되었다 라는 것
  - Staged 상태라는 것을 의미

깃은 3 가지 플로우 상태를 이동

- 작업 디렉토리
- 스테이징 영역(index)
- 저장소(깃 디렉토리)

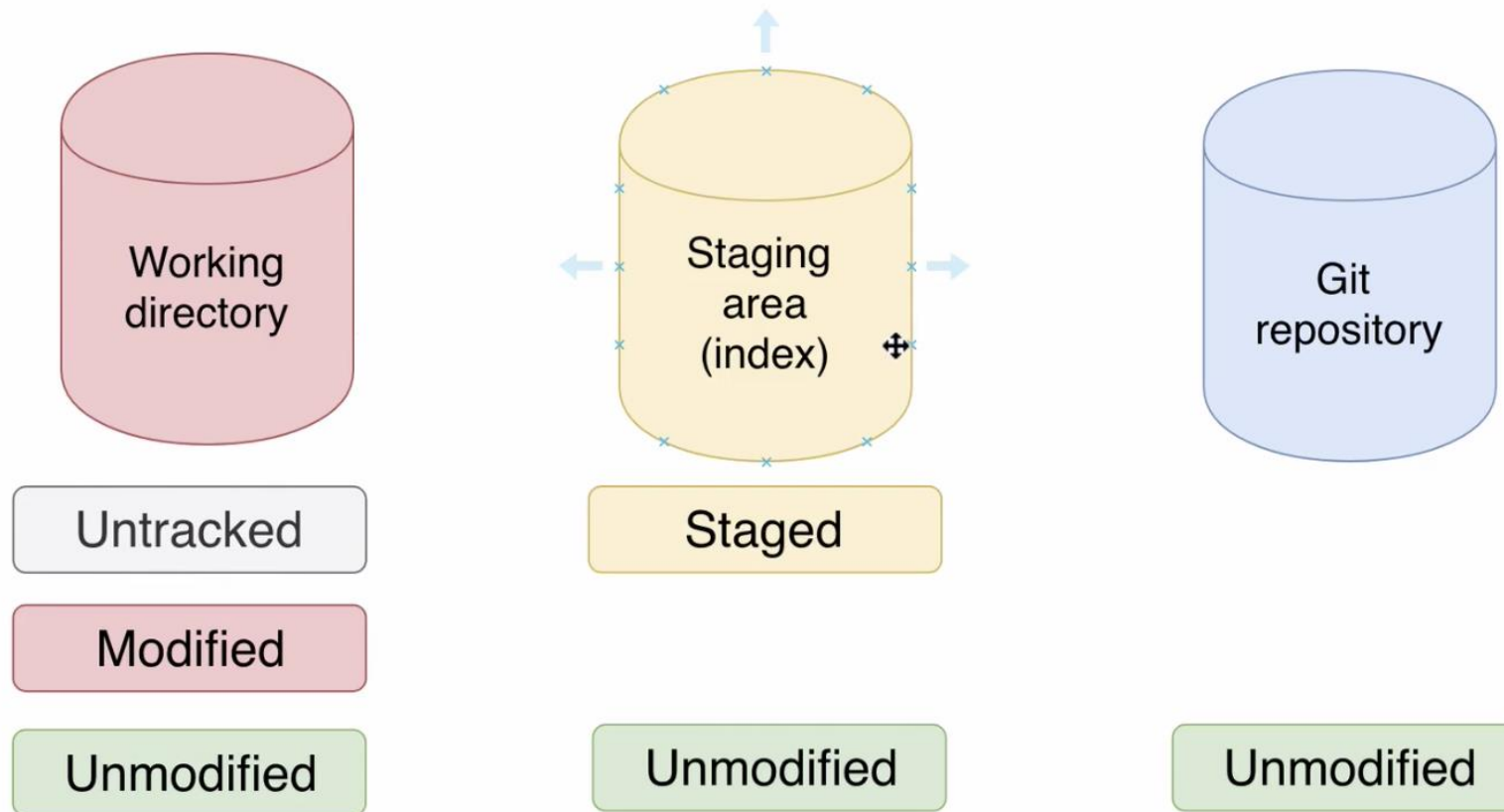
깃 명령어

```
git add
git status
git commit
```

- **commit history 확인**
  - Author 영역의 이름과 이메일 주소
  - git config 명령을 통해 세팅했던 user.name / user.email 값이 표기

- **Tracked(관리 반복대상)와 Untracked(관리대상이 아님)로 나뉨**
  - 워킹 디렉토리의 모든 파일
  - Tracked 파일(Git이 알고 있는 파일)은 다음 중 하나
    - **Unmodified(수정하지 않음)**
    - **Modified(수정함)**
    - **Staged(커밋으로 저장소에 기록할) 상태 중 하나**
  - 나머지 파일은 모두 **Untracked 파일**
    - 워킹 디렉토리에 있는 파일 중 스냅샷(저장소)에도 Staging Area에도 포함되지 않은 파일
    - 처음 저장소를 생성하면 모든 파일은 **untracked**
    - 처음 저장소를 Clone 하면 모든 파일은 **Tracked**이면서 **Unmodified** 상태
- **이미 마지막 커밋을 한 이후**
  - 아직 아무것도 수정하지 않은 상태에서 어떤 파일을 수정하면
    - **Git은 그 파일을 Modified** 상태로 인식
  - 실제로 커밋을 하기 위해서는
    - **이 수정한 파일을 Staged** 상태로 만들고
      - `$ git add [file]`
    - **Staged 상태의 파일을 커밋**
      - `$ git commit -m "message"`

## File states in Git areas



## A TINY CHEATSHEET ON

# GIT



git is the most popular version control system. Being proficient in git is a must for every developer.

### git init

Create an empty repository in the current directory

### git add [file]

Add the [file](s) to the *staging area*

### git commit

Create a snapshot of all the files in the *staging area*

### git push [name] [b]

Push changes to a remote repository called [name] to branch [b]

### git pull [name] [b]

Pull any changes from a remote repository called [name] from branch [b]

### git branch

List all of the branches in your repo. Add a [branch] argument to create a new branch called [branch].

### git remote add

[name] [url]  
Add a remote repository with [url] and an alias of [name]

### git checkout -b [branch]

Switch to a branch named [branch]

### git merge [branch]

Merge branch named [branch] with the current one

### git clone [url]

Download a git repository from [url]

### git log

List all the commits in the current branch's history

## Create

### From existing data

```
cd ~/projects/myproject
git init
git add .
```

### From existing repo

```
git clone ~/existing/repo ~/new/repo
git clone git://host.org/project.git
git clone ssh://you@host.org/proj.git
```

## Show

### Files changed in working directory

```
git status
```

### Changes to tracked files

```
git diff
```

### What changed between \$ID1 and \$ID2

```
git diff $id1 $id2
```

### History of changes

```
git log
```

### History of changes for file with diffs

```
git log -p $file $dir/ec/tory/
```

### Who changed what and when in a file

```
git blame $file
```

### A commit identified by \$ID

```
git show $id
```

### A specific file from a specific \$ID

```
git show $id:$file
```

### All local branches

```
git branch
```

(star '\*' marks the current branch)

AI Experts  
Who Lead  
The Future

## 03

### 깃 저장소 간략한 상태 정보와 취소



- 옵션 **--short [file]**
  - 파일 상태를 짧막하게 확인하기
  - File 있으면 file에 대한 것만 표시, 없으면 모든 파일 표시
- **??**
  - 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

- 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 하려면
    - `$ git restore --staged fname`