02_Git 기초

비고

실습

[1] Git 초기 설정

> 최초 한 번만 설정합니다. 매번 Git을 사용할 때마다 설정할 필요가 없습니다.

1. 누가 커밋 기록을 남겼는지 확인할 수 있도록 이름과 이메일을 설정합니다.

   작성자를 수정하고 싶을 때에는 이름, 메일 주소만 다르게 하여 동일하게 입력합니다.

```
$ git config --global user.name "이름"
$ git config --global user.email "메일 주소"
```

2. 작성자가 올바르게 설정되었는지 확인 가능합니다.

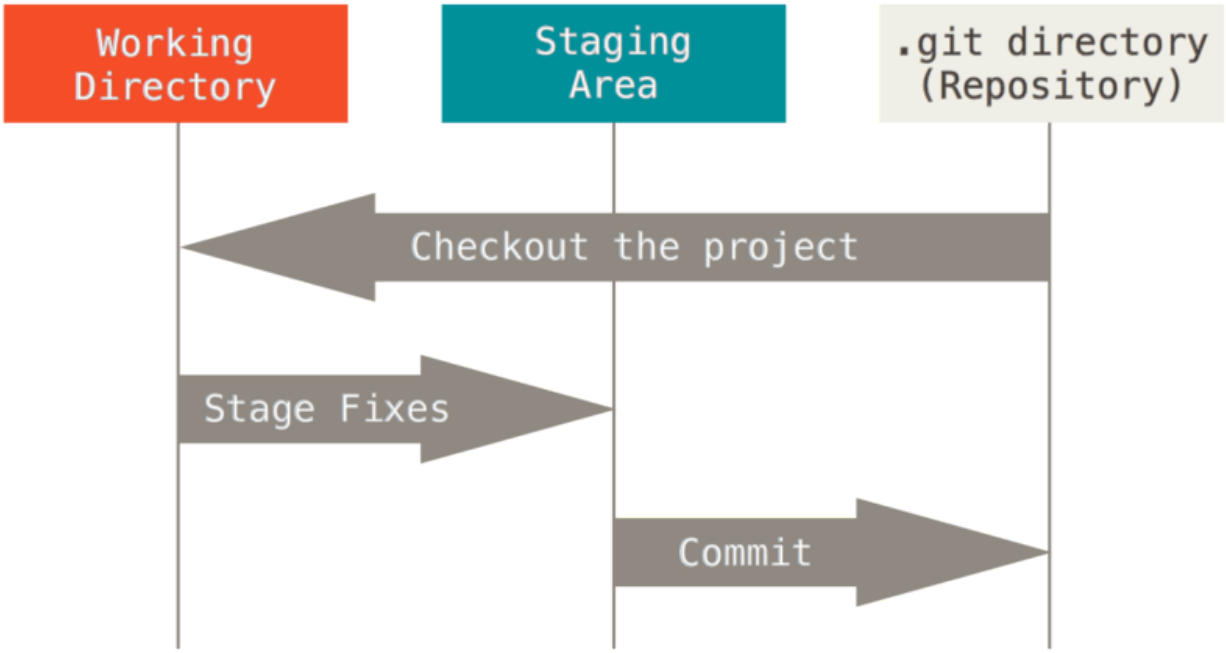
```
$ git config --global -l

또는

$ git config --global --list
```

[2] Git 기본 명령어

(0) 로컬 저장소



- `Working Directory (= Working Tree)` : 사용자의 일반적인 작업이 일어나는 곳
- `Staging Area (= Index)` : 커밋을 위한 파일 및 폴더가 추가되는 곳
- `Repository` : staging area에 있던 파일 및 폴더의 변경사항(커밋)을 저장하는 곳
- Git은 **Working Directory** → **Staging Area** → **Repository** 의 과정으로 버전 관리를 수행합니다.

(1) git init

```
$ git init
Initialized empty Git repository in C:/Users/kyle/git-practice/.git/

kyle@KYLE MINGW64 ~/git-practice (master)
```

- 현재 작업 중인 디렉토리를 Git으로 관리한다는 명령어

- `.git` 이라는 숨김 폴더를 생성하고, 터미널에는 `(master)`라고 표기됩니다.
- Mac OS의 경우 `(master)`가 표기되지 않는데, Terminal 업그레이드를 통해 표기할 수 있습니다.

> ❗ **주의 사항**
>
> 1. 이미 Git 저장소인 폴더 내에 또 다른 Git 저장소를 만들지 않습니다. (중첩 금지)
> 즉, 터미널에 이미 (master)가 있다면, git init을 절대 입력하면 안됩니다.
>
> 2. 절대로 홈 디렉토리에서 git init을 하지 않습니다. 터미널의 경로가 `~` 인지 확인합니다.

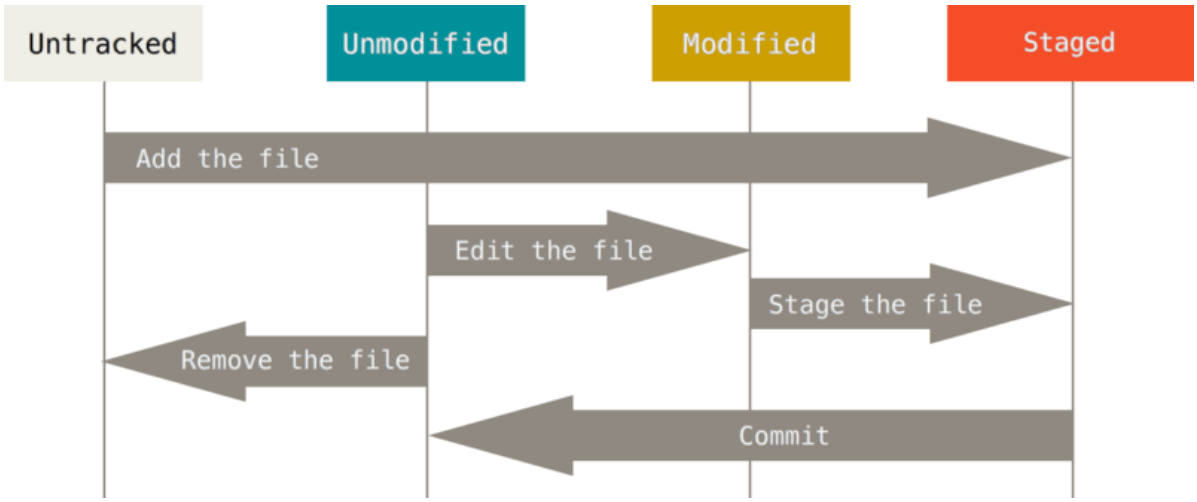
(2) git status

```
$ git status
On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)
```

- Working Directory와 Staging Area에 있는 파일의 현재 상태를 알려주는 명령어
- 어떤 작업을 시행하기 전에 수시로 status를 확인하면 좋습니다.
- 상태
    1. `Untracked` : Git이 관리하지 않는 파일 (한번도 Staging Area에 올라간 적 없는 파일)
    2. `Tracked` : Git이 관리하는 파일
        a. `Unmodified` : 최신 상태
        b. `Modified` : 수정되었지만 아직 Staging Area에는 반영하지 않은 상태
        c. `Staged` : Staging Area에 올라간 상태



파일의 라이프사이클

(3) git add

```
# 특정 파일
$ git add a.txt

# 특정 폴더
$ git add my_folder/

# 현재 디렉토리에 속한 파일/폴더 전부
$ git add .
```

- Working Directory에 있는 파일을 Staging Area로 올리는 명령어
- Git이 해당 파일을 추적(관리)할 수 있도록 만듭니다.
- `Untracked, Modified → Staged` 로 상태를 변경합니다.
- 예시

```
$ touch a.txt b.txt

$ git status
On branch master

No commits yet

Untracked files: # 트래킹 되고 있지 않는 파일 목록
  (use "git add <file>..." to include in what will be committed)
```

```
        a.txt
        b.txt

nothing added to commit but untracked files present (use "git add" to track)
```

```
# a.txt만 Staging Area에 올립니다.

$ git add a.txt
```

```
$ git status

On branch master

No commits yet

Changes to be committed: # 커밋 예정인 변경사항(Staging Area)
  (use "git rm --cached <file>..." to unstage)
        new file:   a.txt

Untracked files: # 트래킹 되고 있지 않은 파일
  (use "git add <file>..." to include in what will be committed)
        b.txt
```

(4) git commit

```
$ git commit -m "first commit"
[master (root-commit) c02659f] first commit
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 a.txt
```

- Staging Area에 올라온 파일의 변경 사항을 하나의 버전(커밋)으로 저장하는 명령어
- `커밋 메세지`는 현재 변경 사항들을 잘 나타낼 수 있도록 `의미` 있게 작성하는 것을 권장합니다.
- 각각의 커밋은 `SHA-1` 알고리즘에 의해 반환 된 고유의 해시 값을 ID로 가집니다.
- `(root-commit)` 은 해당 커밋이 최초의 커밋 일 때만 표시됩니다. 이후 커밋부터는 사라집니다.

(5) git log

```
$ git log
commit 1870222981b4731d14ef91d401c68c0bbb2f6e7d (HEAD -> master)
Author: kyle <kyle123@hphk.kr>
Date:   Thu Dec 9 15:26:46 2021 +0900

    first commit
```

- 커밋의 내역(`ID, 작성자, 시간, 메세지 등`)을 조회할 수 있는 명령어
- 옵션
  - `--oneline` : 한 줄로 축약해서 보여줍니다.
  - `--graph` : 브랜치와 머지 내역을 그래프로 보여줍니다.
  - `--all` : 현재 브랜치를 포함한 모든 브랜치의 내역을 보여줍니다.
  - `--reverse` : 커밋 내역의 순서를 반대로 보여줍니다. (최신이 가장 아래)
  - `-p` : 파일의 변경 내용도 같이 보여줍니다.
  - `-2` : 원하는 갯수 만큼의 내역을 보여줍니다. (2 말고 임의의 숫자 사용 가능)

💡 **옵션과 인자**

명령어를 사용하면서 `-` 혹은 `--`를 통해 옵션을 사용하는 것을 배웠습니다.
옵션과 더불어서 인자라는 개념도 존재하는데요. 옵션과 인자 무엇이 다를까요?

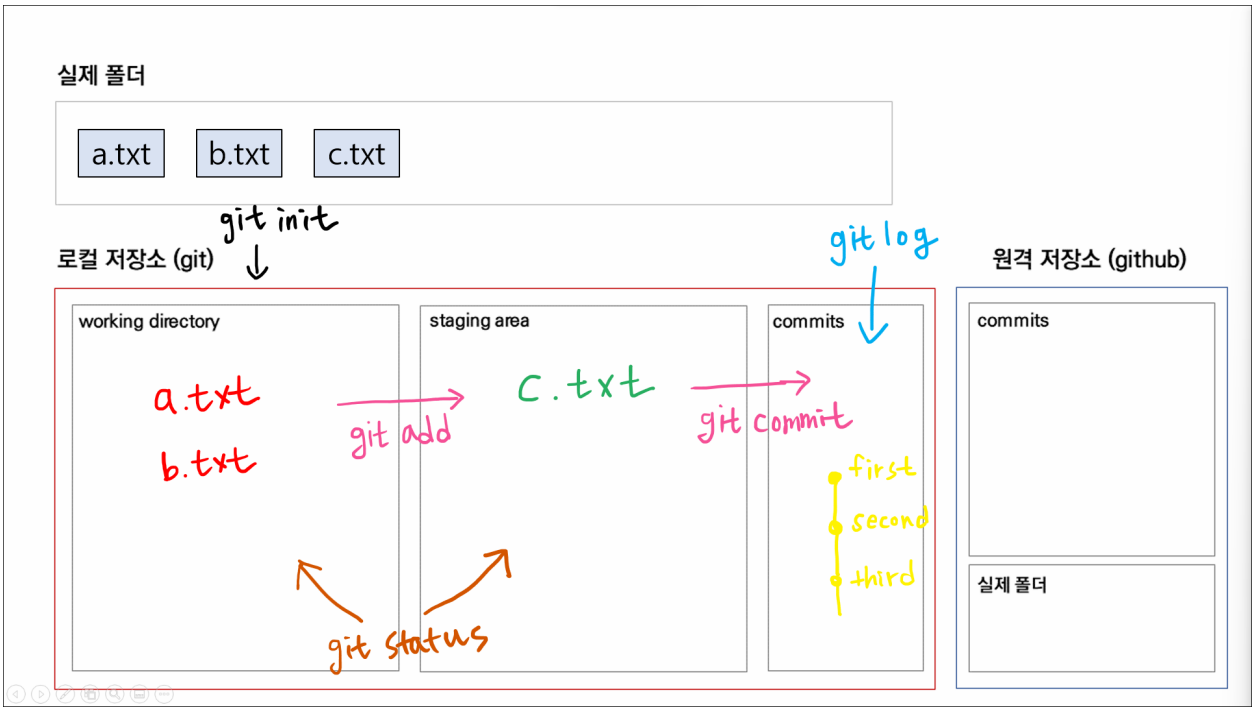
—-

`옵션`은 명령어의 `동작 방식`을 지정하는 것입니다. 따라서 **생략 가능**합니다.
단순히 기존 기능보다 부가 적인 기능을 원할 때 사용합니다.
예를 들면 `git log --oneline`은 커밋 내역을 한 줄로 보고 싶을 때 사용합니다.
`oneline` 옵션은 말 그대로 부가 적인 기능이므로, 생략해도 `git log`는 정상 동작 합니다.

—-

`인자`는 명령어의 `동작 대상`을 지정하는 것입니다. 따라서 **생략이 불가능** 합니다.
예를 들면 `git add` 라고만 작성하면 어떤 파일을 Staging Area에 올릴지 모르게 됩니다.
반드시 `git add a.txt` 와 같이 git add 명령어가 동작할 대상을 지정해야 하는데
이때 `a.txt`와 같은 대상을 인자라고 합니다.

(6) 한눈에 보는 Git 명령어



- staging area 참고

  https://coderefinery.github.io/git-intro/04-staging-area/