1.
$ git --version

2.
$ git config --global user.email <설정할때는 이메일작성>
$ git config --global user.name <설정할때는 이름작성>

3.
$ mkdir gitPractice

4.
$ ls -l
$ ls -al gitPractice/

5.
$ cd gitPractice/

6.
$ git init
Class01@DESKTOP-0CO5278 MINGW64 /c/Class250615/gitPractice (main)

✅ 저장소 초기화 후에는 초기 브랜치 이름이 보입니다. 최근에는 **main** 이름 사용합니다.
초기 브랜치 이름을 변경하고 싶으면 **git branch -m main** 실행합니다.

7. git 저장소 만든 후 **.git** 폴더 생성 확인
$ ls -al

8. echo 명령
$ echo "version 1 coding..." ✅ 콘솔에 문자열 출력

$ echo "version 1 coding..." > hello.txt
✅ 출력스트림을 지정된 파일로 지정(파일이 존재하면 덮어쓰기)

$ echo "version 1 coding..." >> hello.txt ✅ **append** (기존 내용 뒤에 추가)

9. 파일의 내용 확인
$ cat hello.txt

10. git이 추적하는 파일의 상태 확인
$ git status
On branch main

No commits yet

Untracked files: 🔺 추적하지 않는 파일 목록
  (use "git add <file>..." to include in what will be committed)

```
        hello.txt
```

11.

$ git add hello.txt

12.

$ git status

```
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   hello.txt
```

🔺 스테이징한 새로운 파일목록

13. 다시 언스테이징

$ git rm --cached hello.txt 또는

$ git reset hello.txt

14.

$ git commit -m "start hello"

$ git log

```
commit 2600cb60efa74238b3d1361fad5d2598b1617d95 (HEAD -> main)
Author: kimsohee <koreait.sec2020@gmail.com>
Date:   Fri Jul 25 10:13:01 2025 +0900

    start hello
```

✅ 2600cb60efa74238b3d1361fad5d2598b1617d95는 커밋의 해시 16진수 값
HEAD 는 마지막 커밋 위치를 가리키고 있습니다. **HEAD** 는 현재 작업중인 위치 포인터

$ git log --oneline

2600cb6 (HEAD -> main) start hello

15.브랜치 생성과 브랜치 이동**(1)**

$ git checkout -b feature/login

Switched to a new branch 'feature/login'

Class01@DESKTOP-0CO5278 MINGW64 /c/Class250615/gitPractice (feature/login)

✅ 브랜치 이동 확인 : 현재 브랜치 **feature/login**

16. 병합 실습을 위해 새로운 내용 추가

$ echo "login coding...." >> hello.txt

17. 기존 변경사항**(16번)** 을 다시 스테이징하기

$ git add hello.txt

18.

```
$ git commit -m "feature login"
$ git log --oneline
```

19.

```
$ git checkout main
```

20.브랜치 생성과 브랜치 이동(2)

```
$ git switch -c feature/logout
```

21.

```
$ echo "logout coding..." >> hello.txt
```

22.

```
$ git commit -m "feature logout"
```

23. 다른 브랜치로 이동하는 명령어들

```
$ git checkout main
$ git switch feature/logout
```

24. 병합하기 (1) - 충돌없음 (fast-forward 병합)

```
$ git merge feature/login
```

병합하기(2) - 충돌 예상

```
$ git merge feature/logout
```

충돌 발생하면 정상적인 병합 순서

1) 편집기에서 소스코드의 '원하는 병합 결과'로 수정한다.
2) 스테이징
3) 커밋

충돌 발생할 때 취소

1) 스테이징 전 : **git merge --abort**
2) 스테이징 후 : **git reset --merge**

병합 커밋 후 취소

```
$ git revert -m 1 2939400
```

🧪 **Detached(분리된)** 체크아웃

- 특정 커밋을 직접 체크아웃하여 **HEAD**가 브랜치가 아닌 커밋 자체를 가리키는 상태를 말합니다. 이 상태에서 작업하면 브랜치에 연결되지 않아서 커밋이 사라질 수 있으므로 파일을 수정하지 않도록 합니다. 읽기 전용.

브랜치 이동 명령어

- git checkout 은 초기 버전 부터 사용
- git switch 는 2.23(2019년) 버전 부터 사용. git checkout 의 기능 중 브랜치 관련 명령어만 분리함.

```
$ git checkout feature/login

$ git switch feature/login
```

새 브랜치 생성 후 이동

```
$ git checkout -b feature/login

$ git switch -c feature/login
```

특정 커밋으로 이동 (Detached HEAD)

```
$ git checkout abc1234<커밋의 해시값>

$ git switch --detach abc1234
```

바로 이전 브랜치로 이동

```
$ git checkout -

$ git switch -
```

스테이징

```
$ git add  <파일명1>, <파일명2>
$ git add *.java
$ git add *
$ git add .
```

처음 생성된 파일은 **git add** 로 스테이징

한 번 스테이징(커밋) 했던 파일에 변경 사항이 발생하면 다음 커밋에 대상이 되는지를 **git add** 명령으로 지정.(두번째 스테이징 부터는 **commit** 옵션 **-a** 로 변경된 모든 파일 스테이징을 할 수 있음.)
예시 : 프로젝트 폴더안에 파일이 **10**개 있습니다.
파일이 만들어지는 순서대로 **git** 이 이 파일을 관리를 하도록 할것인가?
yes -> 스테이징 **(git** 이 파일의 상태를 추적할 수 있도록 합니다.**)**
no -> 스테이징 안함.

스테이징한 파일들이 커밋의 대상이 됩니다.

커밋이란? **commit** 은 파일들의 변경점.변경 사항들을 저장.
(예시**)**
- 커밋**1 :** 로그인 기능과 관련된 소스파일 **html** 만 스테이징
- 커밋**2 :** 로그인 기능과 관련된 **java** 만 스테이징