

Git을 활용한 프로젝트 관리를 위한 몇 가지 기본 설정을 해보자.

: gitbash를 연다.

git 명령어 구조

👉 git 명령어 [옵션]

단축 옵션 (Short Options) → -한 개

- ✓ 한 글자로 된 옵션은 - 한 개를 사용한다.
- ✓ 여러 개의 한 글자 옵션을 **연결해서** 사용할 수도 있다.

ex) git commit -m "메시지"

긴 옵션 (Long Options) → -- 두 개

- ✓ 두개 이상의 글자로 된 옵션은 -- 두 개를 사용한다.
- ✓ 더 직관적이며, 명확하게 옵션을 전달할 수 있다.

ex) git log --all

1) git전역으로 사용 할 사용자이름과 이메일 주소 설정

👉 전역적 선언 명령어

git config --global user.name "이름설정"

git config --global user.email "이메일설정"

MINGW64:/c/Users/KOSTA

```
KOSTA@DESKTOP-7PLAKSP MINGW64 ~  
$ git config --global user.name "heejung"  
  
KOSTA@DESKTOP-7PLAKSP MINGW64 ~  
$ git config --global user.email "8253jang@daum.net"  
  
KOSTA@DESKTOP-7PLAKSP MINGW64 ~  
$ |
```

👉 git 설정정보 확인 명령어

git config --global user.name

git config --global user.email

MINGW64:/c/Users/KOSTA

```
KOSTA@DESKTOP-7PLAKSP MINGW64 ~  
$ git config --global user.name  
heejung  
  
KOSTA@DESKTOP-7PLAKSP MINGW64 ~  
$ git config --global user.email  
8253jang@daum.net  
  
KOSTA@DESKTOP-7PLAKSP MINGW64 ~  
$ |
```

👉 모든 설정 값 명령어

git config -list

조회 후 q 를 통해 빠져나가기

특정 repository 안에서 조회하면 특정 repository 에 대한 지역정보가
조회되고, git repository 밖에서 조회시 전역 정보 조회

2) 줄바꿈 문자열에 관련 설정

☞ 명령어

window인 경우 :

```
git config --global core.autocrlf true
```

mac인 경우:

```
git config --global core.autocrlf input
```

운영체제마다 에디터에서 줄바꿈을 하는 문자열이 들어가는것이 다르다.

윈도우는 `\r\n` (carriage-return + line feed) , mac는 `\n`(line feed)

이런 차이점때문에 다양한 운영체제에서 git을 사용하면 수정하지 않았음에도 줄바꿈 문자열이 달라져서 git히스토리를 보는데 어려움이 있다.

그래서,

window에서 **git config --global core.autocrlf true**를 설정하면 git에 올릴 때 carriage-return을 삭제하고 다시 git에서 window로 가져올 때는 자동으로 carriage-return을 추가해준다.

mac에서 **git config --global core.autocrlf input** 으로 설정하면 git에서 받아올 때는 별다른 수정이 일어나지 않지만 저장할 때는 carriage-return을 삭제해준다. 맥에서는 carriage-return이 들어가지 않지만 다른곳에서 복사해온 자료에는 carriage-return 들어가 있을수 있기때문에 설정이 필요하다.

3) 기본 브랜치이름 변경

☞ 명령어

```
git config --global init.defaultBranch 변경브랜치명
```

☞ 기본 브랜치이름 확인명령어

```
git config --get init.defaultBranch
```

기본브랜치명 master 로 되어 있는데, master 라는 용어가 인종차별적인 담고 있어서 이 용어들을 제거하기 위해 master 를 안 쓰는 추세이다.

.gitignore 파일

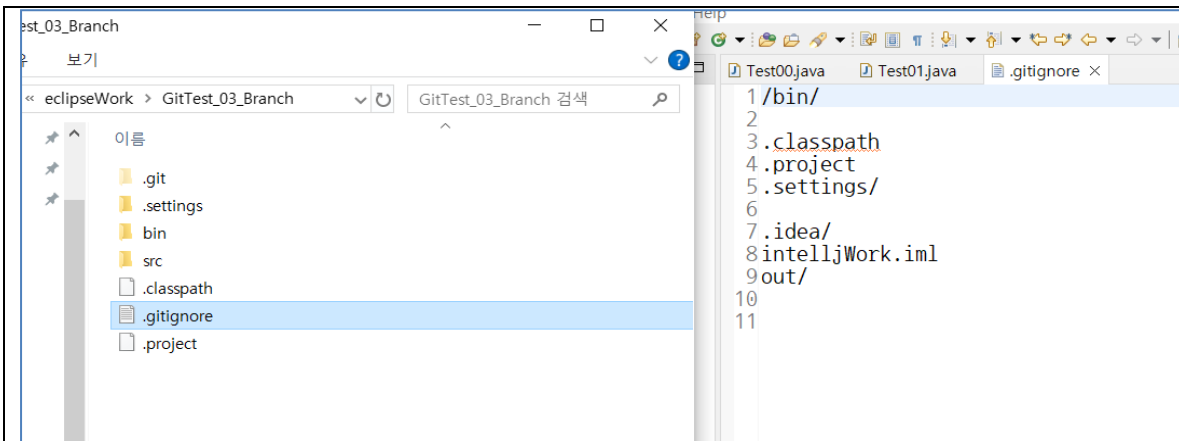
프로젝트에서 원하지 않는 백업 파일이나 로그파일 혹은 컴파일 된 파일들을 Git 에서 제외할 수 있는 설정 파일을 말한다.

- 보안상으로 위험성이 있는 파일
- 프로젝트와 관계없는 파일
- 용량이 너무 커서 제외해야되는 파일
- 편리성 제공

.gitignore 파일 규칙

표현	의미
#, 빈라인	#은 주석을 의미하며, 빈라인은 아무런 영향을 주지 않습니다.
*.a	확장자가 .a 인 모든 파일을 무시합니다.
folder_name/	해당 폴더의 모든 파일을 무시합니다.
folder_name/*.a	해당 폴더의 확장자가 .a 인 모든 파일을 무시합니다.
folder_name/*/*.a	해당 폴더 포함한 하위 모든 폴더에서 확장자가 .a 인 모든 파일을 무시합니다.
/*.a	현재 폴더의 확장자가 .a 인 모든파일을 무시합니다.

.gitignore파일을 열어서 제외하고 싶은 파일을 선언한다.



주의

Git 은 폴더 자체를 추적하지 않고 “파일”만 추적한다.
폴더안에 파일이 없으면 관리되지 않는다.

만약, 수업에서 관리되고 있는 Edu 폴더안에 소스들만 git 으로 관리하고 싶다면

```
# 기본적으로 전부 무시
/**
# Java 디렉터리는 열어준다
!/Java/

# Java/work 디렉터리는 열어준다
!/Java/work/

# work 하위 프로젝트 디렉터리는 열어준다
!/Java/work/*/

# 각 프로젝트의 src 디렉터리 허용
!/Java/work/*/src/
!/Java/work/*/src/**
```

Tip :gitignore 파일 자동 생성 해주는 사이트

gitignore.io

Git 용어 알아보기

1.Repository

Local Repository (로컬 컴퓨터의 .git 폴더),
RemoteRepository (서버의 저장소)

2. Origin

Remote Repository를 Origin 이라고 부름

3.Master or Main

Branch들 중 가장 중요하고 핵심인 원본
기존에는 master라는 이름을 사용하였지만 GitHub가 main으로 기본 이름을 바꾸면서 main으로 사용함.

4. Working Tree(workspace)

Staging Area 이전의 단계 실질적으로 Source를 작성하는 폴더.
Add 하기 이전의 공간

5. Index(Staging area)

Working Tree의 파일들이 Add 명령어를 통해 Staging area에 파일들을 올려야만 Commit이 가능함.

* Working Tree와 Index(Staging area)

작업중인 Source 폴더를 'Working Tree'라고 부르며 Commit을 실행하기 전의 저장소와 Working Tree 사이에 존재하는 공간을 'Index'라고 한다. Commit 작업은 Working Tree에 있는 변경 내용을 저장소에 바로 기록하는 것이 아니라 그 사이

Index에 파일 상태를 기록 (Staging 한다고 표현)하게 되어 있다. 저장소의 변경 사항을 기록하기 위해서는, 기록하고자 하는 모든 변경 사항들이 'Index'에 존재해야 한다. 하지만 Index라는 공간이 있기 때문에 작업 트리 안에 있는 Commit 이 필요 없는 파일 들을 Commit에 포함하지 않을 수 있고, 변경을 원하는 파일 들만 Commit 할 수 있다.

6. Commit

Staging Area의 파일들을 local repository에 버전으로 기록.
Push 하면 해당 버전이 서버로 올라감.

7. HEAD

Local Repository의 작업 버전과 현재 branch 위치.
일종의 커서 역할.

8.push

push를 실행하면, 원격 저장소에 내 변경 이력이 업로드 되어,
원격 저장소와 로컬 저장소가 동일한 상태가 된다.

9. Clone

Remote Repository를 Local Repository에 복사하는 명령어

10. Pull

Remote Repository의 최신 변경 사항을 Local Repository와 합치는 명령어.

11. Fetch

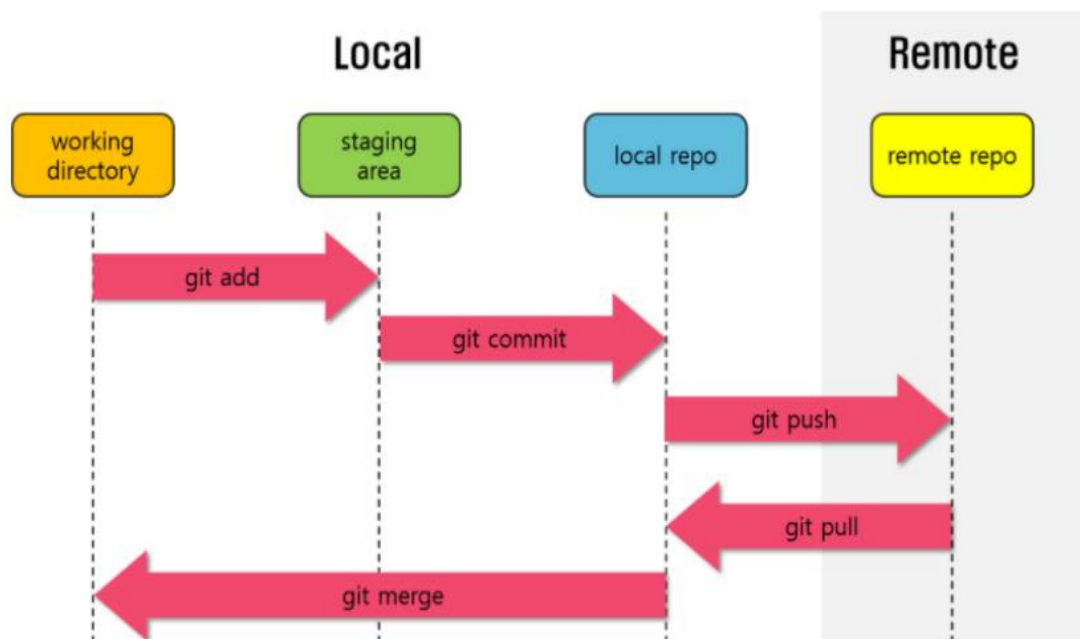
정보를 받아오고 Merge까지 해주는 Pull과 달리 Origin의 main Branch "정보만" 받아온다.

GitHub에 코드를 올리는 과정

- 1. 내 컴퓨터 프로젝트 폴더에 '여기에서 Git을 쓰겠다!' 라고 명령 `git init`
- 2. 즐겁게 코딩
- 3. 내가 변경한 파일 중 올리길 원하는 것만 선택 `git add`
- 4. 선택한 파일들을 한 덩어리로 만들고 설명 적어 주기 `git commit -m "첫페이지 제작"`
- 5. GitHub 사이트에서 프로젝트 저장소(Repository) 만들기
- 6. 내 컴퓨터 프로젝트 폴더에 GitHub 저장소 주소 알려주기 `git remote add`
- 7. 내 컴퓨터에 만들었던 덩어리 GitHub에 올리기 `git push`

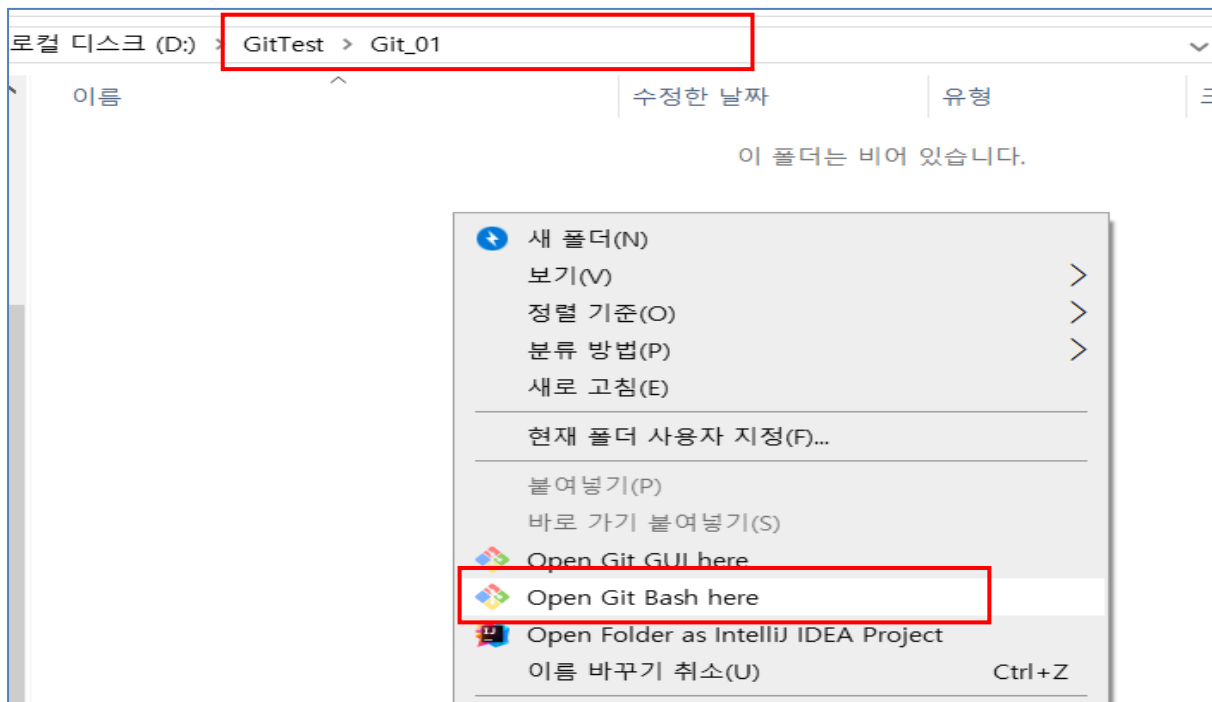


Git Workflow



- Git 테스트를 해보자.

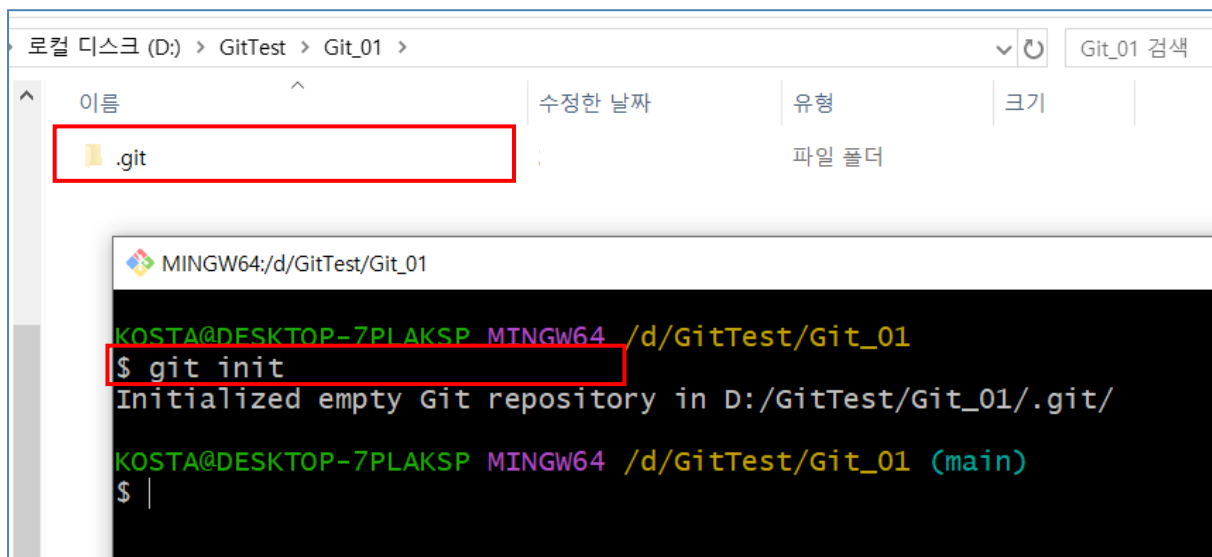
- 1) 먼저, git을 실습 할 폴더를 만든다.
- 2) 만든 폴더 안에서 오른쪽 버튼을 클릭하여 **Open Git Bash here** 를 클릭한다.



3) git을 초기화 한다.(숨김파일 보기설정)

☞ 명령어

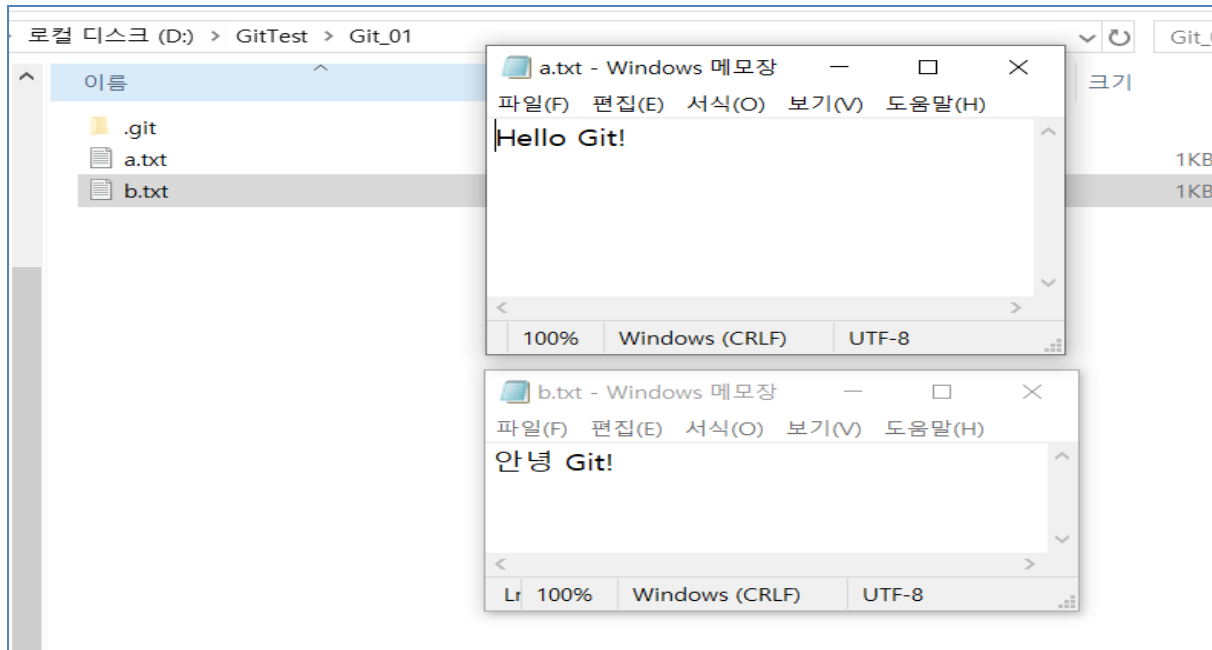
git init



- Git 초기화를 하면 .git 이라는 숨겨진 폴더가 로컬저장소이다.
- 로컬저장소에 내가 만든 버전 정보, 원격 저장소 주소 등이 저장된다.
- 원격저장소에서 내 컴퓨터로 코드를 받아오면 로컬저장소가 자동으로 생성된다.

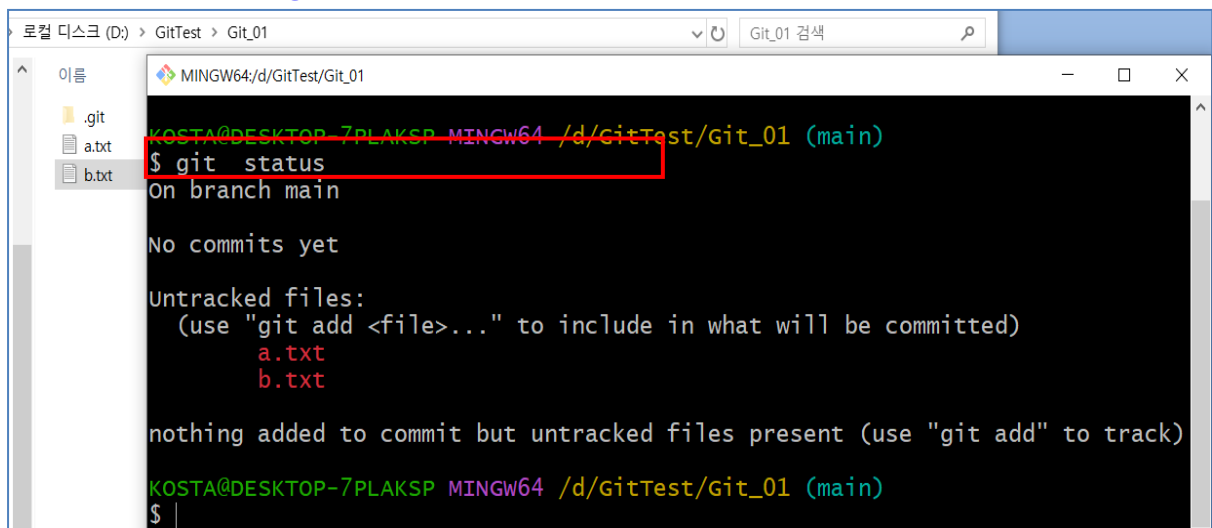
- 한 폴더에 하나의 로컬저장소만 유지해야 한다

4) a.txt , b.txt 파일을 작성하고 폴더에 저장한다.



5) git 상태 확인하기

👉 명령어 : **git status**

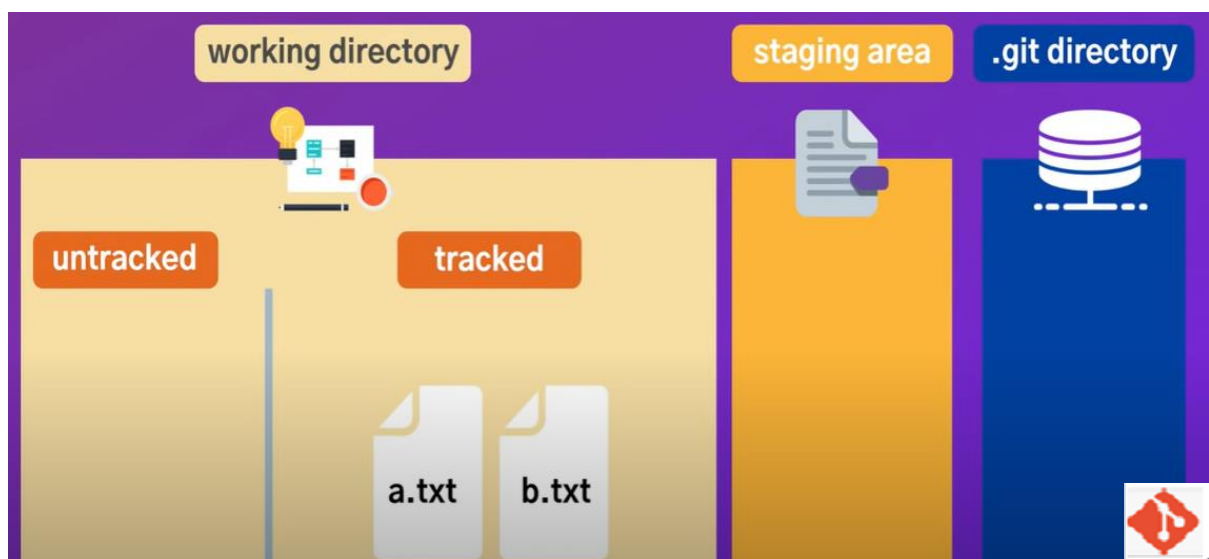


Git WorkFlow

- git status 상태



- Untracked 란 한번도 git이 추적한적이 없는 파일



- tracked 란 git이 한번이라도 추적한적이 있는 파일

6) 원하는 기능만 add해서 commit을 위한 준비하기

: add를 해서 commit 단계 이전의 공간 stage area에 올리기

☞ 명령어

git add .

```
KOSTA@DESKTOP-7PLAKSP MINGW64 /d/GitTest/Git_01 (main)
$ git add .
KOSTA@DESKTOP-7PLAKSP MINGW64 /d/GitTest/Git_01 (main)
$ |
```



git add . 은 모든변경(추가)사항을 staging area 이동
git add .

특정 파일만 add할 경우
git add 특정파일(위치포함)

git add 후 아래와 같은 경고 메시지가 나오는 경우는

“Git이 자동 줄바꿈 변환을 적용할 것이라는 알림” 이므로 무시해도 된다.

warning: in the working copy of 'a.txt', LF will be replaced by CRLF the next time Git touches it

☞ 현재 working directory, staging area의 상태 조회

git status

```
KOSTA@DESKTOP-7PLAKSP MINGW64 /d/GitTest/Git_01 (main)
$ git status
On branch main

No commits yet

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

KOSTA@DESKTOP-7PLAKSP MINGW64 /d/GitTest/Git_01 (main)
$
```

☞ local repository에 커밋이력 생성

git commit -m "메시지 타이틀"

git commit -m "메시지타이틀" -m "메시지내용"

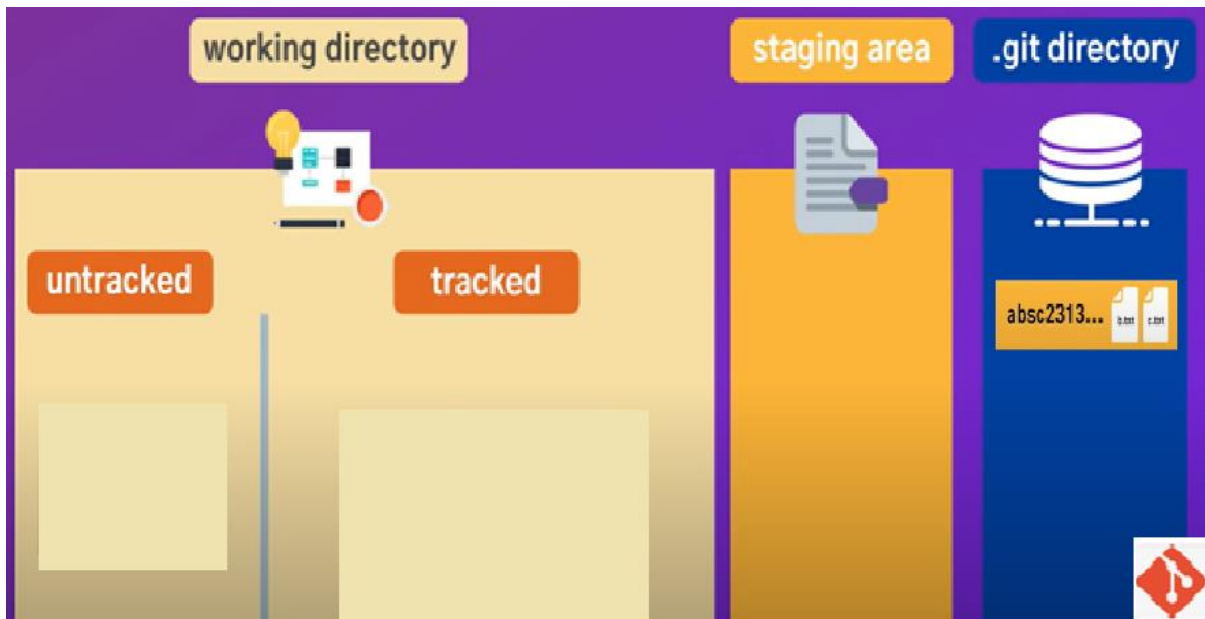
```
KOSTA@DESKTOP-7PLAKSP MINGW64 /d/GitTest/Git_01 (main)
$ git commit -m "fisrt commit a,b 파 일 인 사 말"
[main (root-commit) 583bdc6] fisrt commit a,b 파 일 인 사 말
2 files changed, 2 insertions(+)
create mode 100644 a.txt
create mode 100644 b.txt

KOSTA@DESKTOP-7PLAKSP MINGW64 /d/GitTest/Git_01 (main)
$
```



add와 커밋을 동시에

git commit -am "add와 commit을 동시에"



👉 명령어 : **git log**

```
KOSTA@DESKTOP-7PLAKSP MINGW64 /d/GitTest/Git_01 (main)
$ git log
commit 583bdc68db82c3a46e316c5439efb4fd5584bc67 (HEAD -> main)
Author: heejung <8253jang@daum.net>
Date: Sat Feb 24 15:56:49 2024 +0900

    fisrt commit a,b 파 일 인 사 말

KOSTA@DESKTOP-7PLAKSP MINGW64 /d/GitTest/Git_01 (main)
$ |
```

- 583bdc68db82c3a46e316c5439efb4fd5584bc67

: 메시지, 작성자 ,날짜 ,시간등의 정보가 함께 저장됨

local repo의 commit 이력 조회

git log

최신 3개 커밋만 조회

git log -3

git 로그를 간결하게 조회

git log --oneline

최신 3개 커밋만 간결하게 조회

git log -3 --oneline

git 로그를 그래프형태로 조회

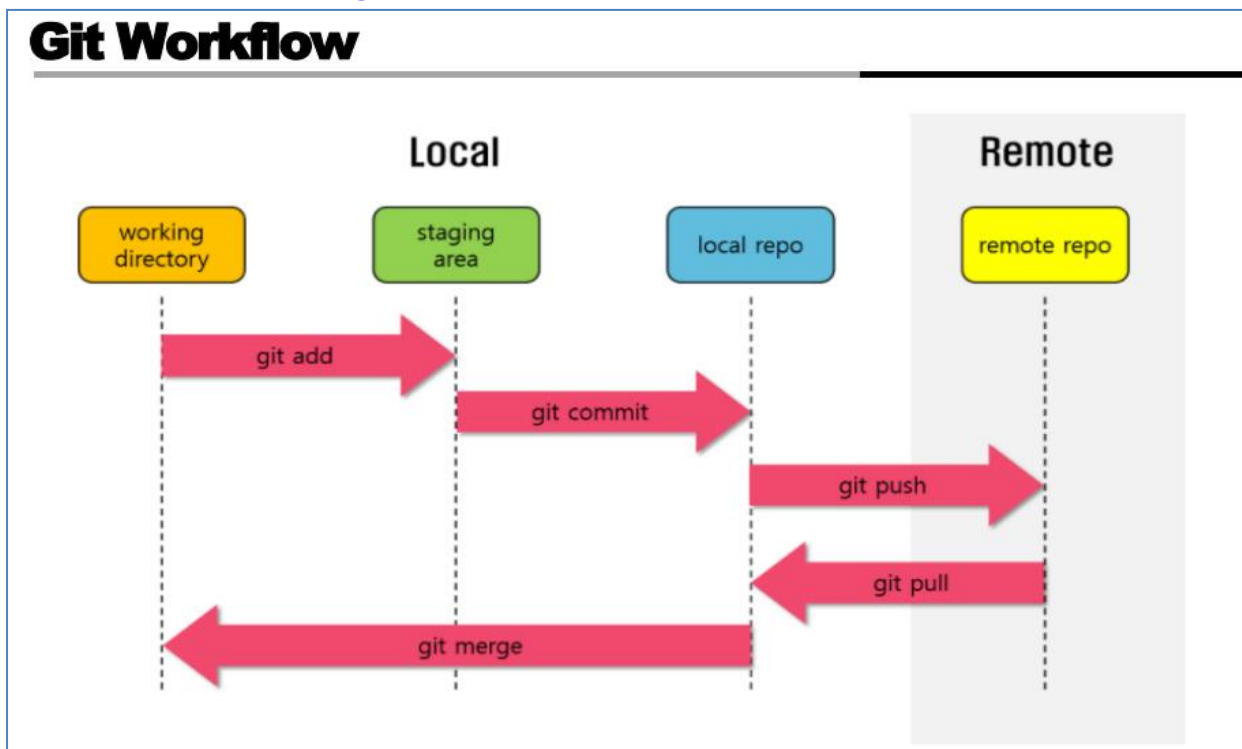
git log --graph

main브랜치 뿐만 전체 commit 이력 조회

git log --all

그림을 통해 다시 git workflow를 살펴보면,

Git Workflow



git 제거하기

👉 명령어 : **rm -rf .git**


```
KOSTA@DESKTOP-7PLAKSP MINGW64 /d/GitTest/Git_01 (main)
$ rm -rf .git

KOSTA@DESKTOP-7PLAKSP MINGW64 /d/GitTest/Git_01
$ |
```

Git을 실습해보자.

- 1) Git을 초기화 해본다.
- 2) 파일 2개(a.txt, b.txt)를 만들고 git 상태를 확인한다.
- 3) 커밋을 위한 파일을 git staging area에 추가한다.
- 4) 커밋 버전 한 개를 만든다.
- 5) c.txt파일을 생성하고, a.txt파일은 수정한다.
- 6) 변경된 상황을 커밋 한다.

```
KOSTA@DESKTOP-7PLAKSP MINGW64 /d/GitTest/Git_01 (main)
$ git status
On branch main
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   a.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    c.txt

no changes added to commit (use "git add" and/or "git commit -a")

KOSTA@DESKTOP-7PLAKSP MINGW64 /d/GitTest/Git_01 (main)
$ |
```

```

KOSTA@DESKTOP-7PLAKSP MINGW64 /d/GitTest/Git_01 (main)
$ git add .

KOSTA@DESKTOP-7PLAKSP MINGW64 /d/GitTest/Git_01 (main)
$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   a.txt
        new file:   c.txt

KOSTA@DESKTOP-7PLAKSP MINGW64 /d/GitTest/Git_01 (main)
$ |

```

```

KOSTA@DESKTOP-7PLAKSP MINGW64 /d/GitTest/Git_01 (main)
$ git log
commit e3f7a2dd43645d2b1fa6d2219390aa228fac44dc (HEAD -> main)
Author: heejung <8253jang@daum.net>
Date:   Sat Feb 24 16:40:40 2024 +0900

    second commit a 수정 , c파일 생성

commit 94260f0293cfd17edf284c7b3d31117bbc52fa5e
Author: heejung <8253jang@daum.net>
Date:   Sat Feb 24 16:37:28 2024 +0900

    first commit a,b파일 생성

```

- 7) b.txt 파일을 수정하고 a.txt파일을 삭제한다.
- 8) 변경된 상황을 커밋한다.
- 9) 최종 git log를 확인 해본다.

```

KOSTA@DESKTOP-7PLAKSP MINGW64 /d/GitTest/Git_01 (main)
$ git status
On branch main
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        deleted:   a.txt
        modified:  b.txt

no changes added to commit (use "git add" and/or "git commit -a")

KOSTA@DESKTOP-7PLAKSP MINGW64 /d/GitTest/Git_01 (main)
$ |

```

```

KOSTA@DESKTOP-7PLAKSP MINGW64 /d/GitTest/Git_01 (main)
$ git add .

KOSTA@DESKTOP-7PLAKSP MINGW64 /d/GitTest/Git_01 (main)
$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        deleted:    a.txt
        modified:   b.txt

KOSTA@DESKTOP-7PLAKSP MINGW64 /d/GitTest/Git_01 (main)
$ |

```

```

KOSTA@DESKTOP-7PLAKSP MINGW64 /d/GitTest/Git_01 (main)
$ git log
commit 15df2f037e6f5b30258bb1124fcea8b1a1681645 (HEAD -> main)
Author: heejung <8253jang@daum.net>
Date:   Sat Feb 24 16:45:55 2024 +0900

    third commit a 파일 삭제 , b파일 수정

commit e3f7a2dd43645d2b1fa6d2219390aa228fac44dc
Author: heejung <8253jang@daum.net>
Date:   Sat Feb 24 16:40:40 2024 +0900

    second commit a 수정 , c파일 생성

commit 94260f0293cfd17edf284c7b3d31117bbc52fa5e
Author: heejung <8253jang@daum.net>
Date:   Sat Feb 24 16:37:28 2024 +0900

    first commit a,b파일 생성

```

특정 커밋 ID로 전환

git checkout 특정commitID

HEAD는 현재 체크아웃된 브랜치의 커밋을 의미

- 다시 원래의 commit 으로 돌아오기
 - git checkout 브랜치(main 등)
 - git switch 브랜치(main 등)

git checkout : 브랜치 전환, 파일 체크아웃, 커밋으로 돌아가기

git switch : 오직 브랜치 전환에만 사용

```

MINGW64:/d/GitTest/eclipseWork/Git_295
KOSTA@DESKTOP-7PLAKSP MINGW64 /d/GitTest/eclipseWork/Git_295 (main)
$ git checkout main
Already on 'main'

KOSTA@DESKTOP-7PLAKSP MINGW64 /d/GitTest/eclipseWork/Git_295 (main)
$ git checkout 03faf0a
Note: switching to '03faf0a'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

    git switch -c <new-branch-name>

Or undo this operation with:

    git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at 03faf0a remote Ttest02 bb메소드 수정

KOSTA@DESKTOP-7PLAKSP MINGW64 /d/GitTest/eclipseWork/Git_295 ((03faf0a...))
$ git log --oneline
03faf0a (HEAD) remote Ttest02 bb메소드 수정
54da1ad Test02추가
9e7923d first commit

KOSTA@DESKTOP-7PLAKSP MINGW64 /d/GitTest/eclipseWork/Git_295 ((03faf0a...))
$ git switch main
Previous HEAD position was 03faf0a remote Ttest02 bb메소드 수정
Switched to branch 'main'

KOSTA@DESKTOP-7PLAKSP MINGW64 /d/GitTest/eclipseWork/Git_295 (main)
$ git log --oneline
d412d8c (HEAD -> main, tag: v1.0, origin/main) Merge pull request #2 from 8253jang/feat/member
bb0465d Merge branch 'main' into feat/member
4b50049 원격에서 Test03.java
6375f78 (origin/feat/member) feat/member Test03파일 수정
c1881b2 Merge pull request #1 from 8253jang/feat/test
7e5323f (origin/feat/test) feat/test 첫번째 커밋
26e231c stash pop 후 충돌 해결 커밋
715f641 원격 Test02.java 내용 수정
6c257c6 충돌 해결 후 커밋
  
```

03faf0a 커밋ID로 전환

HEAD 현재 전환된 커밋

다시 main으로 전환

과거로 돌아가는 2가지 방법

👉 Reset vs Revert

1. Reset

: 시간을 과거로 돌리는것, 과거로 돌아간 후에 이후 흔적은 지워버린다.

[git reset -(option) 명령어]

- ✓ git reset -soft HEAD^ : 커밋한 내용만 되돌려 줌
- ✓ git reset -mixed HEAD^ : add전으로 되돌려 줌
- ✓ git reset -hard HEAD^ : working area에서 작업한 내용까지
전부 바로 전 commit으로 되돌아감

👉명령어

git reset head^ : 기본이 mixed상황이다.

git reset- -hard 돌아갈해시코드

2. Revert

: 최신을 삭제하는 것이 아니라 전의 변화를 거꾸로 수행하여
커밋버전을 하나 더 만드는 것.

서로간에 협업 할때는 Reset보다는 Revert를 권장

👉명령어

git revert HEAD

git revert HEAD -개수

git revert 기존의커밋ID

시나리오01

Local repository 에 커밋 이후 취소

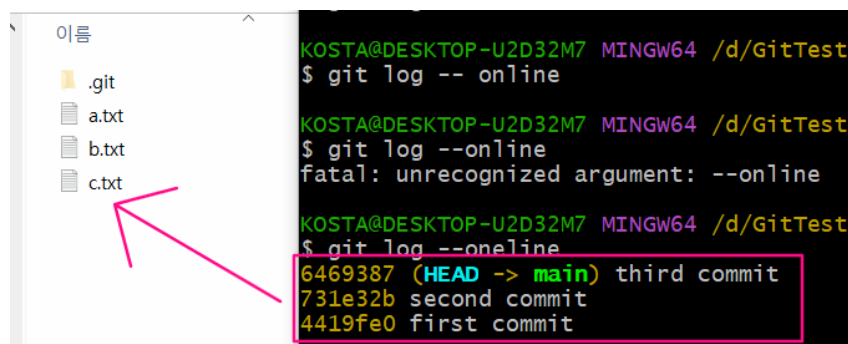
- **git reset HEAD~1(또는 HEAD^)**
 - 마지막 커밋 취소(가장 최신의 커밋을 취소)
 - unstaged 상태로 만듦(working directory 로 내려온다)
 - mixexd 상황이다.
 -
- **git reset --soft HEAD~1**
 - staged 상태 유지

```
KOSTA@DESKTOP-7PLAKSP MINGW64 /d/GitTest/eclipsework/Git_295 (main)
$ git reset head^
Unstaged changes after reset:
M      src/exam/Test.java
```

```
KOSTA@DESKTOP-7PLAKSP MINGW64 /d/GitTest/eclipsework/Git_295 (main)
$ git status
On branch main
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   src/exam/Test.java

no changes added to commit (use "git add" and/or "git commit -a")
```

- **git reset --hard HEAD~1**
 - working directory 에서 작업한 내용까지 바로 전 커밋상태로 되돌린다.



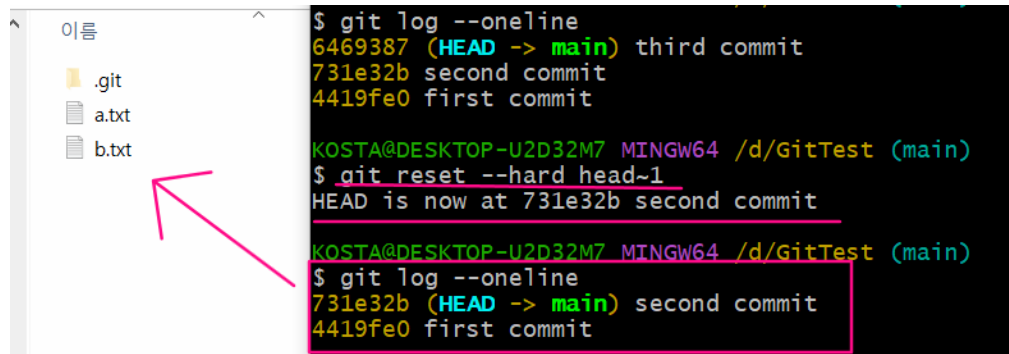
```
KOSTA@DESKTOP-U2D32M7 MINGW64 /d/GitTest
$ git log -- online

KOSTA@DESKTOP-U2D32M7 MINGW64 /d/GitTest
$ git log --online
fatal: unrecognized argument: --online

KOSTA@DESKTOP-U2D32M7 MINGW64 /d/GitTest
$ git log --oneline
6469387 (HEAD -> main) third commit
731e32b second commit
4419fe0 first commit
```

위 상황에서

Git reset --hard head~1 실행하면 c.txt 파일이 삭제된다.



```

$ git log --oneline
6469387 (HEAD -> main) third commit
731e32b second commit
4419fe0 first commit

KOSTA@DESKTOP-U2D32M7 MINGW64 /d/GitTest (main)
$ git reset --hard head~1
HEAD is now at 731e32b second commit

KOSTA@DESKTOP-U2D32M7 MINGW64 /d/GitTest (main)
$ git log --oneline
731e32b (HEAD -> main) second commit
4419fe0 first commit
  
```

시나리오02

Origin까지 push된 이후 취소(push 후 취소)

- **git revert 커밋 ID**
- 특정 커밋버전을 취소시키는 새로운 commit 을 생성 후에 다시 push
- vi 모드를 통해 commit 메시지 작성하는 창으로 바로 변환 → wq!

이미 push 된 것은 취소 할수 없다. 이 변경사항을 다시 뒤집어서 새로운 커밋 아이디를 만들어 그것을 새롭게 push 한다. 만약, 중요한 정보(비밀번호)를 push 한 경우 revert 를 해도 이전 커밋을 보면 남아 있기 때문에 revert 를 해도소용이 없다. 비밀번호를 변경하는 것이 더 좋다.

실습- 원격에 push된 상황에서

git revert 커밋아이디

```
KOSTA@DESKTOP-7PLAKSP-MINGW64 /d/GitTest/eclipsework/Git_295 (main)
$ git log 3
commit 9af109e355873ed77beecad437a89c8296c625b2 (HEAD -> main, origin/main)
Author: BongBong <8253jang@daum.net>
Date: Sat Mar 15 22:10:36 2025 +0900

    revert 원 격 push

commit d412d8c1258e18b501ae67935b168b2a077df184 (tag: v1.0)
Merge: 4b50049 bb0465d
Author: BongBong <8253jang@daum.net>
Date: Sat Mar 15 12:25:58 2025 +0900

    Merge pull request #2 from 8253jang/feat/member

    feat/member Test03파 일 수 정

commit bb0465db740b3e4d8ee386d4ef92e0db1b2aa8a1
Merge: 6375f78 4b50049
Author: BongBong <8253jang@daum.net>
Date: Sat Mar 15 12:23:11 2025 +0900

    Merge branch 'main' into feat/member

KOSTA@DESKTOP-7PLAKSP-MINGW64 /d/GitTest/eclipsework/Git_295 (main)
$ git revert 9af109e35587
```

취소하는 커밋 메시지를 확인하고 :wq! 입력한다.

[illegible]

git log --oneline으로 커밋이력 확인한다.

새로운 커밋아이디 생성됨

```

MINGW64:/d/GitTest/eclipseWork/Git_295
KOSTA@DESKTOP-7PLAKSP MINGW64 /d/GitTest/eclipseWork/Git_295 (main)
$ git log --oneline
2152bcc (HEAD -> main) Revert "revert 원격 push"
9af109e (origin/main) revert 원격 push
d412d8c (tag: v1.0) Merge pull request #2 from 8253jang/feat/member
bb0465d Merge branch 'main' into feat/member
4b50049 원격에서 Test03.java
  
```

git push origin main

```

MINGW64:/d/GitTest/eclipseWork/Git_295
7e9fbcd 원격에서 Test02.java bb 수정함
75b16a2 충돌해결 후 커밋
8e06b6e Test01 bb 메소드 로커에서 수정함
596dee6 원격에서 Test02 bb메소드 수정함
f4f44f5 Merge branch 'main' of https://github.com/8253jang/Git_295
94cd6c4 로컬에서 다른 파일수정함
d80f66e 원격에서 수정함
4f65b22 원격에서 수정함
03faf0a remote Ttest02 bb메소드 수정
54dalad Test02추가
9e7923d first commit
$ git push origin main
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (4/4), 335 bytes | 335.00 KiB/s, done.
Total 4 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/8253jang/Git_295.git
  9af109e..2152bcc  main -> main
  
```

원격에 push 한다