

졸업작품 포트폴리오

학부

컴퓨터공학부

학과

컴퓨터 정보공학과

과목명

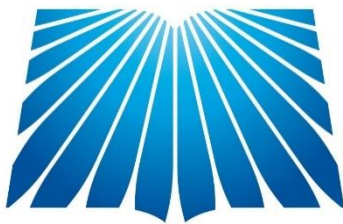
졸업 작품

학번

20180665

성명

황민식



동양미래대학교
DONGYANG MIRAE UNIVERSITY

목차

1.	Git의 이해
2.	Git의 명령어
3.	Git의 명령어 종류
4.	Git의 명령어 실행
5.	Git의 4가지 영역
6.	Git reset
7.	분산 환경에서의 워크 플로
8.	Wifi 쉼드
9.	아두이노 블루투스
10.	블루투스를 통한 아두이노 연동
11.	안드로이드 서버 연동 이미지 업로드

Git의 이해

깃 : 분산 버전 관리 시스템

깃허브 : 깃의 저장소 및 관리 서비스

버전관리 : 파일의 수정 이력 관리, 여러 파일의 수정 묶음 관리

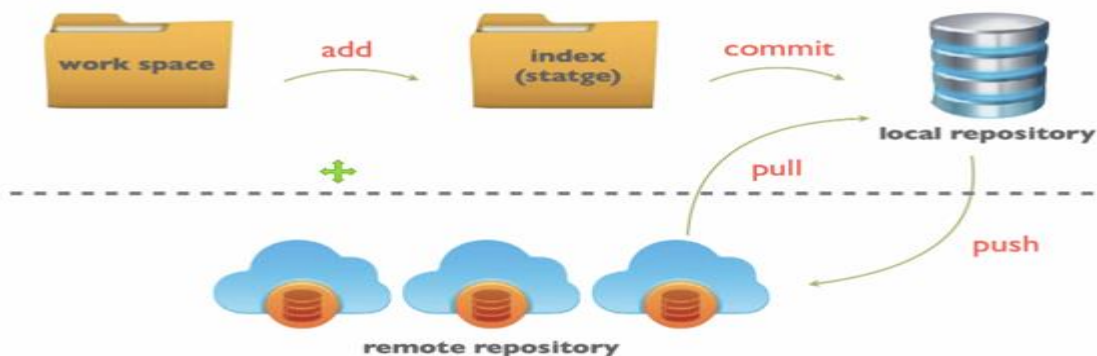
형상관리 : 버전관리(수정이 안된 것은 그대로 유지, 수정된 부분만 저장 관리)

중앙집중식 버전 관리 : 서버의 문제(복구 불가능)

분산 버전 관리 : 서버의 문제(분산된 저장소에 의해 복구가 가능)

깃의 구조

작업공간, 스테이징 영역, 저장소



저장소(Git repository)

파일이 변경 이력 별로 구분되어 저장

비슷한 파일이라도 실제 내용 일부 문구가 서로 다르면 다른 파일로 인식하기 때문에 파일을 변경 사항 별로 구분해 저장

원격 저장소(Remote Repository)

파일이 원격 저장소 전용 서버에서 관리되며 여러 사람이 함께 공유하기 위한 장소

로컬 저장소

내 PC에 파일이 저장되는 개인 전용 저장소

활용

평소에는 내 PC의 로컬 저장소에서 작업하다가 작업한 내용을 공개하고 싶을때
에 원격 저장소에 업로드

Git 명령어

셸 프로세스를 띄우고 git 명령어를 실행하는 방법이 있다. 이게 가장 표준적인
방법으로 git의 모든 기능을 사용할 수 있다. 웬만한 환경에서는 명령어를 프로세
스로 실행하는 것은 간단하므로 이방법은 사용하기 쉬운편이다. 그러나 이 방법
은 몇 가지 제약사항이 있다.

1. 결과가 텍스트로 출력된다. Git이 상황에 따라 출력하는 결과를 파싱해야
한다. 진행 상태와 결과 정보를 구분해서 잘 읽어야 해서 어렵고 예러가
나기 쉽다.
2. 에러처리가 어렵다. 저장소가 깨져 있거나 사용자가 잘못 설정했을 때 git
은 그냥 제대로 실행되지 않을 뿐이다.
3. 프로세스를 관리해야 한다는 점. 별도의 프로세스로 git을 실행하기 때문에
애플리케이션에 불필요한 복잡성이 추가된다. 여러 프로세스를 조종하는
일은 지뢰밭이라 할 수 있다.

Git 명령어 종류

현재 폴더 확인 : `$pwd`

도움말 보기 : `$ git help`

버전 보기 : `$ git --version`

사용자 설정

\$ git config -- global user.name [이름] : 자신이 생성한 커밋에 들어갈 이름 설정

\$ git config -- global user.email [이메일 주소] : 자신이 생성한 커밋에 들어갈 이메일 주소 설정

git config -- list : 설정 확인

\$ git config -- global alias.새명령어'원명령어' : 명령어를 다른 이름으로 지정 (\$ git config --global alias.st 'status')

저장소 생성

\$git init [프로젝트 이름] : 새로운 로컬 저장소를 생성하고 이름을 정함

\$git clone [url] : 기존 프로젝트의 모든 커밋 내역을 가져와 저장소를 만듦

\$git init, \$git init. : 현 폴더에 저장소 생성

\$git init[project-name] : 지정된 이름의 폴더를 만들면서 저장소 생성

git add

\$git add : Untracked에서 Index로, Modified에서 Index로

git status

\$git status : Git 상태 정보 보이기

Git commit

\$git commit, \$git commit-m "메시지" : Index에서 깃 저장소 로

\$git commit -am "메시지", \$git commit -am

\$git commit -a -m "메시지" \$git commit -a -m "메시지"

- 작업디렉토리에서 index를 거치지 않고 바로 깃 저장소로

\$git commit --amend -m "new message" : 새로운 커밋 메시지로 수정

git log

옵션

- P : 변경사항 확인
- -- oneline : 커밋 메시지만 한 줄씩 표시
- -- all : 모든 브랜치 로그 표시
- -- graph : 브랜치 트리 그래프 표시
- -n : 최근 n개 보이기

\$git log : 커밋기록 보기

\$git log -p [file] : 특정 파일의 커밋기록 보기

git show

\$git show[commit] : 특정 커밋에 포함된 변경 사항과 메타데이터를 표시

\$git show : 현재 브랜치의 가장 최근 커밋 정보를 확인함

\$git show 커밋해시값 : 특정 커밋 정보를 확인함

\$git show HEAD: HEAD 포인터가 가리키는 커밋 정보를 확인함

\$git show 커밋해시값 또는 HEAD^ : ^ 표시 한 개면 한 개전 두 개면 두 개전, 개수로 얼마나 이전 값인지 알 수 있음

\$git show 커밋해시값 또는 HEAD ~ 숫자 : ~숫자 는 명시적으로 몇 개 전인지 표시

git diff

\$git diff : commit된 파일 상태와 현재 수정 중인 상태 비교

\$git diff -- staged : commit된 파일 상태와 add된 파일 상태 비교

\$git diff[비교할 commit해쉬1] [비교할 commit해쉬2] : commit간의 상태 비교하기 – commit hash 이용

\$git diff HEAD HEAD^ : commit간의 상태 비교하기 – HEAD 이용 , 가장 최근의 커밋과 그전의 커밋을 비교한다.

되돌리기

\$git reset : 커밋을 아예 취소

\$git checkout[file] : 파일을 수정 전의 파일로 복구

\$git revert : 현재 HEAD를 특정 시점의 commit 상태로 변경하는 commit을 추가 로 수행

- 과거의 모든 커밋은 그대로 유지 관리되며 새로운 커밋을 추가해 측정 시점으로 이동

Git 명령어 실행

프로젝트(소스코드들이 있는 디렉토리)를 git repository로 만들기 위해서 사용
디렉토리를 git repository로 만들어야 git으로 버전 관리가 가능

```
MINGW64 /d/Git 연습/repo1
FORDESKTOP-482NQAB MINGW64 /d/Git 연습
$ pwd
/d/Git 연습
FORDESKTOP-482NQAB MINGW64 /d/Git 연습
$ mkdir repo1
FORDESKTOP-482NQAB MINGW64 /d/Git 연습
$ cd repo1
FORDESKTOP-482NQAB MINGW64 /d/Git 연습/repo1
$ pwd
/d/Git 연습/repo1
FORDESKTOP-482NQAB MINGW64 /d/Git 연습/repo1
$ ls -al
total 0
drwxr-xr-x 1 PC 197121 0  9월 12 15:44 ./
drwxr-xr-x 1 PC 197121 0  9월 12 15:44 ../
FORDESKTOP-482NQAB MINGW64 /d/Git 연습/repo1
$ git init
Initialized empty Git repository in D:/Git 연습/repo1/.git/
FORDESKTOP-482NQAB MINGW64 /d/Git 연습/repo1 (master)
$ ls -al
total 4
drwxr-xr-x 1 PC 197121 0  9월 12 15:44 ./
drwxr-xr-x 1 PC 197121 0  9월 12 15:44 ../
drwxr-xr-x 1 PC 197121 0  9월 12 15:44 .git/
FORDESKTOP-482NQAB MINGW64 /d/Git 연습/repo1 (master)
$
```

저장소 만든 후 초기

\$git status

On branch master

No commits yet

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

파일 2 개 만든후

\$git status

On branch master

No commits yet

Untracked files:

(use "git add <file>..." to include in what will be committed)

git-cli.txt

git-study.txt

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

```
MINGW64/c/Git Self Tutorial/git-repo01
drwxr-xr-x 1 PC 197121 0 9월 17 22:04 ./
drwxr-xr-x 1 PC 197121 0 9월 17 21:30 ../
drwxr-xr-x 1 PC 197121 0 9월 17 22:04 .git/

PC\LAPTOP-QK7LS4IS MINGW64 /c/Git Self Tutorial/git-repo01 (master)
$ git status
On branch master

No commits yet

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

PC\LAPTOP-QK7LS4IS MINGW64 /c/Git Self Tutorial/git-repo01 (master)
$ git status
On branch master

No commits yet

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

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

PC\LAPTOP-QK7LS4IS MINGW64 /c/Git Self Tutorial/git-repo01 (master)
$
```

Tracked(관리 반복대상) 와 Untracked(관리대상이 아님)로 나뉨

- 워킹 디렉토리의 모든 파일
- Tracked 파일 (Git 이 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 Area 에도 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 Git 은 그파일을 Modified 상태로 인식
- 실제로 커밋을 하기위해서는 이 수정한 파을 Staged 상태로 만들고 \$git add[file]
- Staged 상태의 파일을 커밋 \$git commit -m "message"

```

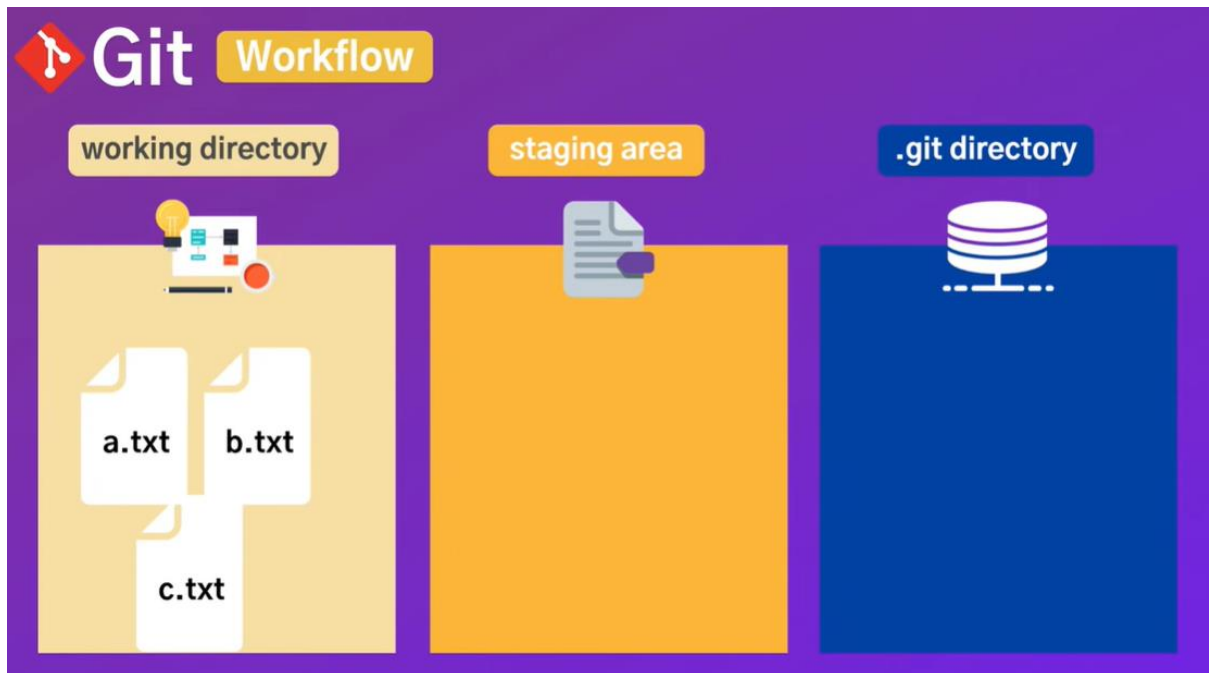
MINGW64/c/Git Self Tutorial/git-repo01
PC@LAPTOP-QK7LS41S MINGW64 /c/Git Self Tutorial/git-repo01 (master)
$ git add .
PC@LAPTOP-QK7LS41S MINGW64 /c/Git Self Tutorial/git-repo01 (master)
$ git status
On branch master
No commits yet
Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   git-cli.txt
        new file:   git-study.txt
PC@LAPTOP-QK7LS41S MINGW64 /c/Git Self Tutorial/git-repo01 (master)
$ git commit -m "처음 빈 파일로 저장"
[master (root-commit) a49bf60] 처음 빈 파일로 저장
 2 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 git-cli.txt
 create mode 100644 git-study.txt
PC@LAPTOP-QK7LS41S MINGW64 /c/Git Self Tutorial/git-repo01 (master)
$ git status
On branch master
nothing to commit, working tree clean
PC@LAPTOP-QK7LS41S MINGW64 /c/Git Self Tutorial/git-repo01 (master)
$ git log
commit a49bf6099f159f2ade082a3c0aac82739856a8ac (HEAD -> master)
Author: ai7dnn <ai7dnn@gmail.com>
Date: Sat Sep 18 18:10:32 2021 +0900
    처음 빈 파일로 저장
PC@LAPTOP-QK7LS41S MINGW64 /c/Git Self Tutorial/git-repo01 (master)
$

```

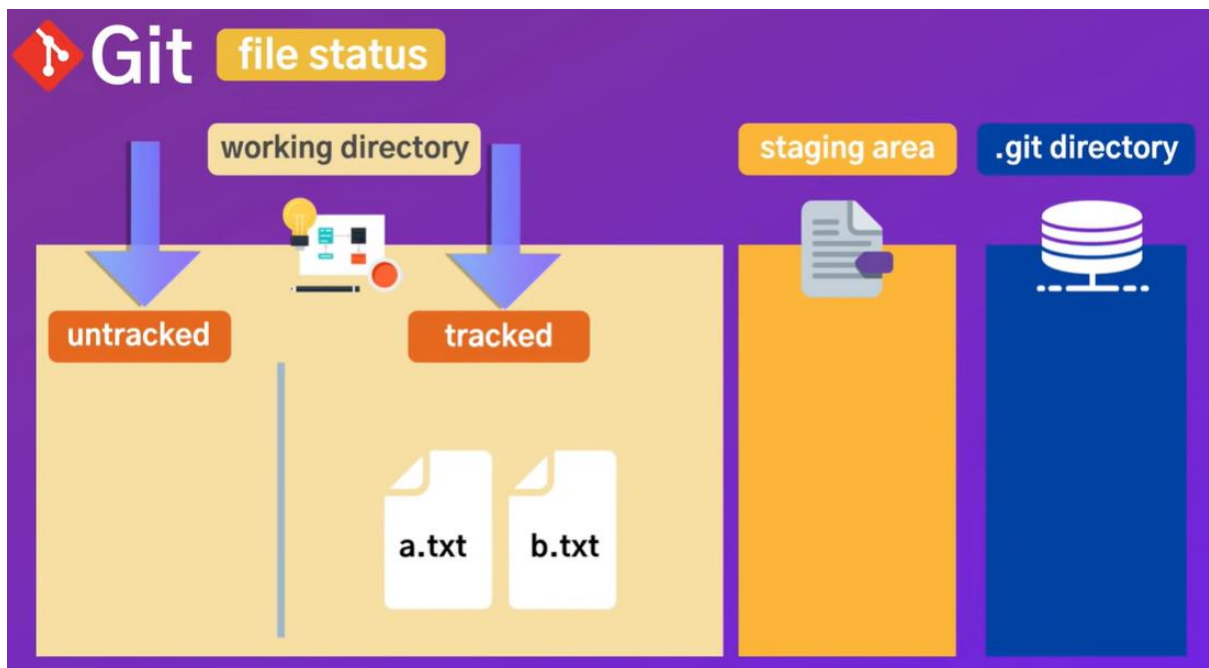
git log

- Commit history 확인
- Author 영역의 이름과 이메일 주소
- Git config 명령을 통해 세팅했던 user.name / user.email 값이 표기

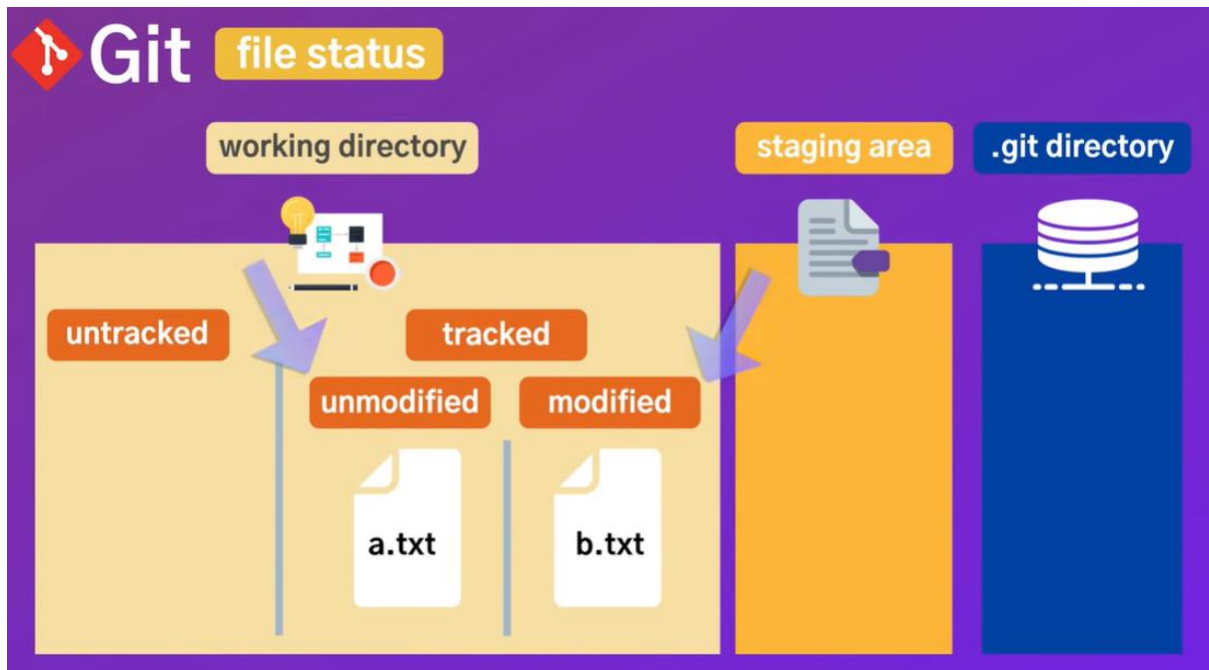
세가지 flow 상태



관리 대상 여부

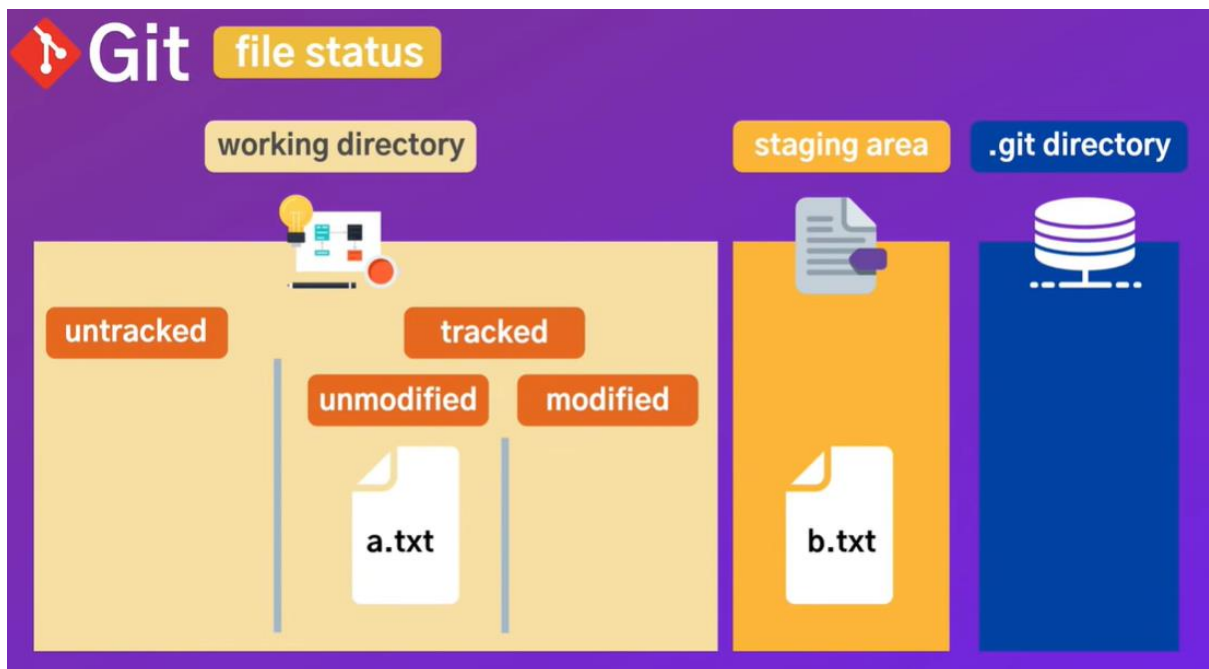


수정 여부

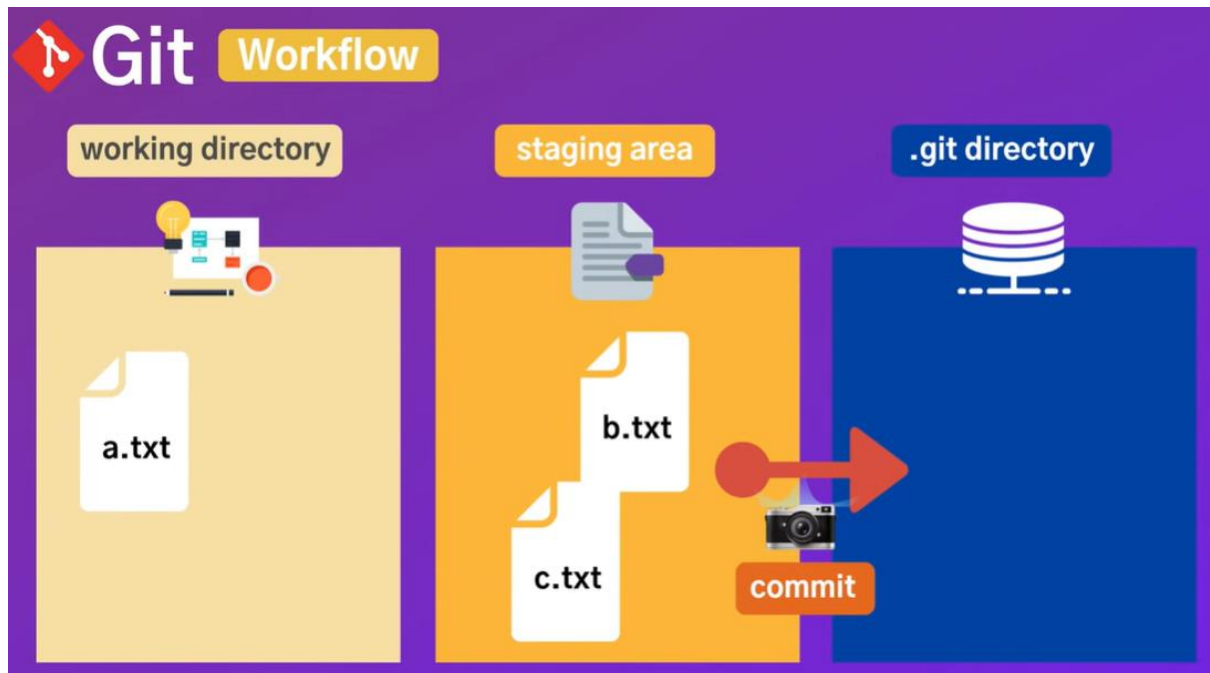


명령 add

수정 파일을 index 로

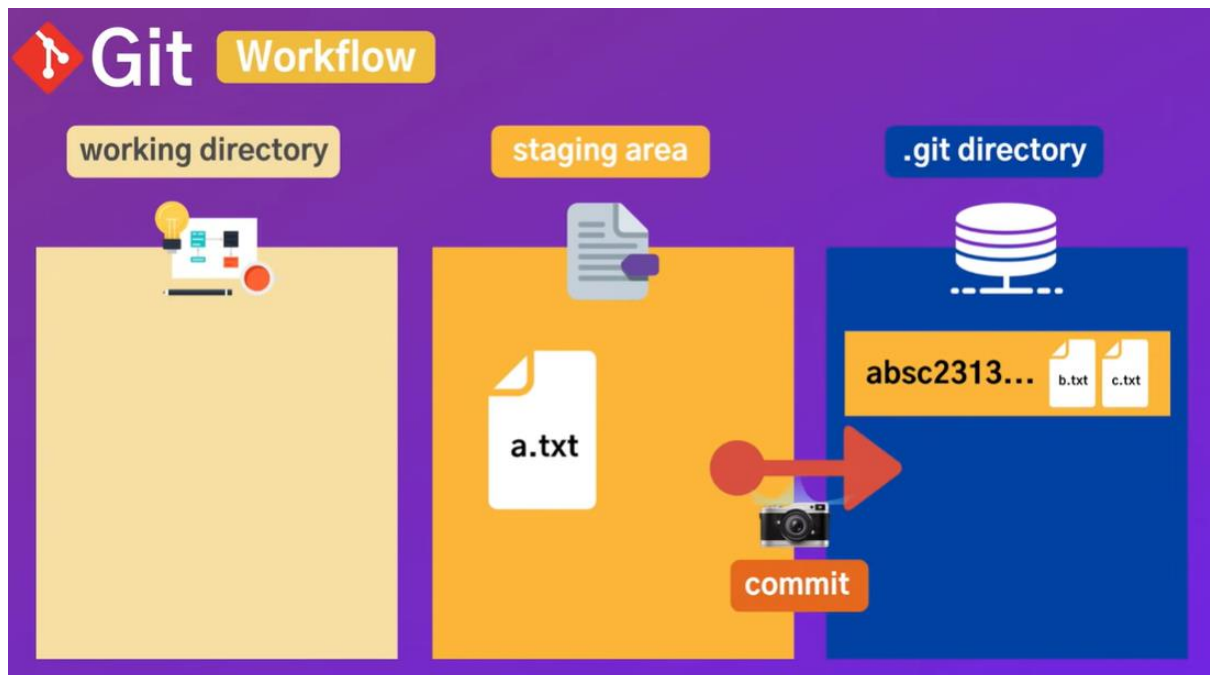


명령어 add commit



커밋 결과

버전 관리를 위한 git 디렉토리에 반영



환경 설정

모든 로컬 저장소에 적용할 사용자 정보를 설정합니다.

\$git config -- global user.name [name] : 자신이 생성한 커밋에 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 설정을 파일로 편집

\$git config -- global core.editor "notepad" : 자동 편집기 설정

다시 staged 에 있는 파일을 수정

- Git-study.txt 파일을 수정
- git status 명령 확인
- git add 명령을 실행
- 파일을 Staged 상태로

다시 staged 에 있는 파일을 수정

- 하나의 파일이 작업 디렉토리와 스테이징에 함께 있는 상태가 됨

이 상태에서 커밋을 하면

- 스테이징에 있는 파일의 내용을 버전 관리에 복사
- 작업 디렉토리에 있는 내용은 반영이 안됨

선택 MINGW64/c/Git Self Tutorial/git-repo01

PC@LAPTOP-QK7LS4IS MINGW64 /c/Git Self Tutorial/git-repo01 (master)

\$ git status

On branch master

Changes to be committed:

(use "git restore --staged <file>..." to unstage)

modified: git-study.txt

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: git-study.txt

PC@LAPTOP-QK7LS4IS MINGW64 /c/Git Self Tutorial/git-repo01 (master)

\$ git commit -m 'git-study 두 상태'

[master d9d1096] git-study 두 상태

1 file changed, 4 insertions(+)

PC@LAPTOP-QK7LS4IS MINGW64 /c/Git Self Tutorial/git-repo01 (master)

\$ git log

commit d9d1096ad9bc8417e3558502a92083c9b66472d5 (HEAD -> master)

Author: ai7dnn <ai7dnn@gmail.com>

Date: Sun Sep 19 12:57:46 2021 +0900

git-study 두 상태

commit 180a81e91f60e82e8325ed644193baa97df299c9

Author: ai7dnn <ai7dnn@gmail.com>

Date: Sun Sep 19 12:44:58 2021 +0900

제목

두번째 커밋

내용

README 파일 생성

commit a49bf6099f159f2ade082a3c0aac82739856a8ac

Author: ai7dnn <ai7dnn@gmail.com>

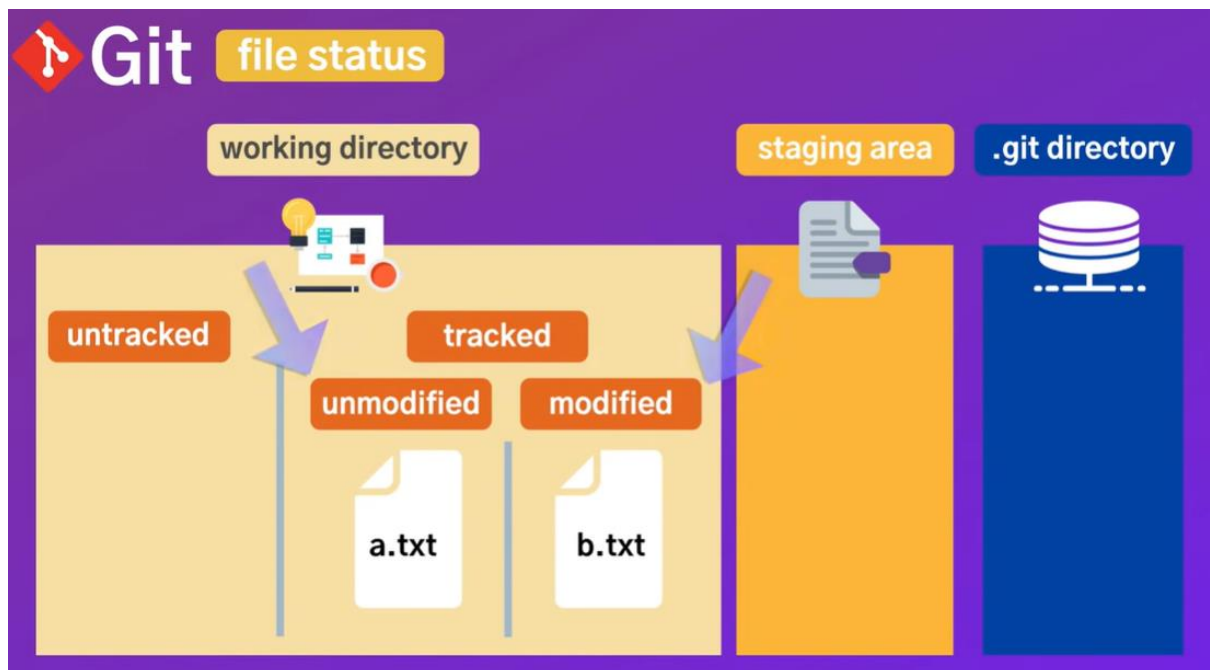
Date: Sat Sep 18 18:10:32 2021 +0900

처음 빈 파일로 저장

스테이징 상태에서 수정 을 하고 커밋을 하면 동시에 존재한다.



작업 디렉토리와 저장소에 함께 있는 경우



작업하고 있는 staging area나 working directory에서 작업하는 내용을 초기화 하는 방법

Unstaging a staged file

\$git restore -- staged 파일명

\$git reset HEAD 파일명 : HEAD가 가리키는 시점의 버전으로 파일을 unstage하고 되돌린다.

Unstaging a modified file

\$git restore 파일명 : modified 내용 삭제한다. 즉, 수정한 내용을 삭제해버리고 기존 상태로 되돌린다.

Git의 4가지 영역

1. Working Directory(작업영역)

- 프로젝트 디렉토리이며, 개발자가 직접 코드를 수정하는 공간을 의미
- .git 을 제외한 모든 영역에 해당

2. Index (Staging Area)

- Working Directory 에서 Repository 로 정보가 저장되기 전 준비 영역
- .git/index 파일로 관리

3. Repository(저장소)

- 파일이나 폴더를 변경 이력 별로 저장해두는 영역
- .git 디렉토리 내에 존재
- Local, Remote Repository 로 구분

4. Stash(임시영역)

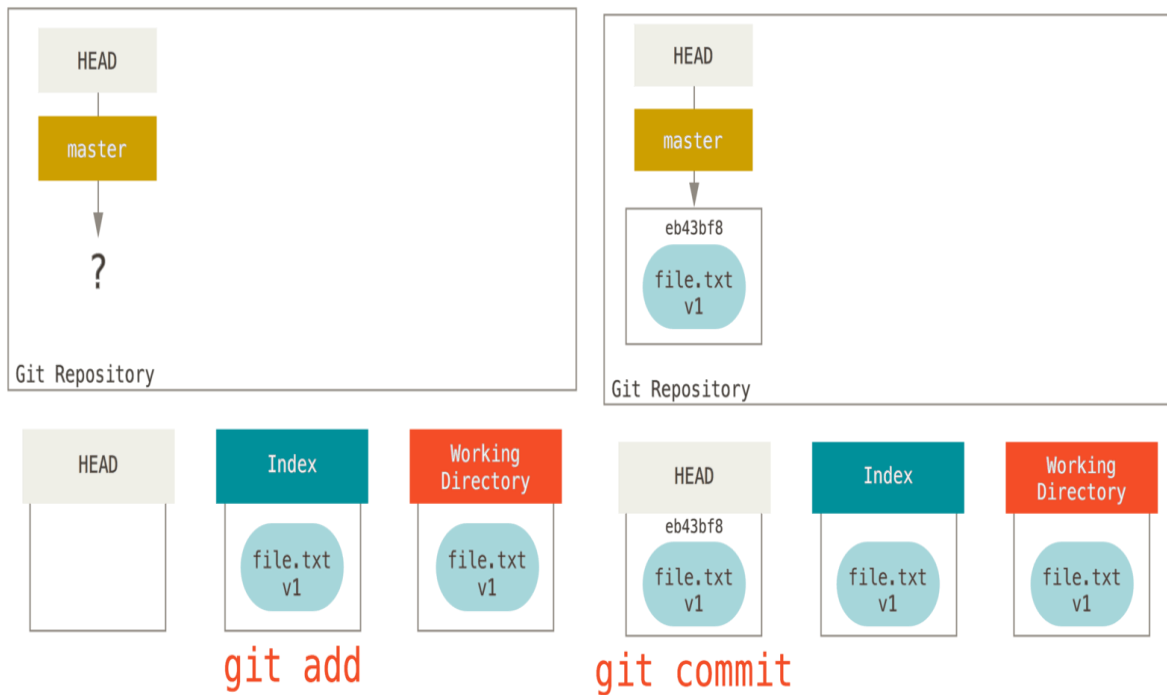
- 임시적으로 작업 사항을 저장해두고, 나중에 꺼내올 수 있는 영역

상태

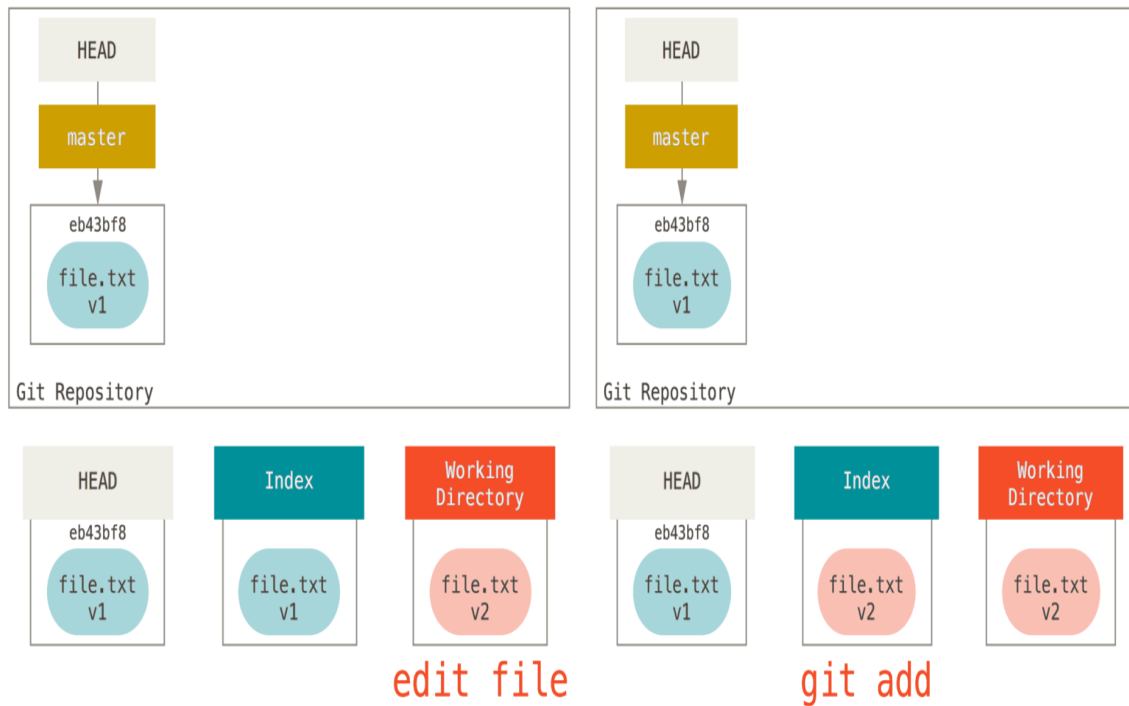
작업 디렉토리에 파일 저장



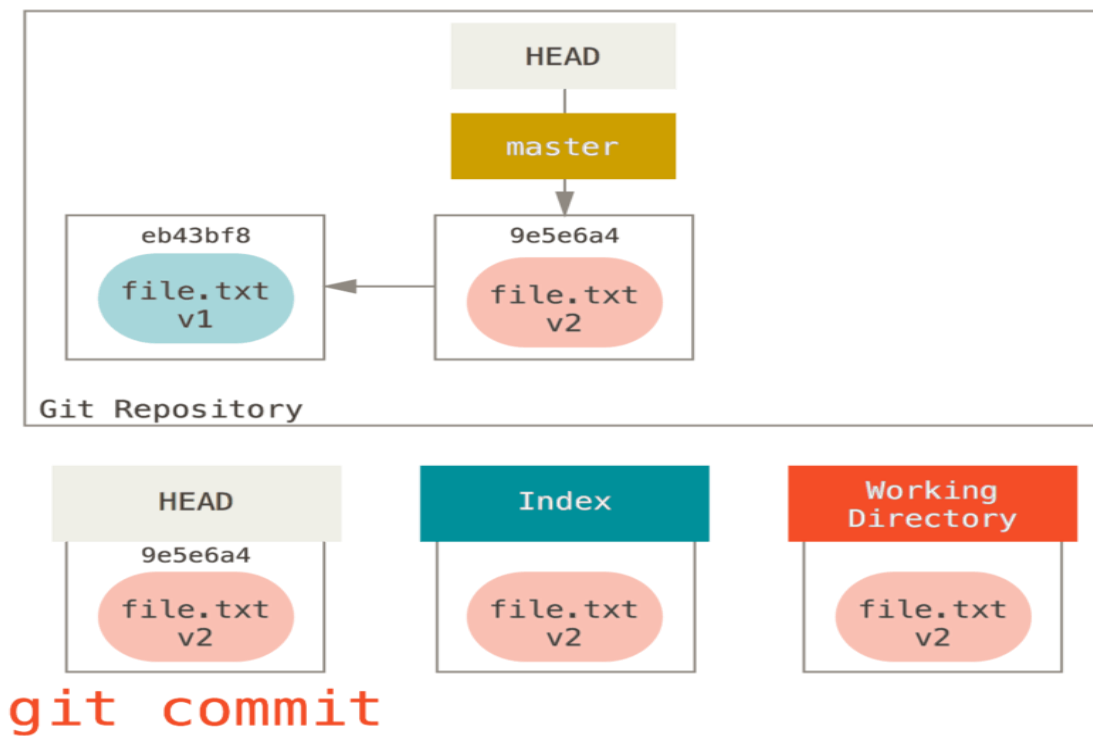
Add 와 commit



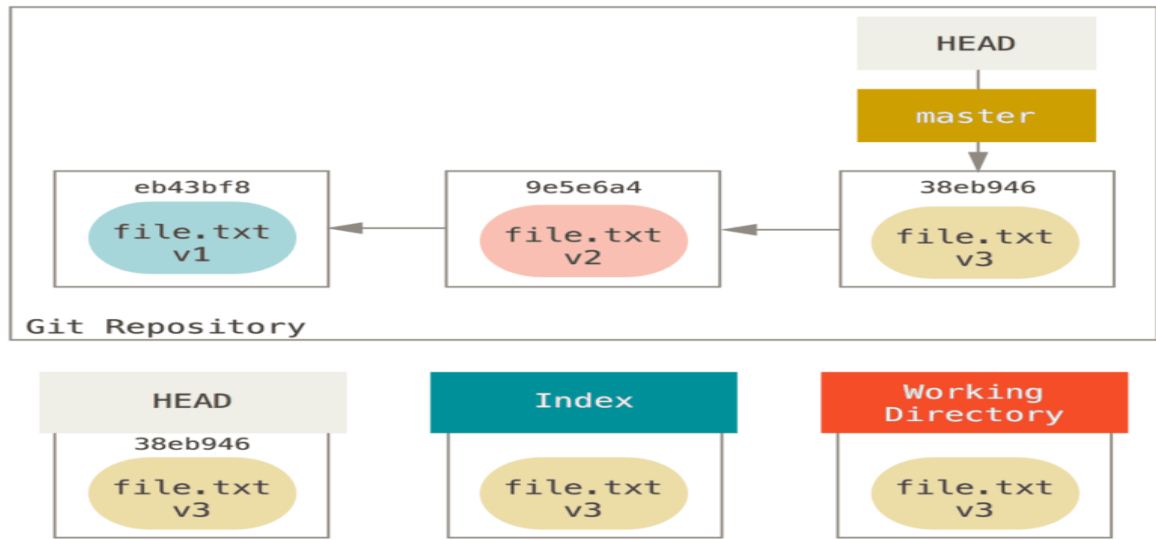
파일 수정 과 add



Commit



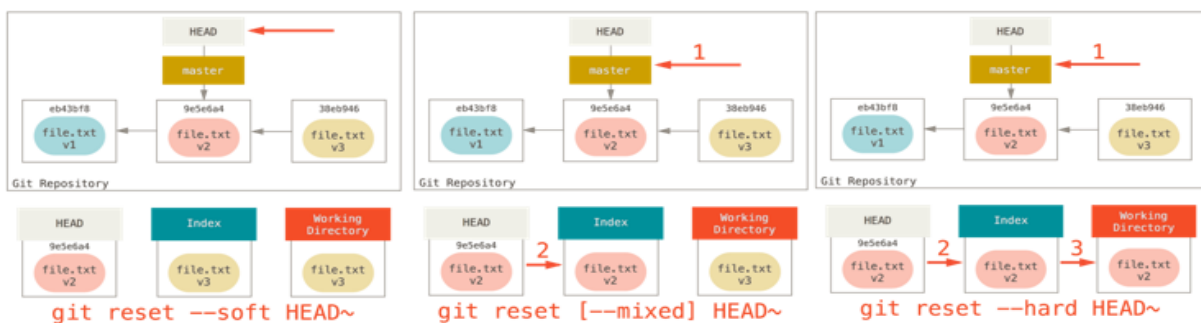
Commit 3회 상태



git reset

\$git reset<옵션><돌아가고싶은 커밋>: hard, mixed, soft 세가지

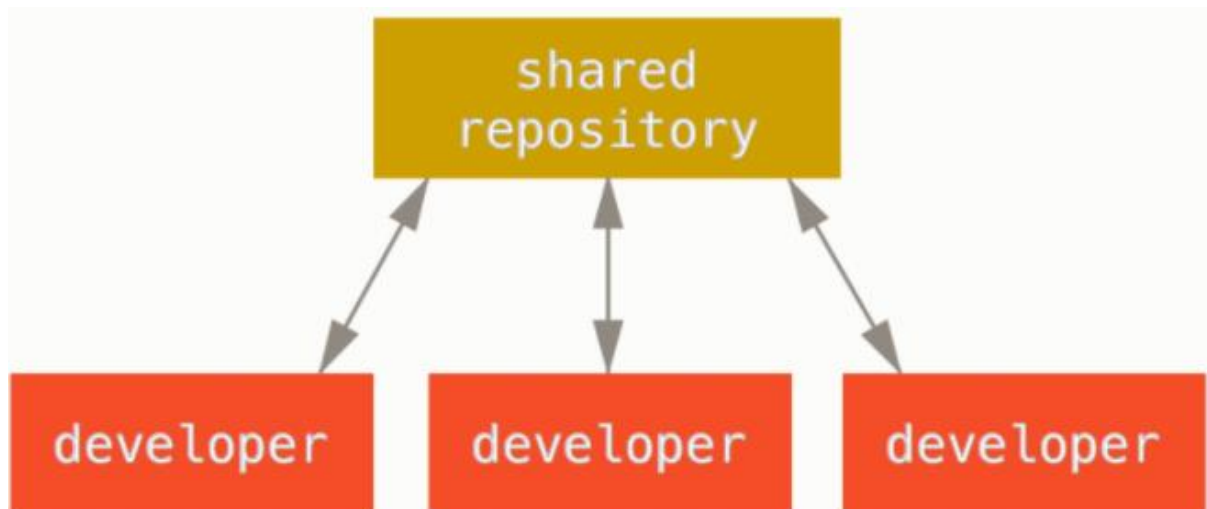
- Hard : 돌아가려는 이력 이후의 모든 내용을 삭제 , 다시 돌아가려면 (파일수정, add, commit필요)
- Mixed : 역시 이력은 되돌려집니다. 이후에 변경된 내용에 대해서는 남아있지만 인덱스는 초기화 된다. 커밋을 하려면 다시 변경된 내용은 추가해야 하는 상태 다시 돌아가려면(Add, commit이 필요)
- Soft : 돌아가려 했던 이력으로 되돌아 갔지만, 이후의 내용이 지워지지 않고, 해당 내용의 인덱스 도 그대로 있다. 바로 다시 커밋할 수 있는 상태로 남아 있는 것 다시 돌아가려면(commit 만 필요)



분산 환경에서의 워크플로

중앙집중형 버전 관리 시스템과는 달리 Git은 분산형이다. Git은 구조가 매우 유연하기 때문에 여러 개발자가 함께 작업하는 방식을 더 다양하게 구성할 수 있다. 중앙집중형 버전 관리 시스템에서 각 개발자는 중앙 저장소를 중심으로 하는 한 노드일 뿐이다. 하지만 Git에서는 각 개발자의 저장소가 하나의 노드이기도 하고 중앙 저장소 같은 역할도 할 수 있다. 즉 모든 개발자는 다른 개발자의 저장소에 일한 내용을 전송하거나, 다른 개발자들이 참여할 수 있도록 자신이 운영하는 저장소 위치를 공개할 수도 있다. 이런 특징은 프로젝트나 팀이 코드를 운영할 때 다양한 워크플로를 만들 수 있도록 해준다. 이런 유연성을 살려 저장소를 운영하는 몇 가지 방식을 소개한다. 각 방식의 장단점을 살펴보고 그 방식 중 하나를 고르거나 여러 가지를 적절히 섞어 쓰면된다.

중앙 집중식 워크플로



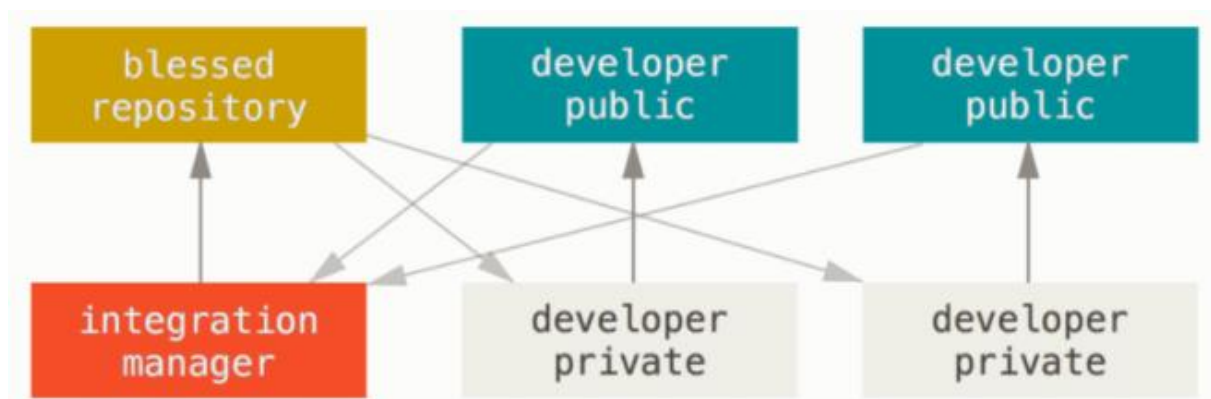
1. 한 개발자가 자신이 한 일을 커밋하고 아무 문제 없이 서버에 push한다.
2. 다른 개발자는 자신의 일을 커밋하고 push 하기전에 첫번째 개발자가 한 일을 먼저 Merge 해야 한다.

Merge 를 해야 첫 번째 개발자가 작업한 내용을 덮어쓰지 않는다. 이런 개념은 Subversion 과 같은 중앙 집중식 버전 관리 시스템에서 사용하는 방식이고 Git 에서도 사용 할 수 있다.

Integration - Manager 워크플로

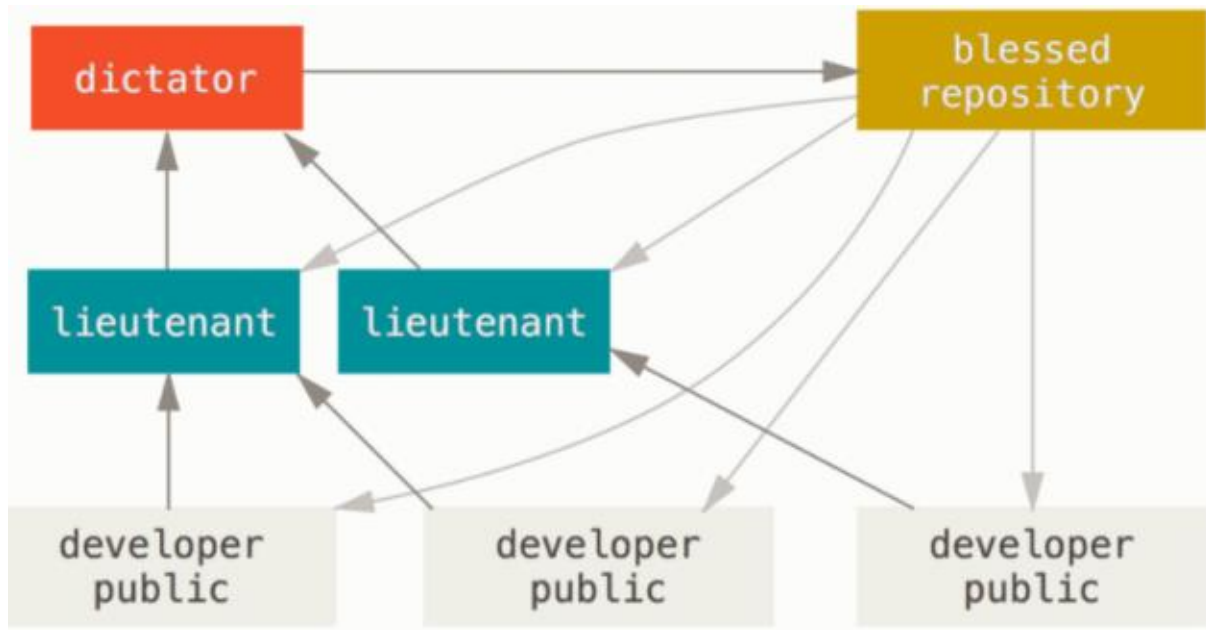
Git을 사용하면 리모트 저장소를 여러 개 운영할 수 있다. 다른 개발자는 읽기만 가능하고 자신은 쓰기도 가능한 공개 저장소를 만드는 워크플로도 된다. 이 Workflow에는 보통 프로젝트를 대표하는 공식 저장소가 있다.

1. 프로젝트 Integration-Manager 는 프로젝트 메인 저장소에 Push 를 한다.
2. 프로젝트 기여자는 메인 저장소를 Clone 하고 수정한다.
3. 기여자는 자신의 저장소에 Push 하고 Integration-Manager 가 접근할 수 있도록 공개해 놓는다.
4. 기여자는 Integration-Manager 에게 변경사항을 적용해 줄 것을 이메일로 요청한다.
5. Integration-Manager 는 기여자의 저장소를 리모트 저장소로 등록하고 수정사항을 Merge 하여 테스트한다.
6. Integration-Manager 는 Merge 한 사항을 메인 저장소에 Push 한다.



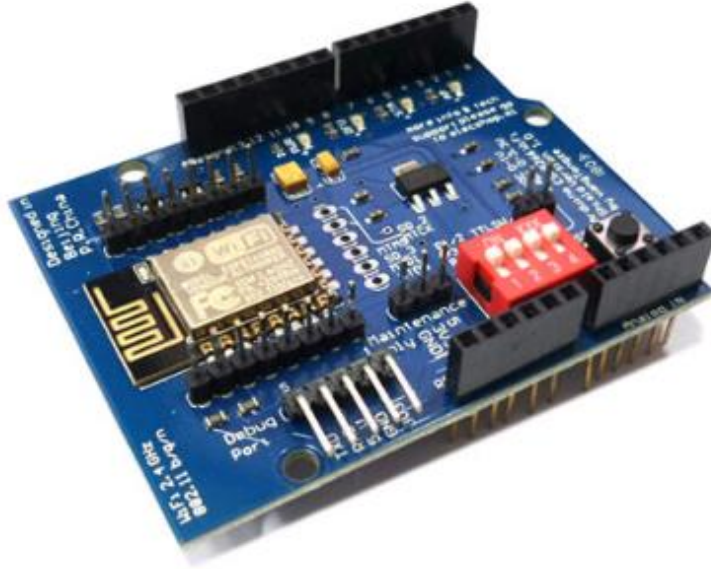
Dictator and Lieutenants 워크플로

1. 개발자는 코드를 수정하고 master 브랜치를 기준으로 자신의 토픽 브랜치를 Rebase 한다. 여기서 master 브랜치란 공식 저장소의 브랜치를 말한다.
2. Lieutenant 들은 개발자들의 수정사항을 자신이 관리하는 master 브랜치에 Merge 한다.
3. Dictator 는 Lieutenant 의 master 브랜치를 자신의 master 브랜치로 Merge 한다.
4. Dictator 는 자신의 master 브랜치를 Push 하며 다른 모든 개발자는 Dictator 의 master 브랜치를 기준으로 Rebase 한다.



이 세가지 워크플로가 Git 같은 분산 버전 관리 시스템에서 주로 사용하는 것들이다.

WIFI 쉴드



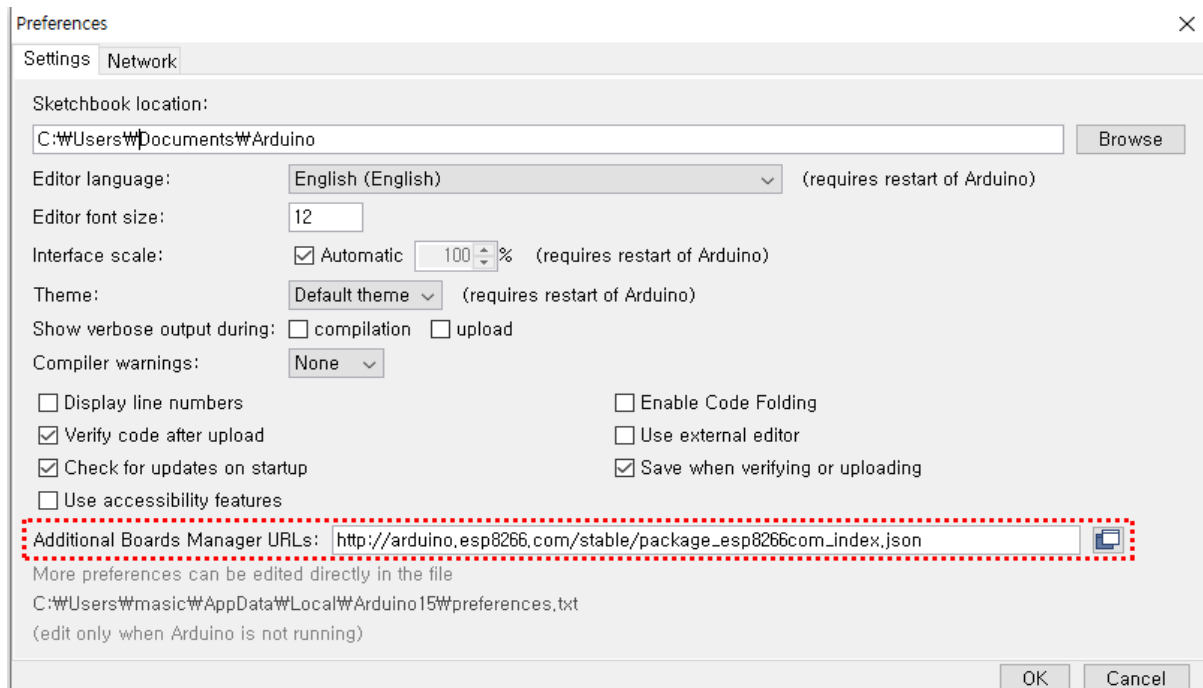
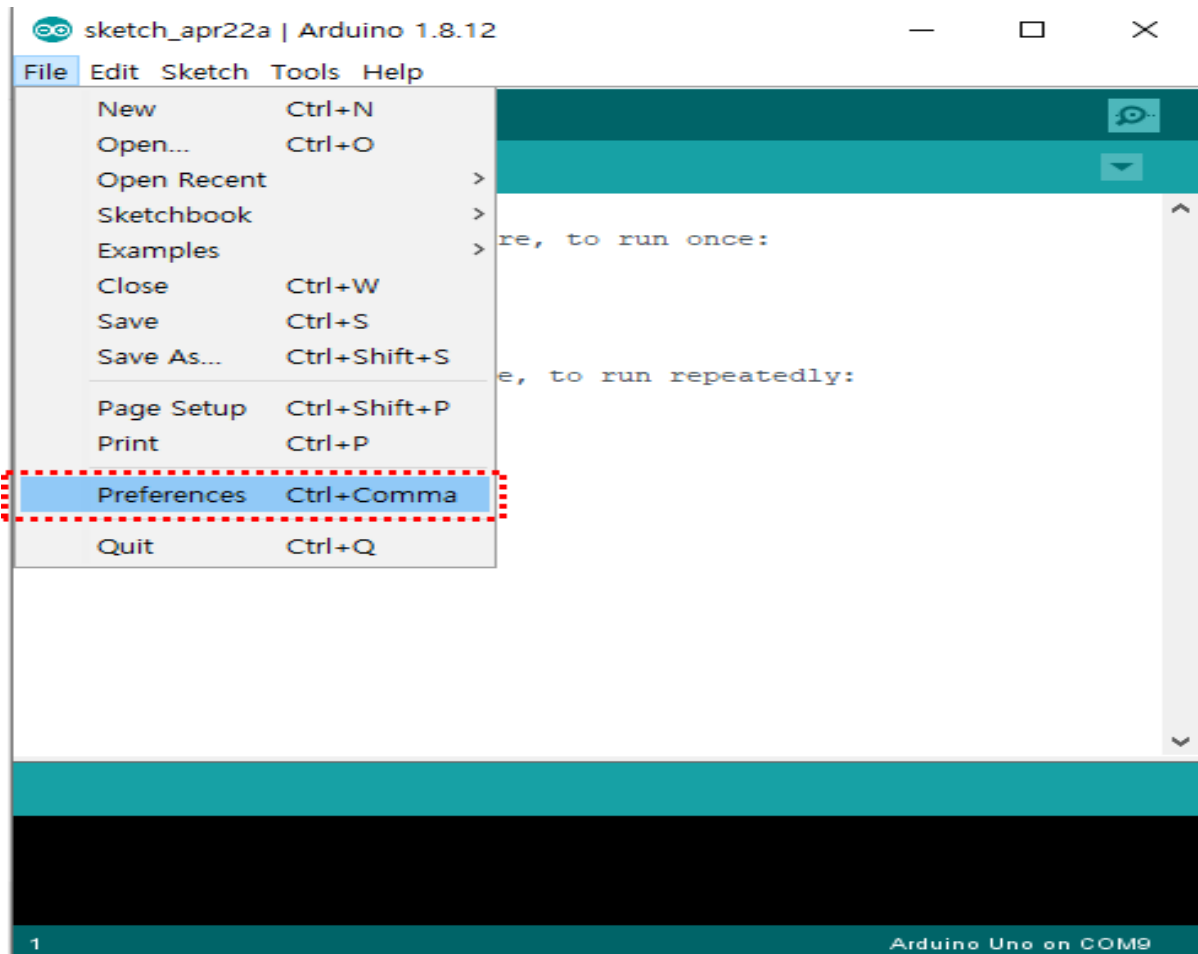
ESP8266 Wifi 쉴드(shield)

ESP8266 Wifi 쉴드 제품은 ESP8266 Wifi 쉴드과 같이 ESP8266 칩셋을 사용하기 때문에 Wifi 모듈과 동일한 역할을 합니다. 아두이노에는 일반적으로 기본보드에는 와이파이 칩이 없기 때문에 와이파이 기능이 되려면 Wifi 쉴드를 사용해야 합니다.

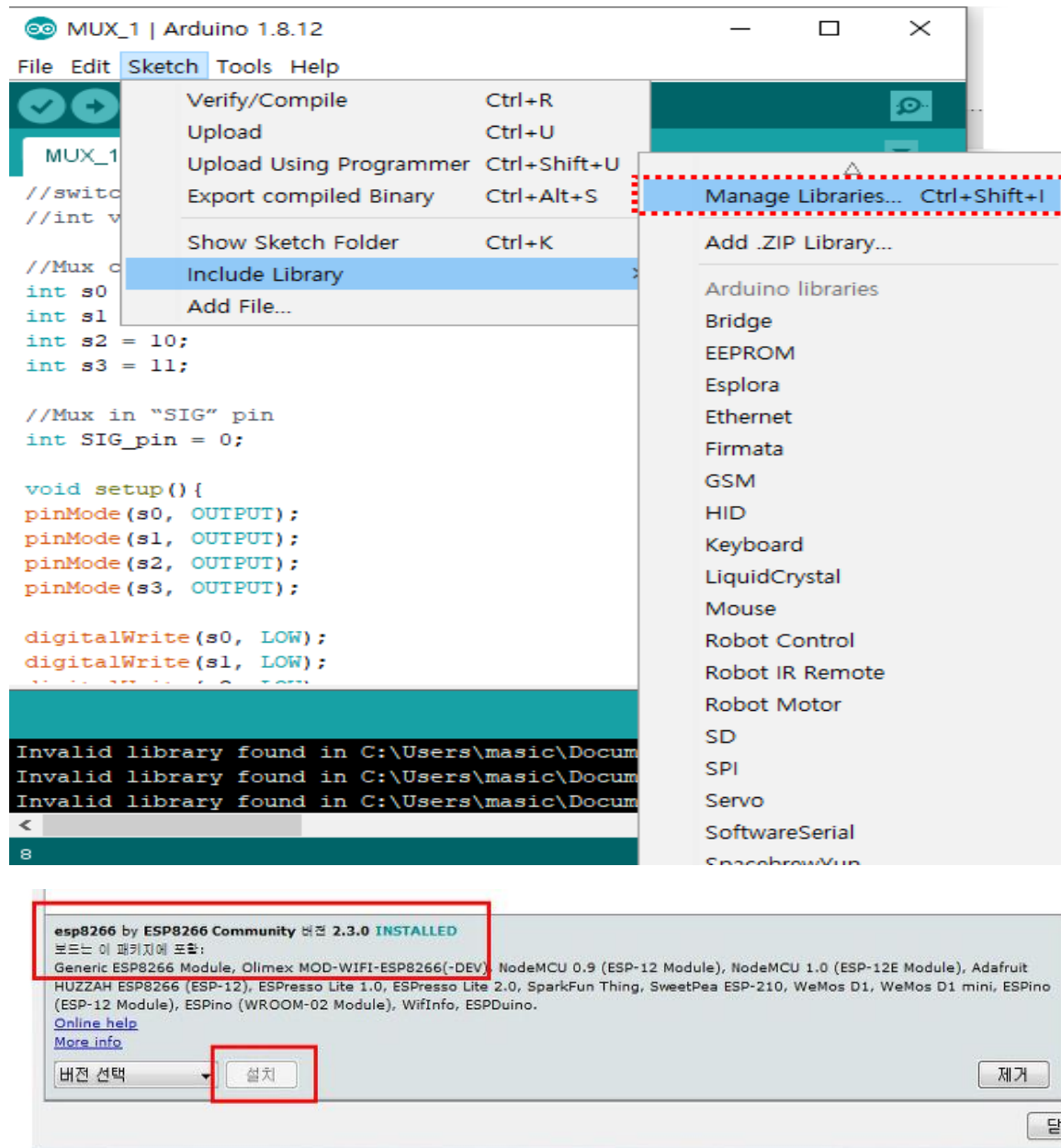
ESP8266 Wifi 쉴드 환경구축

1. 스케치 메뉴에서 File(파일) -> preferences(환경설정)에서 Additional Boards ManagerURLs(추가적인보드매니저URLs)에

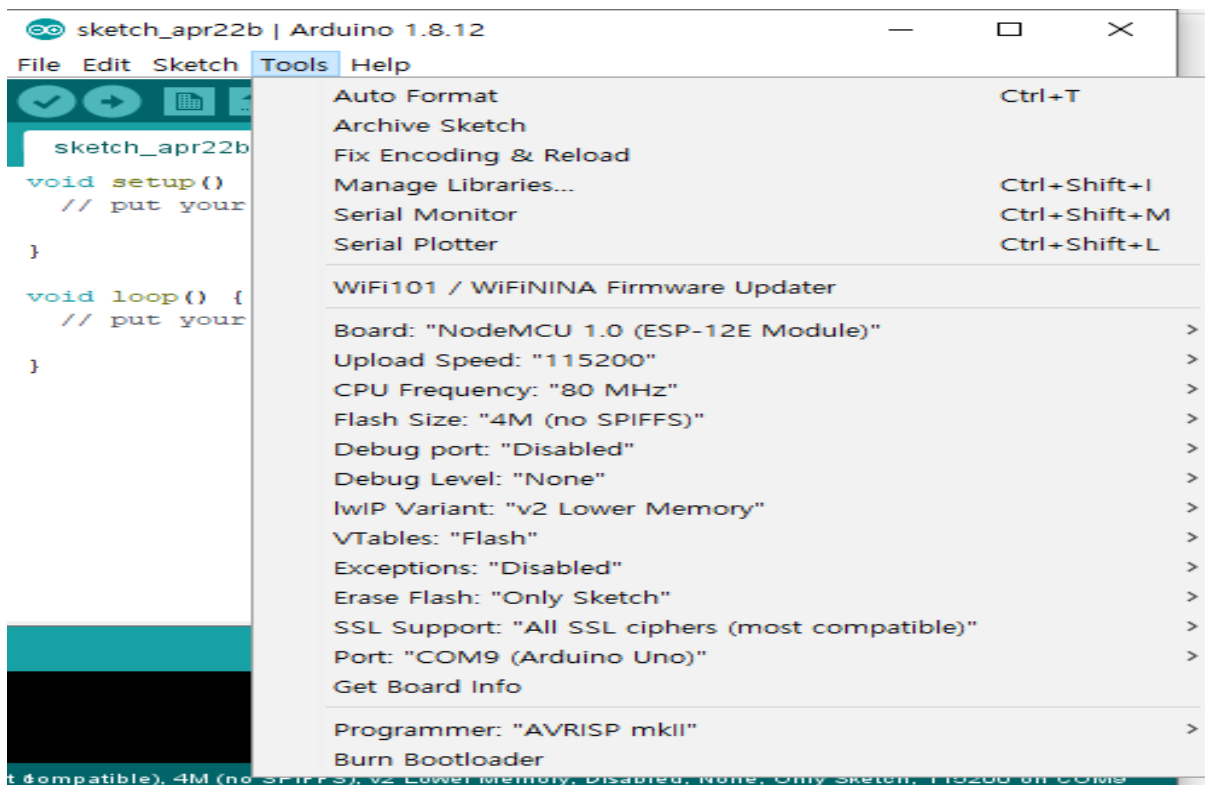
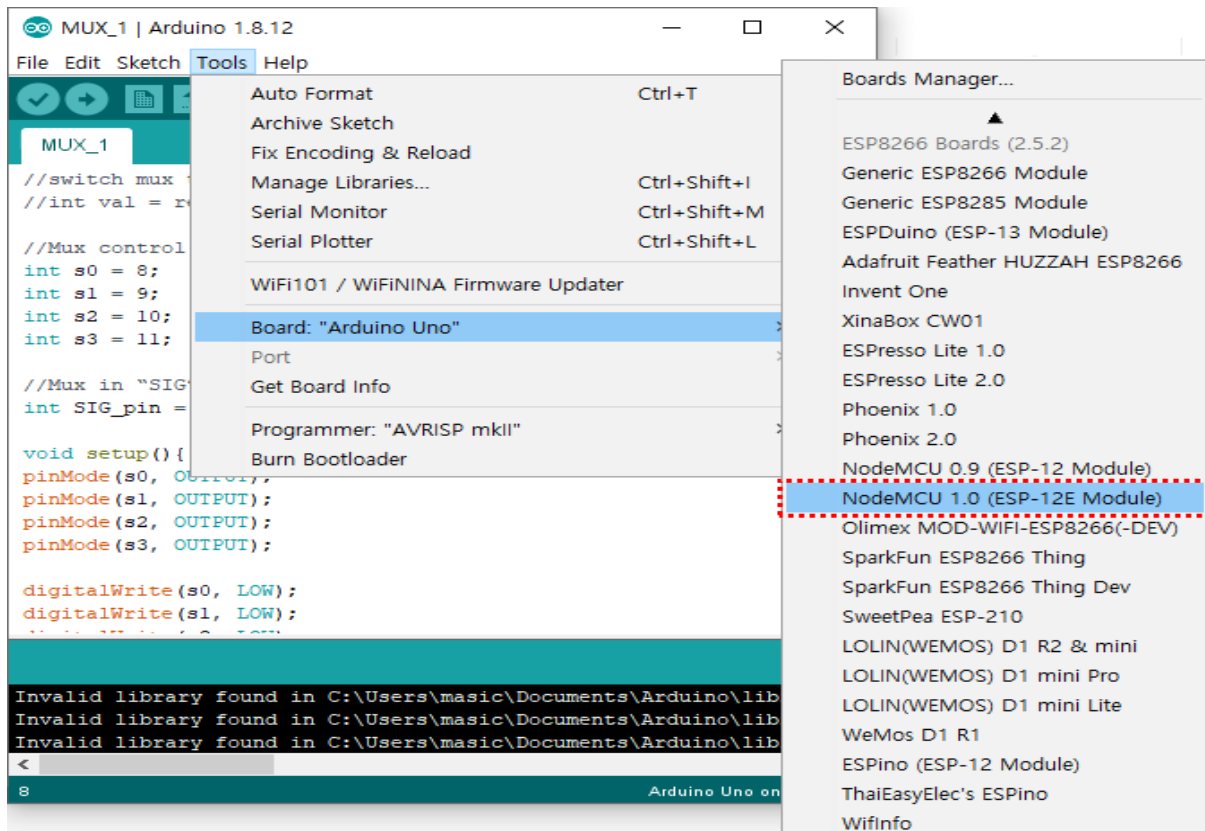
http://arduino.esp8266.com/stable/package_esp8266com_index.json를 추가해줍니다.



2. 라이브러리 포함하기(Include Library) 메뉴의 라이브러리 관리(Manage Libraries)에서 ESP8266 Community의 esp8266 라이브러리를 설치합니다.



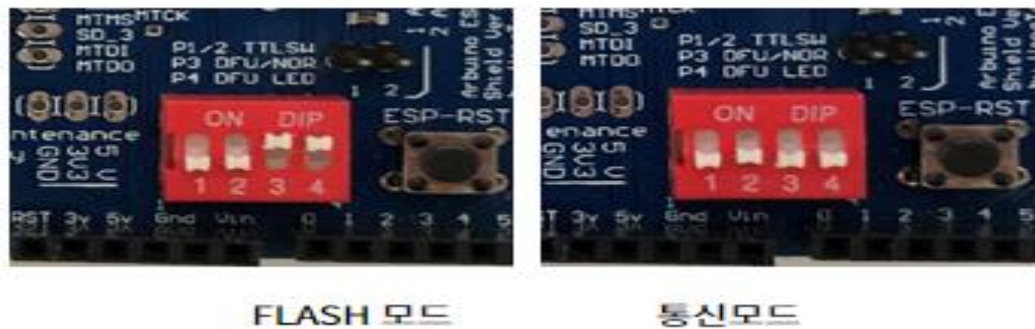
3. esp8266 라이브러리가 정상적으로 설치되고 나면 스케치의 툴(Tool) 메뉴의 보드(Board)에서 NodeMCU 1.0 (ESP-12E Module)을 선택합니다.



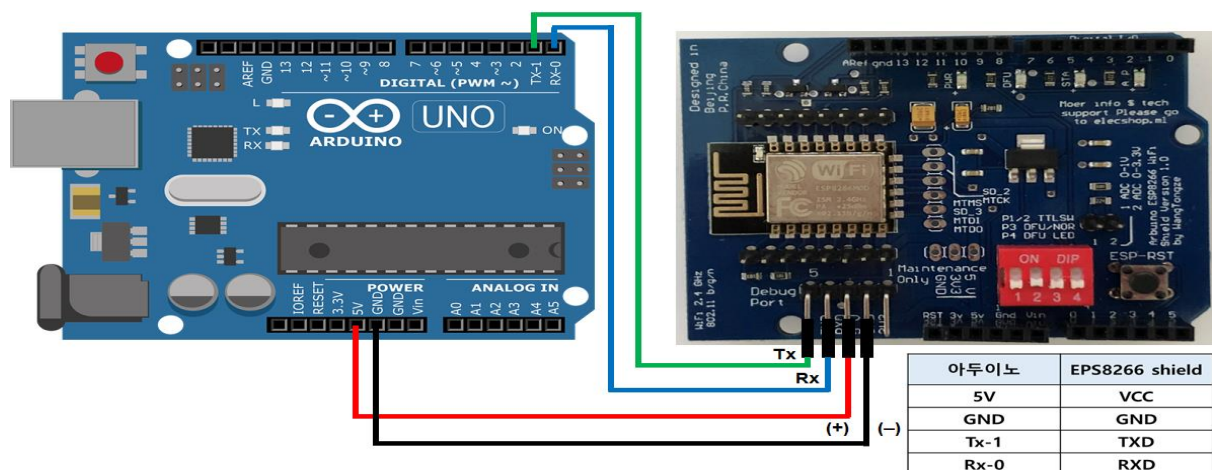
ESP8266 쉴드 보드 설정

스위치 DIP 업 : FLASH모드 (펌웨어 업로드 및 스케치 업로드 시 사용)

스위치 DIP 다운 : 통신모드(TX,RX 통신 및 아두이노 시리얼 통신 시 사용)



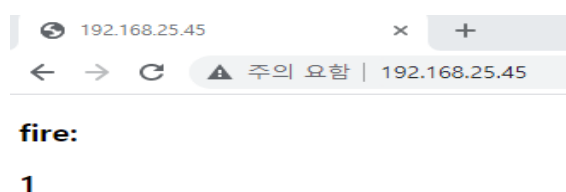
아두이노 보드와 ESP8266 쉴드 연결 방법



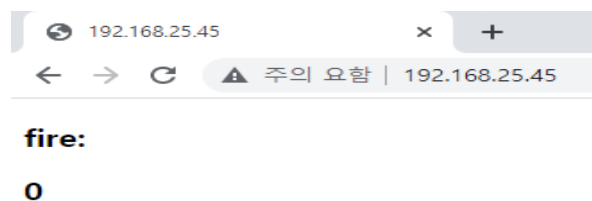
스케치를 통해 프로그램 업로드

1. FASH모드로 설정하고 스케치에서 업로드 실행
2. 그리고 타이밍 맞춰서 ESPRST 버튼 누르기

Wifi 쉴드를 통해 서버로 값을 넘겨줄 수 있다.(JSP, PHP, firebase 등)



원래 상태 1

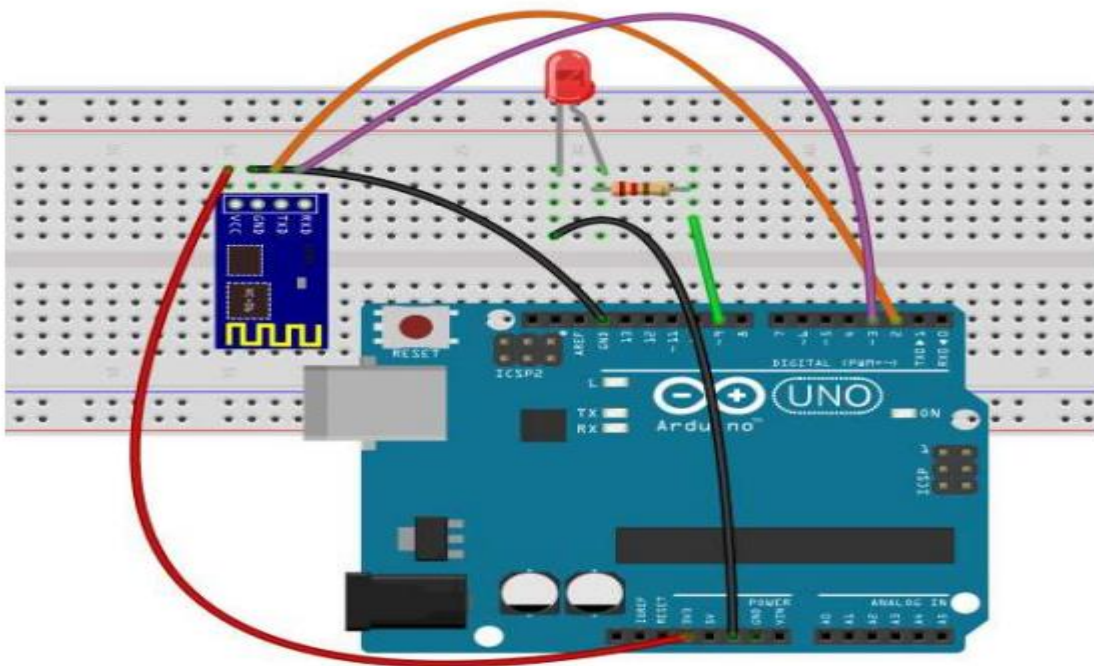


감지 될 경우 0 으로 바뀜

아두이노 블루투스

1. 대표적인 근거리 통신장치의 한 종류로 복잡한 통신장비없이 쉽게 송 수신을 할 수 있다.
2. 마스터/슬레이브 구조로 구성되며 하나의 마스터가 최대 7개까지 슬레이브 장치들과 연결될 수 있다.
3. 페어링을 시켜 두개의 기기를 하나의 쌍으로 만들어 주어야 하며, 여러 개의 기기를 동시에 사용할 수 없다.

블루투스 배선도

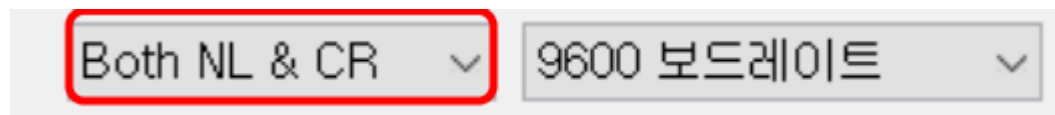


블루투스 설정

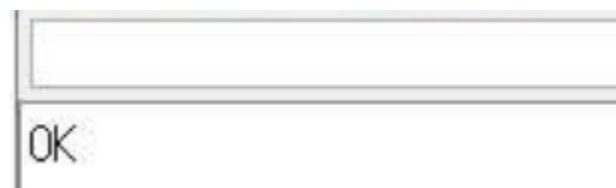
핸드폰에서 블루투스 등록 요청 -> 블루투스 페어링 등록요청(1234 or 0000) -> 연결 확인 후 이름변경

블루투스 연결 확인 및 이름 바꾸는 법

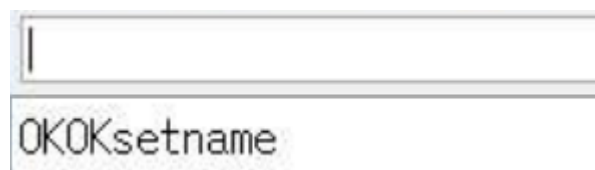
```
#include <SoftwareSerial.h>
SoftwareSerial BT_Serial(2,3);//TX,RX
void setup()
{
  Serial.begin(9600);//마두미노<-> PC통신
  BT_Serial.begin(9600);// 블루투스 <-> 마두미노통신
}
void loop()
{
  if(BT_Serial.available()) //블루투스에서 데이터가 들어오면
    Serial.write(BT_Serial.read());//읽은데이터 PC로 전송
  if(Serial.available()) //PC에서 데이터가 들어오면
    BT_Serial.write(Serial.read());//읽은데이터 블루투스로 전송
}
```



AT 입력 후 전송 연결이 되면 OK출력



AT+NAME(이름) 이름 변경 완료



블루투스를 통한 아두이노

안드로이드 연동

```
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
```

안드로이드에서 블루투스 기능을 사용하기 위해서 권한이 필요하다.

```
public class BluetoothActivity extends AppCompatActivity {

    TextView mTvBluetoothStatus;
    TextView mTvReceiveData;
    TextView mTvSendData;
    Button mBtnBluetoothOn;
    Button mBtnBluetoothOff;
    Button mBtnConnect;
    Button mBtnSendData;

    BluetoothAdapter mBluetoothAdapter;
    Set<BluetoothDevice> mPairedDevices;
    List<String> mListPairedDevices;

    Handler mBluetoothHandler;
    ConnectedBluetoothThread mThreadConnectedBluetooth;
    BluetoothDevice mBluetoothDevice;
    BluetoothSocket mBluetoothSocket;
    final static int BT_REQUEST_ENABLE = 1;

    final static int BT_MESSAGE_READ = 2;

    final static int BT_CONNECTING_STATUS = 3;

    final static UUID BT_UUID = UUID.fromString("00001101-0000-1000-8000-00805f9b34fb");
```

전역변수, 객체, 상수를 선언하는 부분

```
@Override
protected void onCreate(Bundle savedInstanceState) {

    super.onCreate(savedInstanceState);

    setContentView(R.layout.activity_bluetooth);

    mTvBluetoothStatus = (TextView)findViewById(R.id.tvBluetoothStatus);
    mTvReceiveData = (TextView)findViewById(R.id.tvReceiveData);
    mBtnBluetoothOn = (Button)findViewById(R.id.btnBluetoothOn);
    mBtnBluetoothOff = (Button)findViewById(R.id.btnBluetoothOff);
    mBtnConnect = (Button)findViewById(R.id.btnConnect);
```

전역으로 선언한 버튼, 텍스트 뷰를 참조시키기.


```

mBtnBluetoothOn.setOnClickListener(new Button.OnClickListener() {
    @Override
    public void onClick(View view) {
        bluetoothOn();
    }
});

mBtnBluetoothOff.setOnClickListener(new Button.OnClickListener() {
    @Override
    public void onClick(View view) {
        bluetoothOff();
    }
});

mBtnConnect.setOnClickListener(new Button.OnClickListener() {
    @Override
    public void onClick(View view) {
        listPairedDevices();
    }
});

```

버튼이 클릭되었을 때 발생하는 이벤트 리스너

```

void bluetoothOn() {
    if(mBluetoothAdapter == null) {
        Toast.makeText(getApplicationContext(), "블루투스를 지원하지 않는 기기입니다.", Toast.LENGTH_LONG).show();
    }
    else {
        if (mBluetoothAdapter.isEnabled()) {
            Toast.makeText(getApplicationContext(), "블루투스가 이미 활성화 되어 있습니다.", Toast.LENGTH_LONG).show();
            mTvBluetoothStatus.setText("활성화");
        }
        else {
            Toast.makeText(getApplicationContext(), "블루투스가 활성화 되어 있지 않습니다.", Toast.LENGTH_LONG).show();
            Intent intentBluetoothEnable = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
            startActivityForResult(intentBluetoothEnable, BT_REQUEST_ENABLE);
        }
    }
}

```

블루투스 활성화 메소드

```

void bluetoothOff() {
    if (mBluetoothAdapter.isEnabled()) {
        mBluetoothAdapter.disable();
        Toast.makeText(getApplicationContext(), "블루투스가 비활성화 되었습니다.", Toast.LENGTH_SHORT).show();
        mTvBluetoothStatus.setText("비활성화");
    }
    else {
        Toast.makeText(getApplicationContext(), "블루투스가 이미 비활성화 되어 있습니다.", Toast.LENGTH_SHORT).show();
    }
}

```

비활성화 메소드

```

private class ConnectedBluetoothThread extends Thread {
    private final BluetoothSocket mmSocket;
    private final InputStream mmInStream;
    private final OutputStream mmOutStream;

    public ConnectedBluetoothThread(BluetoothSocket socket) {
        mmSocket = socket;
        InputStream tmpIn = null;
        OutputStream tmpOut = null;

        try {
            tmpIn = socket.getInputStream();
            tmpOut = socket.getOutputStream();
        } catch (IOException e) {
            Toast.makeText(getApplicationContext(), "소켓 연결 중 오류가 발생했습니다.", Toast.LENGTH_LONG).show();
        }

        mmInStream = tmpIn;
        mmOutStream = tmpOut;
    }
}

```

사용할 전역 객체들 선언 과 소켓을 통한 전송처리 가져오기(데이터 전송 및 수신하는 길 만드는 작업)

```

// 센서값 받아오는 부분
public void run() {
    byte[] buffer = new byte[1024];
    int bytes;
    while (true) { //while 반복문 처리로 데이터가
        try {
            bytes = mmInStream.available();
            if (bytes != 0) {
                SystemClock.sleep(100);
                bytes = mmInStream.available();
                bytes = mmInStream.read(buffer, 0, bytes);
                mBluetoothHandler.obtainMessage(BT_MESSAGE_READ, bytes, -1, buffer).sendToTarget();
            }
        } catch (IOException e) {
            break;
        }
    }
}
}

```

수신받은 데이터는 항상 확인해야함 그에 따라 while 반복문 처리로 데이터가 존재한다면 데이터를 읽어오는 작업을 해줌.

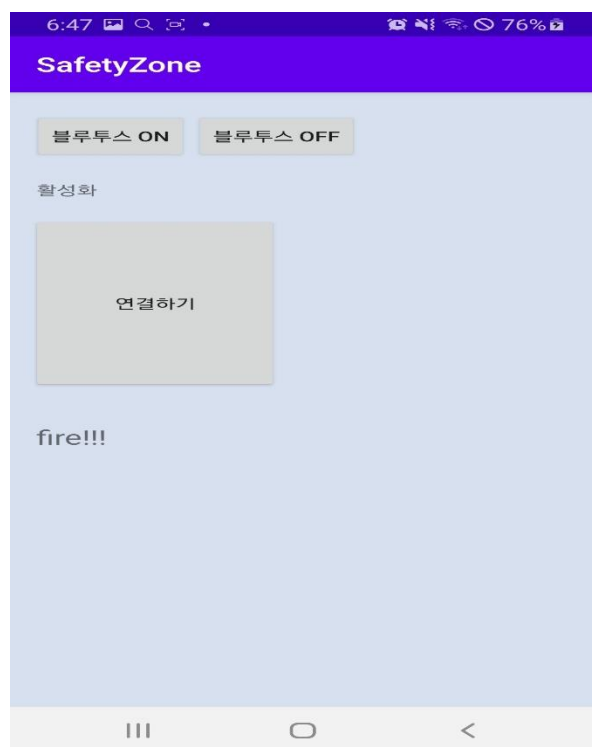
아두이노 블루투스과 화염감지 센서를 이용해서 안드로이드에 글씨 출력

불이 감지가되면 안드로이드 화면에 Fire!!!이라는 글이 출력

```
#include <SoftwareSerial.h>

SoftwareSerial bt(3, 2); //Tx,Rx 핀번호 설정
int flame = 10;         //화염감지센서 핀번호
int co = 1;
void setup() {
  pinMode(flame, INPUT);
  Serial.begin(9600);    // 아두이노 <> PC 통신
  bt.begin(9600);        // 블루투스 <> 아두이노 통신
}
void loop() {
  if(bt.available()) {   // 블루투스에서 데이터가 들어오면
    Serial.write(bt.read()); // 읽은데이터를 PC로 전송
  }
  if(Serial.available()) { // PC에서 데이터가 들어오면
    bt.write(Serial.read()); // 읽은데이터를 블루투스로 전송
  }
  if(digitalRead(flame) == 0) { // 불이 감지되면
    bt.print(" fire!!!"); // 안드로이드로 센서값 출력
    delay(10000);
    co=0;
  }

  if(co==0) {
    bt.print(" no fire");
    co=1;
  }
}
```



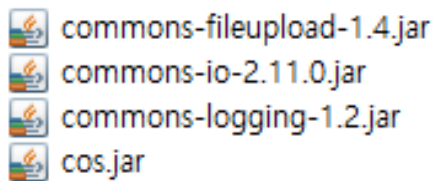
화염감지 센서에서 감지가 되면 fire로 바뀜.

안드로이드 서버 연동





이미지 업로드

환경 설정하기

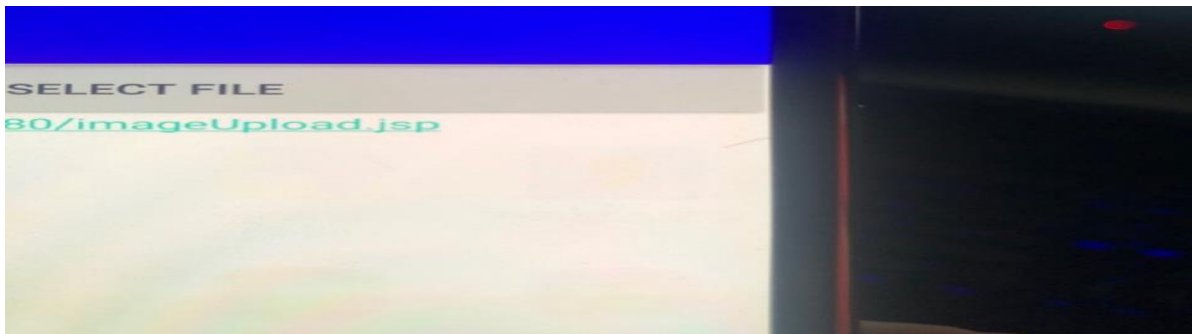
1. 필요한 라이브러리



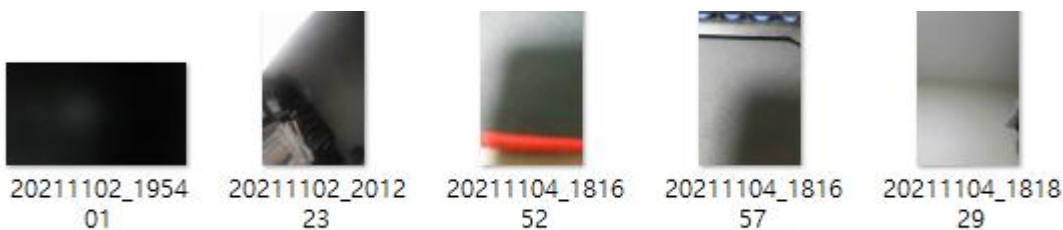
2. Uplad 파일 생성

 META-INF	2021-11-01 오후 10:34	파일 폴더	
 upload	2021-11-04 오후 9:46	파일 폴더	
 WEB-INF	2021-11-03 오후 10:23	파일 폴더	
 imageUpload	2021-11-03 오후 11:17	JSP 파일	4KB

3. 핸드폰으로 사진올리면



4. Upload 파일안에 사진이 들어감



블루투스 소스코드 중요 부분

```
implementation 'com.squareup.retrofit2:retrofit:2.5.0'  
implementation 'com.squareup.retrofit2:converter-gson:2.4.0'  
implementation 'com.squareup.okhttp3:okhttp:3.10.0'  
implementation 'com.squareup.okhttp3:logging-interceptor:3.8.1'
```

Gradle에 추가

```
<uses-permission android:name="android.permission.INTERNET"/>  
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>  
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
```

androidManifest에 권한 추가

```
package com.example.file;  
import okhttp3.OkHttpClient;  
import okhttp3.logging.HttpLoggingInterceptor;  
import retrofit2.Retrofit;  
import retrofit2.converter.gson.GsonConverterFactory;  
  
public class MyRetrofit2 {  
  
    public static final String URL = "http://192.168.0.100:8080/";  
  
    static Retrofit mRetrofit;  
  
    public static Retrofit getRetrofit2(){  
  
        if(mRetrofit == null){  
  
            HttpLoggingInterceptor logging = new HttpLoggingInterceptor();  
            logging.setLevel(HttpLoggingInterceptor.Level.BODY);  
            OkHttpClient.Builder httpClient = new OkHttpClient.Builder();  
            httpClient.addInterceptor(logging);  
  
            mRetrofit = new Retrofit.Builder()  
                .baseUrl(URL)  
                .addConverterFactory(GsonConverterFactory.create())  
                .client(httpClient.build())  
                .build();  
        }  
        return mRetrofit;  
    }  
}
```

URL 부분에 각자 주소 입력

```
public void uploadImage(Uri uri){

    UploadService service = MyRetrofit2.getRetrofit2().create(UploadService.class);

    File file = new File(getRealPathFromURI(uri));
    MultipartBody.Part body1 = prepareFilePart("image", uri);

    RequestBody description = createPartFromString("hello, this is description speaking");

    Call<ResponseBody> call = service.uploadFile(description, body1);
    tv_message.setText(call.request().url().toString()); //todo 디버깅용

    call.enqueue(new Callback<ResponseBody>() {
        @Override
        public void onResponse(Call<ResponseBody> call, Response<ResponseBody> response) {
        }

        @Override
        public void onFailure(Call<ResponseBody> call, Throwable t) {
        }
    });
}
```

이미지 업로드

JSP 소스 코드 중요부분

```
if ( !fi.isFormField () ){

    // 업로드된 파일 매개 변수 가져오기
    String fieldName = fi.getFieldName();
    System.out.println("@@ fieldName : "+fieldName);
    String fileName = fi.getName();
    boolean isInMemory = fi.isInMemory();
    long sizeInBytes = fi.getSize();
    // 파일 쓰기
    if( fileName.lastIndexOf("\\") >= 0 ){
        System.out.println("111111111");
        file = new File( filePath +
            fileName.substring( fileName.lastIndexOf("\\"))) ;
    }else{
        file = new File( filePath +
            fileName.substring(fileName.lastIndexOf("\\")+1)) ;
    }
    fi.write( file ) ;
    out.println("Uploaded Filename: " + filePath +
        fileName + "<br>");
} else{
    System.out.println("bbbbbbb");
}
```