

정규 교육 세미나

ToBig's 10기 정윤호

# Git & Github

# Content

---

Unit 01 | Git & Github 소개

---

Unit 02 | Git & Github 개념

---

Unit 03 | Git 사용

---

Unit 04 | Github 사용

---

Unit 05 | 과제

---

## Unit 01 | Git & Github 소개

분산 버전 관리  
시스템



**git** = Version Control System 중 하나

## Unit 01 | Git & Github 소개

**버전 관리(Version Control)란 대체 무엇일까?**

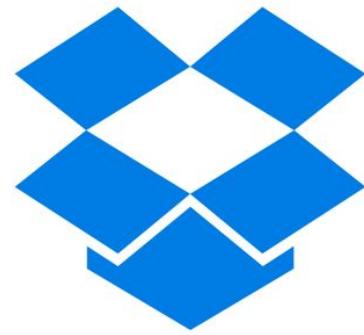
## Unit 01 | Git & Github 소개

우린 사실 버전 관리를 해오고 있었다...!



Google Drive

Bodiflora



Dropbox



## Unit 01 | Git & Github 소개

파이널 !



**models\_final.py**

## Unit 01 | Git & Github 소개

파이널 !



models\_final.py

이게 파이널...!



models\_final2.py

# Unit 01 | Git & Github 소개

파이널 !



models\_final.py

이게 파이널...!



models\_final2.py

ㄹㅇ 파이널....



models\_real\_final.py

# Unit 01 | Git & Github 소개

파이널 !



models\_final.py

이게 파이널...!



models\_final2.py

ㄹㅇ 파이널....



models\_real\_final.py

오늘은 여기까지...



models\_0116final.py

## Unit 01 | Git & Github 소개

파이널 !      이게 파이널...      르파이널....      오늘은  
**파일에 담겨있는 내용을  
여러 개의 버전으로 보관하는 행위를  
이미 하고 있었다 !**

models\_final.py

models\_final2.py

models\_real\_final.py

models\_0116final.py

## Unit 01 | Git & Github 소개

파이널 !

이게 파이널 !

로운 파이널....

오늘은  
여기까지...

파일 이름은 그대로 두고  
버전관리는 컴퓨터가 하게 하자



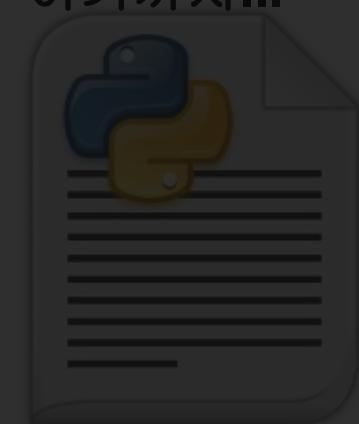
models\_final.py



models\_final2.py



models\_real\_final.py



models\_0116final.py

## Unit 01 | Git & Github 소개



- **Managing version**
- **Backup**
- **Recovery**
- **Collaboration**

## Unit 01 | Git & Github 소개

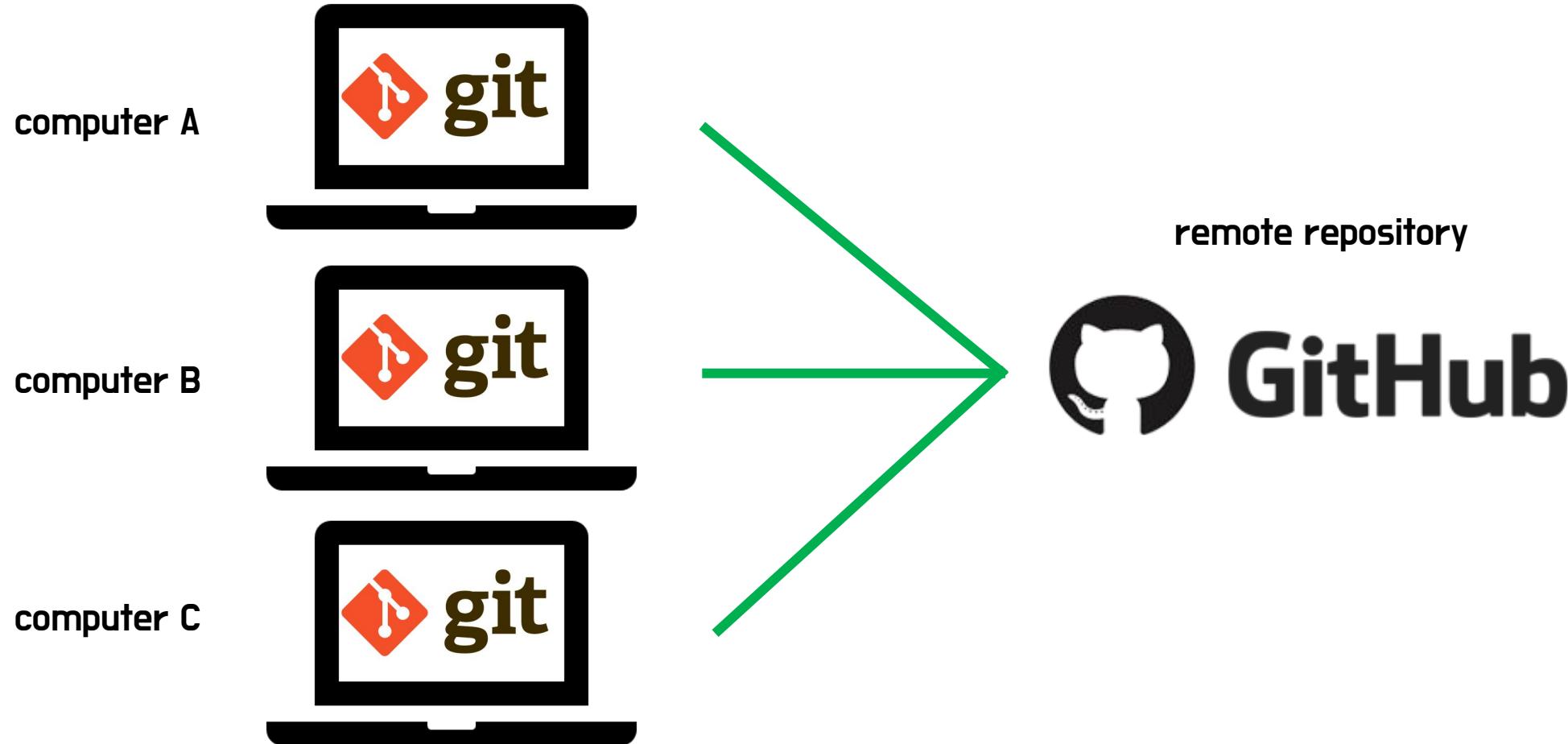


깃허브  
깃허브



깃허브  
깃허브

## Unit 01 | Git & Github 소개



## Unit 01 | Git & Github 소개

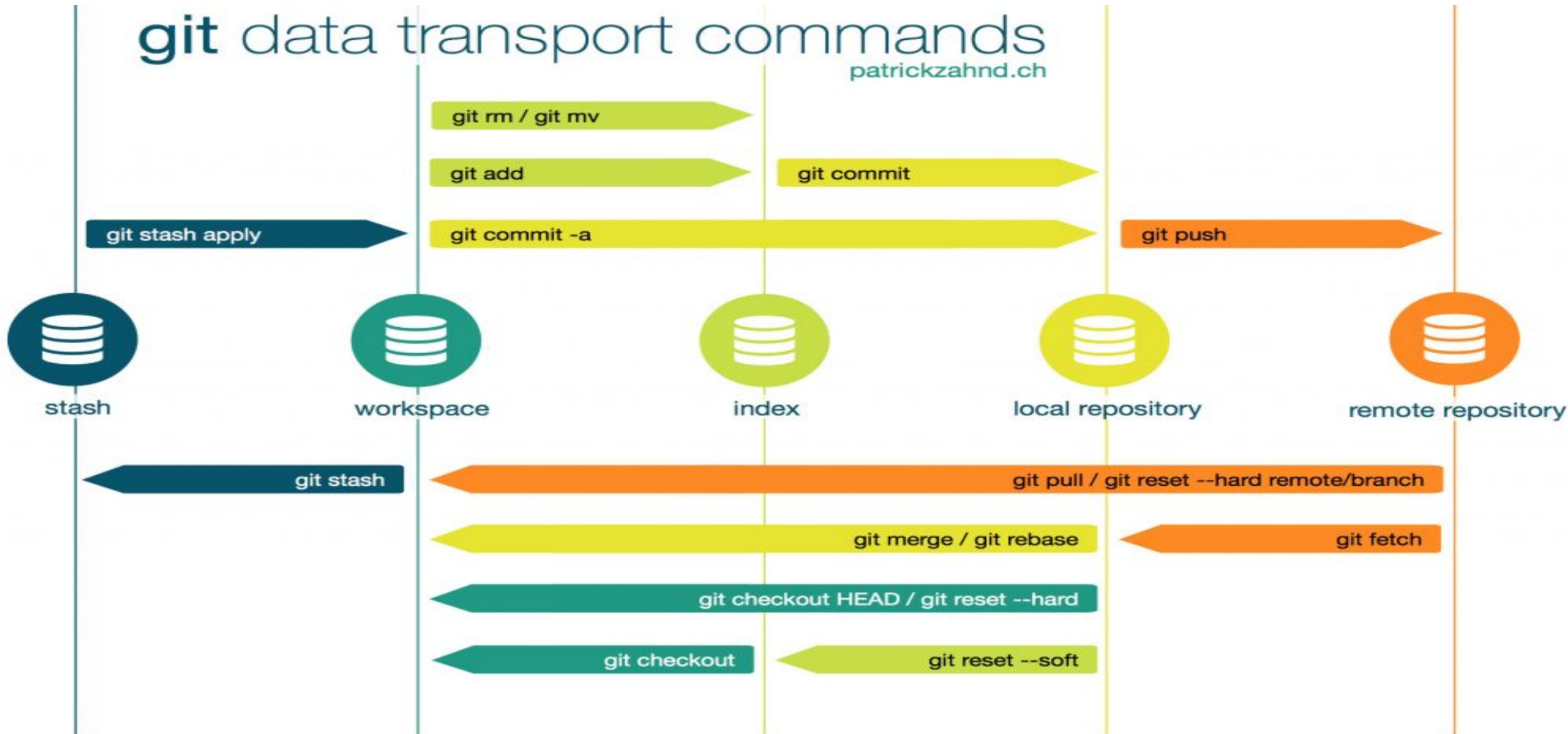
# Summary

- **Git is version management tool using local repository**
- **Github is remote repository for collaboration by multiple git users**

## Unit 02 | Git & Github 개념

### git data transport commands

patrickzahnd.ch



## Unit 02 | Git & Github 개념



**Working Directory**

**Local Repository**



**Remote Repository**



## Unit 02 | Git & Github 개념

### Working Directory

내 PC안의  
작업공간들 중  
git을 사용하는  
작업공간



임시 버전들이  
올라가는 공간

### Local Repository

최종 확정본이  
올라가는 공간

## Unit 02 | Git & Github 개념

### Working Directory

→  
**init**



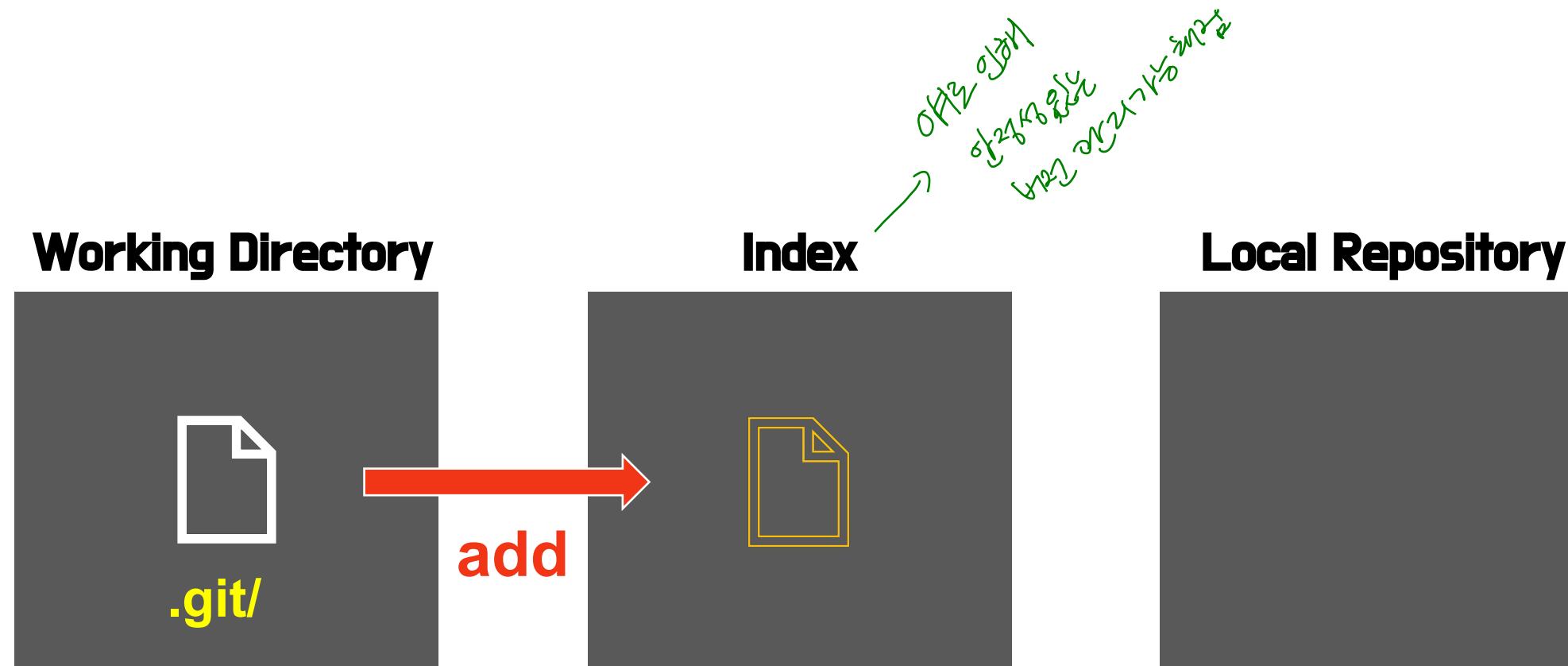
### Index



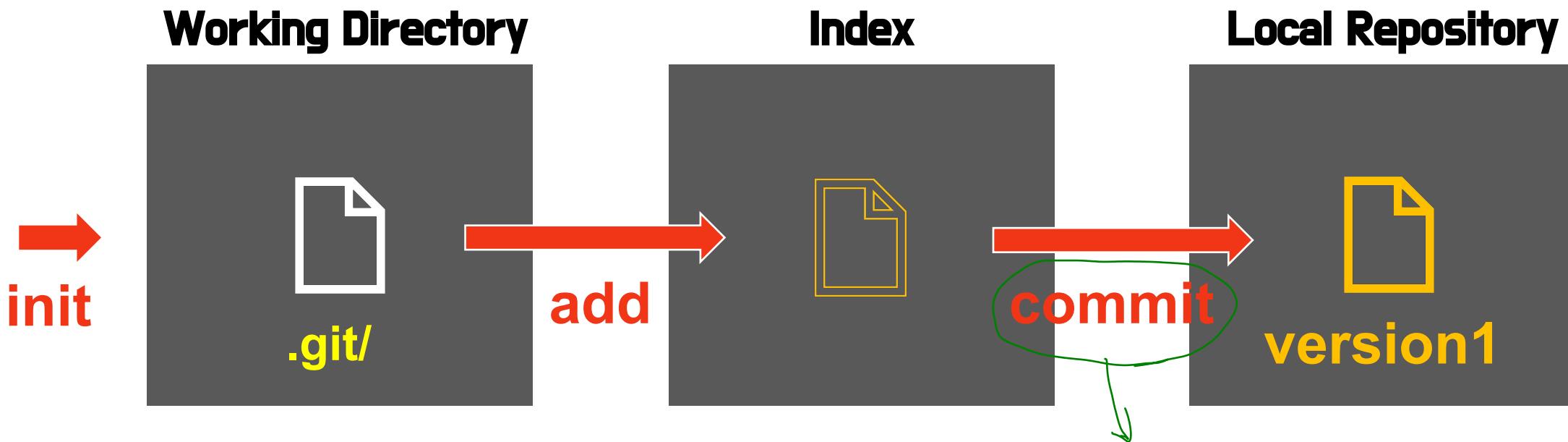
### Local Repository



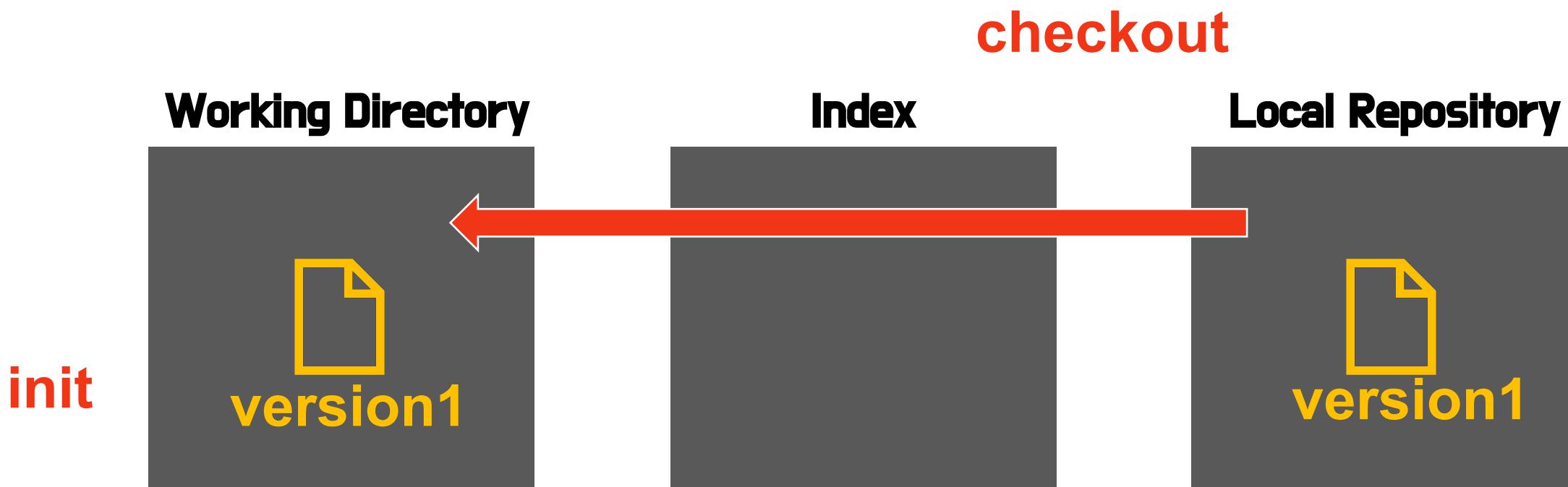
## Unit 02 | Git & Github 개념



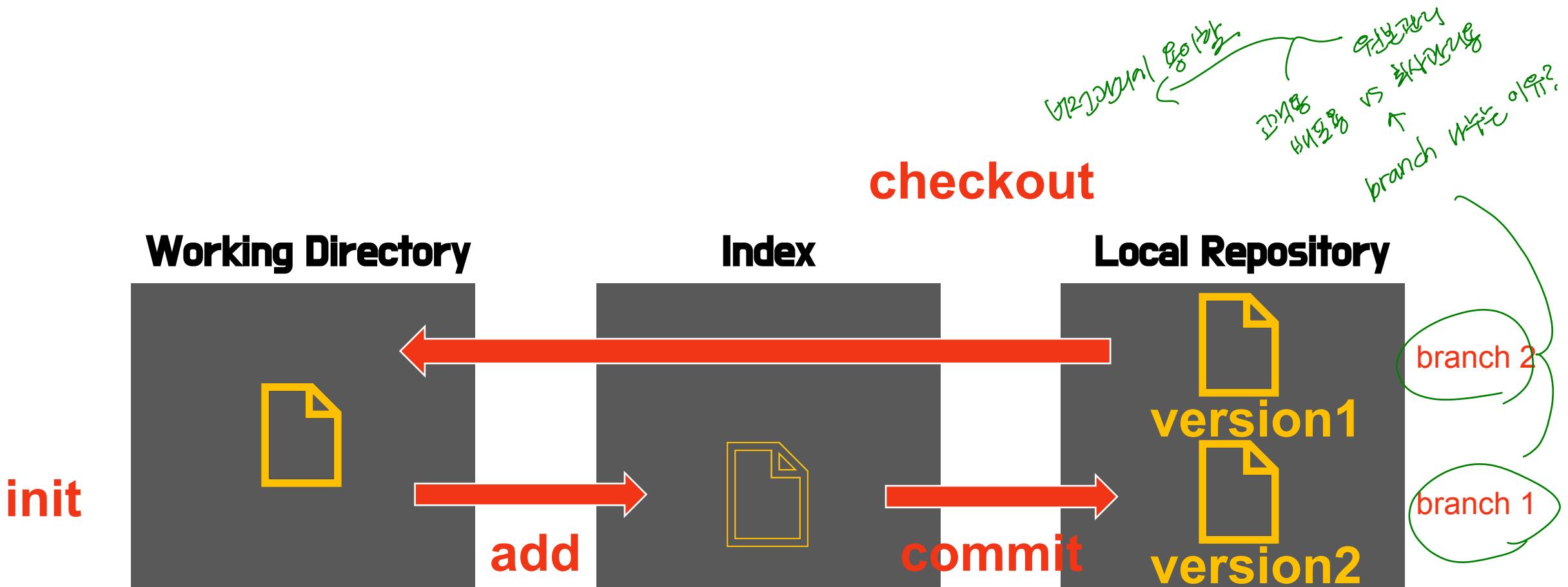
## Unit 02 | Git & Github 개념



## Unit 02 | Git & Github 개념



## Unit 02 | Git & Github 개념



## Unit 02 | Git & Github 개념

### Local repository

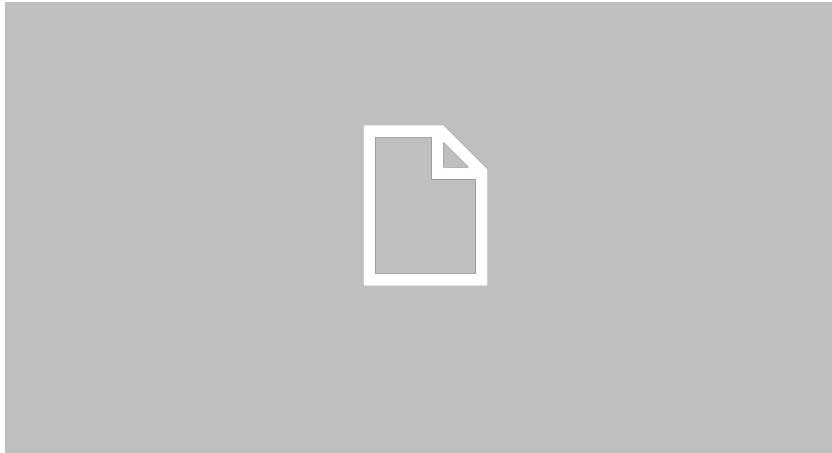
개인 PC에 파일이 저장되는  
개인 전용 저장소

### Remote repository

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

## Unit 02 | Git & Github 개념

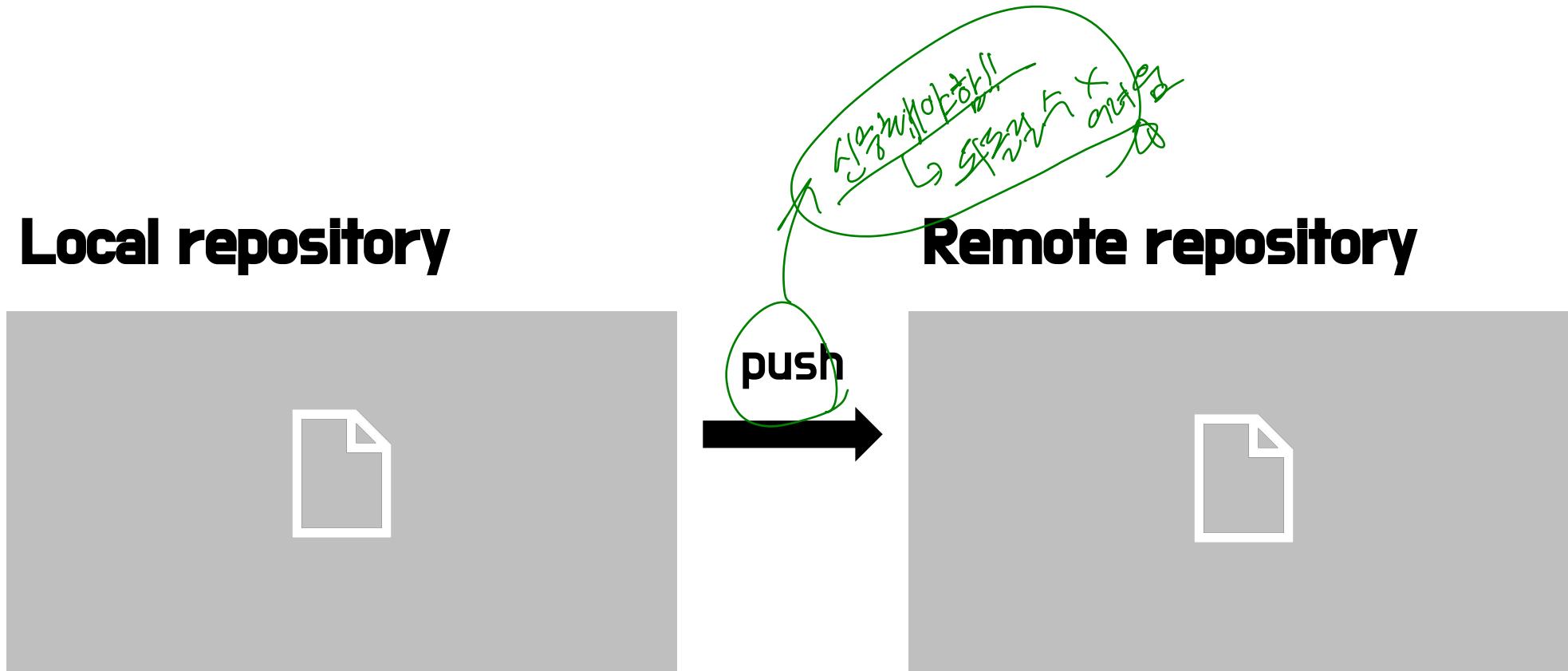
### Local repository



### Remote repository

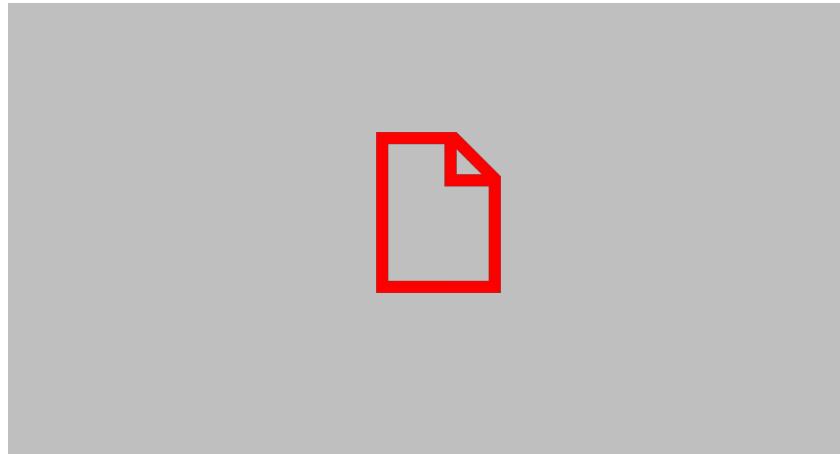


## Unit 02 | Git & Github 개념

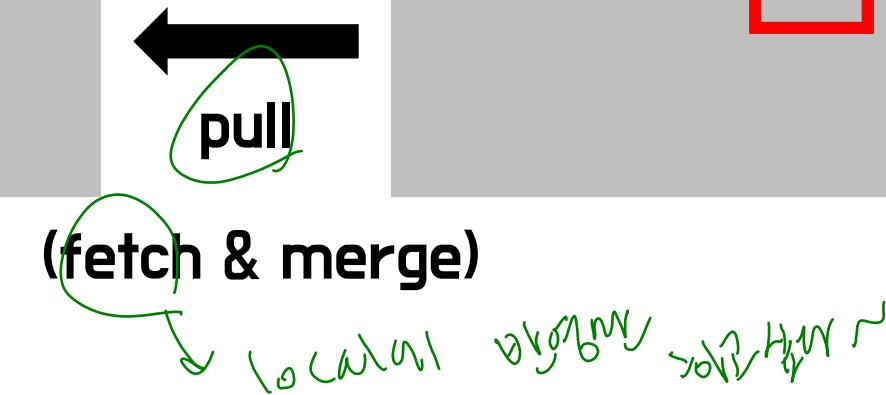
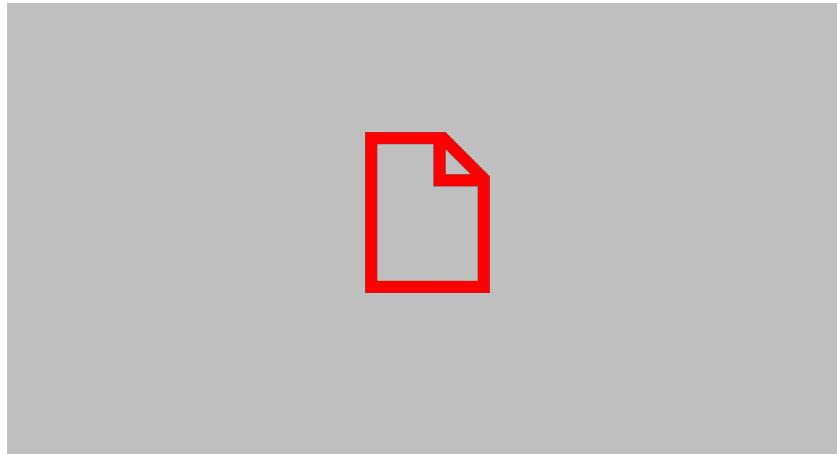


## Unit 02 | Git & Github 개념

### Local repository



### Remote repository



## Unit 02 | Git & Github 개념

- git init : 현재 디렉토리에 git을 적용하겠다고 git에게 알림(start working area)
- git add : git에게 해당 파일을 버전 관리해라고 알림(staging area에 등록)
- git commit : 하나의 버전을 저장소 히스토리에 저장
- git push : 로컬 저장소에 있는 내용들을 원격 저장소에 저장
- git branch "branch\_name" : "branch\_name"의 브랜치를 생성
- git checkout "branch\_name" : "branch\_name"으로 이동
- git checkout -- "file\_name" : 현 브랜치에서 지웠던 파일 되살리기
- git merge "branch\_name" : 현재 브랜치에서 "branch\_name"을 merge
- git fetch : 원격 저장소에 있는 내용을 로컬 저장소에 반영
- git pull : 원격 저장소에 있는 내용을 로컬 저장소로 반영하고 브랜치를 병합함으로써 업데이트
- git log : 커밋 로그 상태를 확인
- git diff : 변경 사항 확인
- git reset "commit\_ID" --hard : 해당 "commit\_ID"의 상태로 돌아감
- git revert "commit\_ID" : 해당 "commit\_ID" 상태로 돌아가면서 새로운 버전으로 바꿈
- git "명령어" --help : 해당 명령어에 대한 설명, 옵션
- .git/ : 여러 파일들이 존재하는데 여기에 모든 로그 정보들이 담겨있고 이 정보들을 가지고 git이 실행됨
- index : 커밋대기상태(staging area)

## Unit 02 | Git & Github 개념

### 참고) 간단한 bash 명령어 정리

linux 기반 명령어

mkdir : 디렉토리를 만듦

ls : 현재 디렉토리의 list

ls : 파일명만 보여줌

ls -a : 디렉토리 내 모든 것 (ex. 폴더, .. )

ls -al : 디렉토리 내 모든 것 + 접근권한 + 생성날짜 등

pwd : 현재 위치

cd : 디렉토리 변경

cd : 최상위 폴더로 이동

cd dir\_name : dir 로 이동

vim file\_name : 파일 생성, 편집

file\_name 이 현재 dir에 없는 경우 생성

file\_name 이 현재 dir에 있는 경우 편집

( 편집 방법들

i - 삽입, :w - 저장,

:q - 나가기, :wq - 저장 후 나가기 )

cat file\_name : 파일의 내용 출력

## Unit 03 | Git 사용

# Git 실습

## Unit 03 | Git 사용

### **working directory 생성 방법**

- **git init**
- **git clone**

## Unit 03 | Git 사용

### working directory 생성 방법

- **git init**
- **git clone**

## Unit 03 | Git 사용

**git init/git add/git commit**

## Unit 03 | Git 사용

### git init

```
→ ~ mkdir tobigs_git
→ ~ cd tobigs_git
→ tobigs_git git init
Initialized empty Git repository in /Users/yunho/tobigs_git/.git/
→ tobigs_git git:(master) ls -al
total 0
drwxr-xr-x    3 yunho  staff   96  1 16 11:49 .
drwxr-xr-x+ 107 yunho  staff  3424  1 16 11:50 ..
drwxr-xr-x    10 yunho  staff   320  1 16 11:49 .git
→ tobigs_git git:(master)
```

## Unit 03 | Git 사용

### a.py 작업

```
→ tobigs_git git:(master) vim a.py → a.py를 만든거임
→ tobigs_git git:(master) ✘ git status ← 201801!
On branch master
No commits yet
Untracked files:
(use "git add <file>..." to include in what will be committed)
a.py → add orign
nothing added to commit but untracked files present (use "git add" to track)
```

↑ 푸시하고 깃헙에

→ 앤디지언 누르면 평점남  
Untrack GCS : wq

```
1 # my name is Yunho Jung
```

## Unit 03 | Git 사용

### git add/git status/git commit

```
→ tobigs_git git:(master) ✘ git add a.py           index\ add
→ tobigs_git git:(master) ✘ git status
On branch master

No commits yet

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

    new file:   a.py → Commit [new work yes/no] ✓

→ tobigs_git git:(master) ✘ git commit -m "first commit"
[master (root-commit) 6ae1357] first commit
  1 file changed, 1 insertion(+)
  create mode 100644 a.py
```

## Unit 03 | Git 사용

### git log

```
commit 6ae1357c9903c99559754c65c0195a55cf0edb2a (HEAD -> master)
Author: Jung YunHo <saijdu2198@gmail.com>
Date:   Wed Jan 16 11:54:57 2019 +0900
```

first commit

(END)

## Unit 03 | Git 사용

**branch 생성과 merge**

## Unit 03 | Git 사용

### branch 생성

```
→ tobigs_git git:(master) git branch test  
→ tobigs_git git:(master) git branch -a  
→ tobigs_git git:(master) █
```

```
* master  
  test  
  (END)
```

## Unit 03 | Git 사용

### test 브랜치로 이동/a.py 수정

```
→ tobigs_git git:(master) git checkout test
Switched to branch 'test'
→ tobigs_git git:(test) vim a.py
→ tobigs_git git:(test) ✘ git add a.py
→ tobigs_git git:(test) ✘ git status
On branch test
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    modified:   a.py
```

→ git branch  
→ git checkout test

```
1 # 20190116 tobigs seminar
2 # my name is Yunho Jung
```

## Unit 03 | Git 사용

### b.py 작업

```
→ tobigs_git git:(test) ✘ vim b.py
→ tobigs_git git:(test) ✘ git status
On branch test
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified:   a.py

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

b.py  
add ok

```
1 def tobigs(s):
2     pass
```

~  
~

## Unit 03 | Git 사용

### git add/git status/ git commit

```
→ tobigs_git git:(test) ✘ git add -A → 파일 전체 add 했을 때 A로 표기  
→ tobigs_git git:(test) ✘ git status
```

On branch test

Changes to be committed:

(use "git reset HEAD <file>..." to unstage)

```
modified:   a.py  
new file:   b.py
```

```
→ tobigs_git git:(test) ✘ git commit -m "edit a.py and make b.py"  
[test 470824f] edit a.py and make b.py  
  2 files changed, 3 insertions(+)  
  create mode 100644 b.py  
→ tobigs_git git:(test) █
```

edit a.py and make b.py  
→ 메시지에 이름 써온 거!

## Unit 03 | Git 사용

**git log branches --decorate --graph --oneline**

(commit  
번호  
생략)

```
* 470824f (HEAD -> test) edit a.py and make b.py
* 6ae1357 (master) first commit
(END)
```

## Unit 03 | Git 사용

### git log master..test

master branch가  
test branch가  
되어 있음을.

```
commit 470824f369432f3ddd14550a81e3b36e43c79f0d (HEAD -> test)
Author: Jung YunHo <saijdu2198@gmail.com>
Date:   Wed Jan 16 12:09:10 2019 +0900
```

edit a.py and make b.py

(END)

## Unit 03 | Git 사용

### master 브랜치로 이동/edit a.py

```
→ tobigs_git git:(test) git checkout master  
Switched to branch 'master'  
→ tobigs_git git:(master) vim a.py  
→ tobigs_git git:(master) ✘ git add a.py  
→ tobigs_git git:(master) ✘ git commit -m "edit a.py"  
[master 2b6560f] edit a.py  
 1 file changed, 1 insertion(+)
```

→ master 브랜치

edit a.py

```
1 # my name is Yunho Jung  
2 # Let's git it !
```

~  
~  
~  
~

## Unit 03 | Git 사용

**git log --branches --decorate --graph --oneline**

branch 목록

```
* 2b6560f (HEAD -> master) edit a.py
| * 470824f (test) edit a.py and make b.py
|/
* 6ae1357 first commit
(END)
```

master 브랜치에 파일 추가  
branch 목록 b.py  
파일

## Unit 03 | Git 사용

merge = 합치기

### master 브랜치에서 test 브랜치 merge

on master  
합친다!

```
→ tobigs_git git:(master) git merge test
```

Auto-merging a.py

Merge made by the 'recursive' strategy.

a.py | 1 +

b.py | 2 ++

2 files changed, 3 insertions(+)

create mode 100644 b.py

```
→ tobigs_git git:(master) git log --branches --decorate --graph --oneline
```

```
* fb0fc59 (HEAD -> master) Merge branch 'test'
```

```
| \
```

```
| * 470824f (test) edit a.py and make b.py
```

```
* | 2b6560f edit a.py
```

```
| /
```

```
* 6ae1357 first commit
```

```
(END)
```

## Unit 03 | Git 사용

항상 merge가 잘 될까...?

충돌도 냄새!

## Unit 03 | Git 사용

### test2 브랜치 생성 후 이동/edit a.py

```
→ tobigs_git git:(master) git checkout -b test2
Switched to a new branch 'test2'
→ tobigs_git git:(test2) vim a.py
→ tobigs_git git:(test2) ✘ git add a.py
→ tobigs_git git:(test2) ✘ git commit -m "edit a.py 2nd line"
[test2 09437d8] edit a.py 2nd line
 1 file changed, 1 insertion(+), 1 deletion(-)
```

test2 브랜치 생성 및 이동

```
1 # 20190116 tobigs seminar
2 # my name is Yunho Jung and I love music
3 # Let's git it !
~ 20190116 투빅스 정규과정
~
```

## Unit 03 | Git 사용

### master 브랜치로 이동/edit a.py

```
→ tobigs_git git:(test2) git checkout master
Switched to branch 'master'
→ tobigs_git git:(master) vim a.py
→ tobigs_git git:(master) ✘ git add a.py
→ tobigs_git git:(master) ✘ git commit -m "edit a.py 2nd line"
[master 6d7b15c] edit a.py 2nd line
 1 file changed, 1 insertion(+), 1 deletion(-)
```

```
1 # 20190116 tobigs seminar
2 # what is your name?
3 # Let's git it !
~
```

## Unit 03 | Git 사용

### CONFLICT 발생 !!

merger  
join 되어야!

```
→ tobigs_git git:(master) git merge test2
Auto-merging a.py
CONFLICT (content): Merge conflict in a.py
Automatic merge failed; fix conflicts and then commit the result.
→ tobigs_git git:(master) x
```

gitbranch 들다 같은부분은 고쳤고  
합치면 상충!!

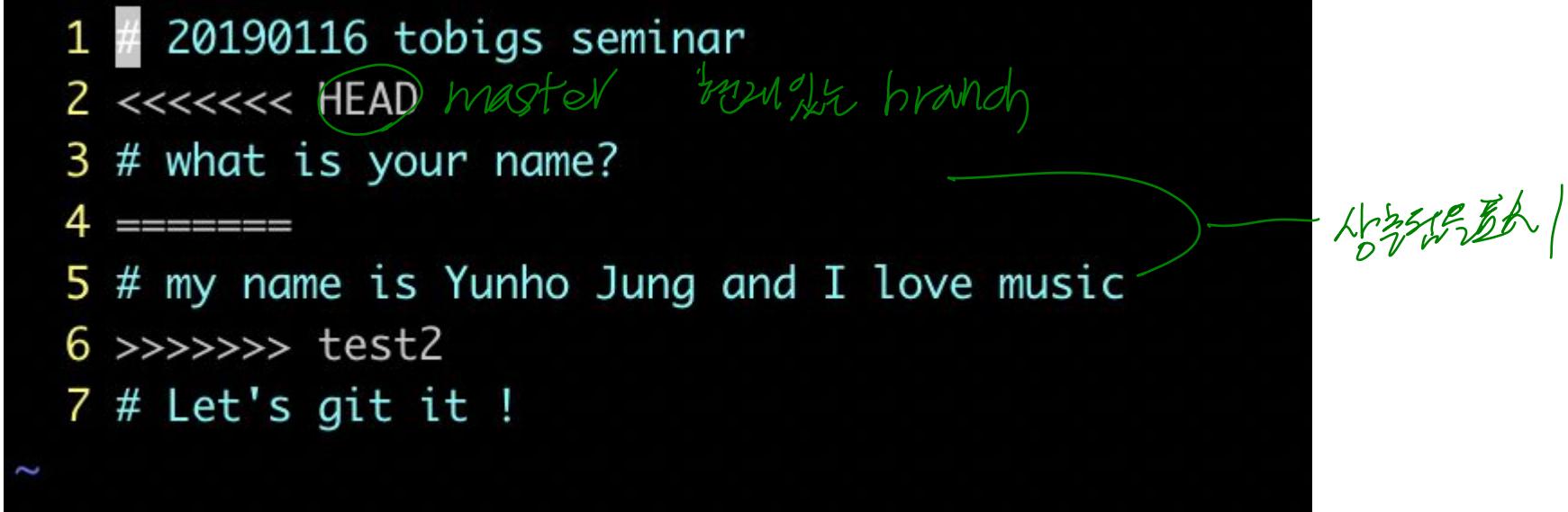
## Unit 03 | Git 사용

**같은 부분을 고치고 merge하려고 했기 때문 !**

## Unit 03 | Git 사용

### vim a.py -> 충돌 일어난 부분 수정

```
1 # 20190116 tobigs seminar
2 <<<<< HEAD master 현재 있는 branch
3 # what is your name?
4 =====
5 # my name is Yunho Jung and I love music
6 >>>>> test2
7 # Let's git it !
~
```



## Unit 03 | Git 사용

### vim a.py -> 충돌 일어난 부분 수정

```
1 # 20190116 tobigs seminar
2 # what is your name?
3 # my name is Yunho Jung and I love music
4 # Let's git it !
```

## Unit 03 | Git 사용

### solve a conflict and merge

:q → myX  
UpdEd

당신이 푸시한 Add commit 를 myX merge를 해줌!!

```
→ tobigs_git git:(master) ✘ vim a.py
→ tobigs_git git:(master) ✘ git add a.py
→ tobigs_git git:(master) ✘ git commit -m "solve a conflict and merge"
[master 0d4c158] solve a conflict and merge
→ tobigs_git git:(master) ✘ git log --branches --decorate --graph --oneline
```

```
* 0d4c158 (HEAD -> master) solve a conflict and merge
|\ 
| * 09437d8 (test2) edit a.py 2nd line
| * 6d7b15c edit a.py 2nd line
|/
| * fb0fc59 Merge branch 'test'
|\ 
| * 470824f (test) edit a.py and make b.py
| * 2b6560f edit a.py
|/
| * 6ae1357 first commit
(END)
```

→ 푸시로 돌아온 것

## Unit 03 | Git 사용

이전 커밋으로 돌아가고 싶다...?

t  
커밋을  
되돌리기  
위해  
수정하기

## Unit 03 | Git 사용

git log 하고 나온 결과  
**git reset commit\_id --hard**

```
→ tobigs_git git:(master) git reset fb0fc59b35164345bb36f1d3931aab57b8369f95 --hard
HEAD is now at fb0fc59 Merge branch 'test'
→ tobigs_git git:(master) vim a.py
→ tobigs_git git:(master) git log --branches --decorate --graph --oneline
→ tobigs_git git:(master) |
```

→ 토비스님

```
* 09437d8 (test2) edit a.py 2nd line
* fb0fc59 (HEAD -> master) Merge branch 'test'
|\
| * 470824f (test) edit a.py and make b.py
* | 2b6560f edit a.py
|/
* 6ae1357 first commit
(END)
```

## Unit 03 | Git 사용

**다시 최신 커밋으로 돌아가고 싶다...?**

## Unit 03 | Git 사용

**git reset --hard ORIG\_HEAD**

↳ 원상복구를 가리키는 것

```
→ tobigs_git git:(master) git reset --hard ORIG_HEAD
HEAD is now at 0d4c158 solve a conflict and merge
→ tobigs_git git:(master) git log --branches --decorate --graph --oneline
→ tobigs_git git:(master)
```

## Unit 03 | Git 사용

**git log --branches --decorate --graph --oneline**

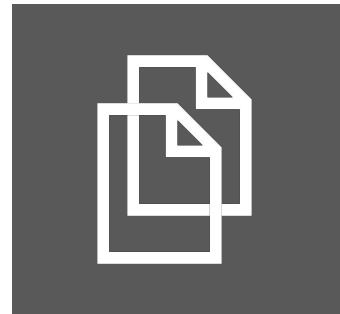
```
*   0d4c158 (HEAD -> master) solve a conflict and merge
|\ \
| * 09437d8 (test2) edit a.py 2nd line
* | 6d7b15c edit a.py 2nd line
| /
*   fb0fc59 Merge branch 'test'
|\ \
| * 470824f (test) edit a.py and make b.py
* | 2b6560f edit a.py
| /
* 6ae1357 first commit
(END)
```

## Unit 04 | Github 사용

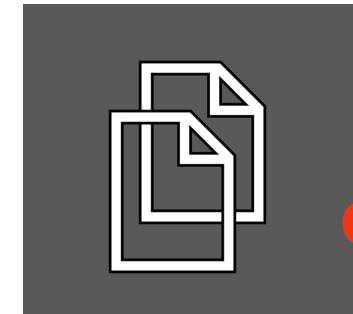
# Github 실습

## Unit 04 | Github 사용

### Working Directory

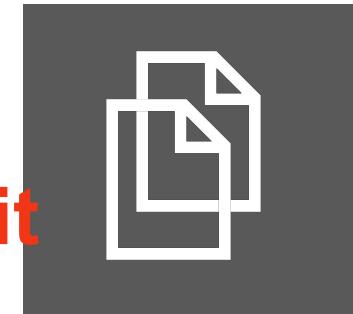


### Index



add

### Local Repository



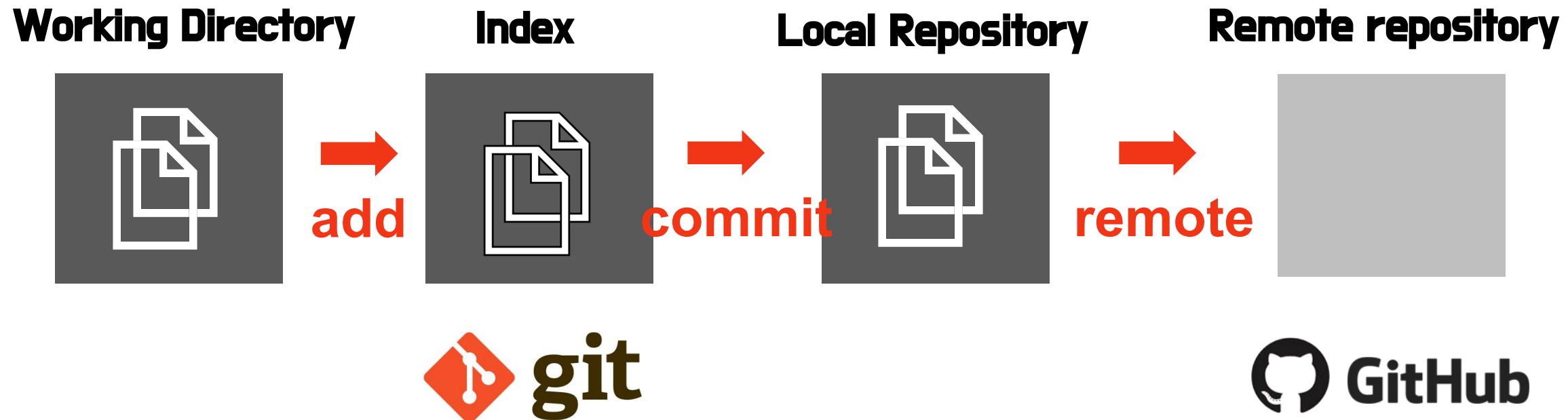
commit



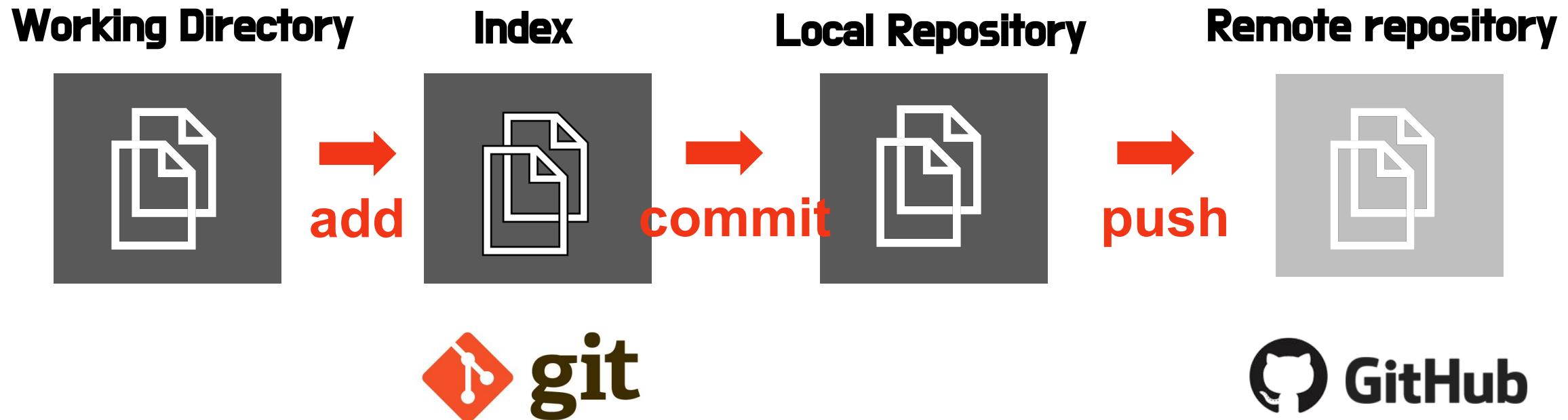
### Remote repository



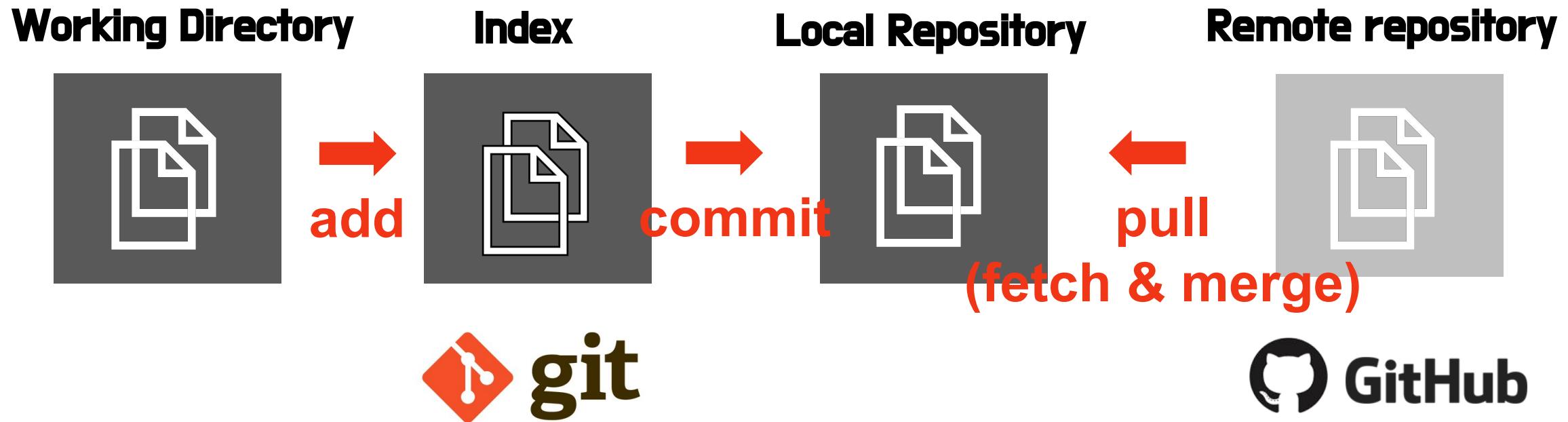
## Unit 04 | Github 사용



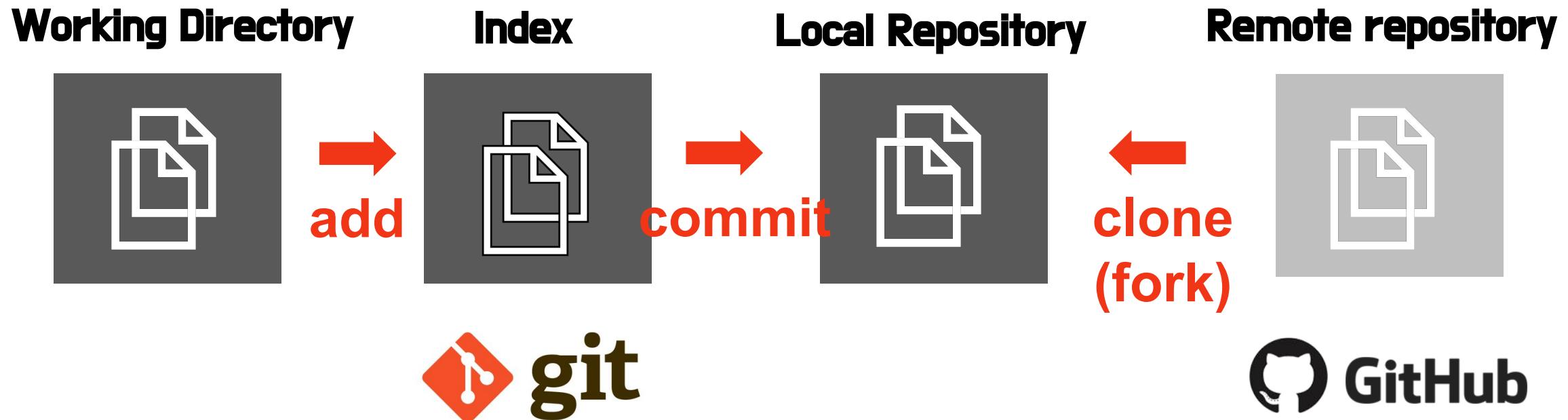
## Unit 04 | Github 사용



## Unit 04 | Github 사용

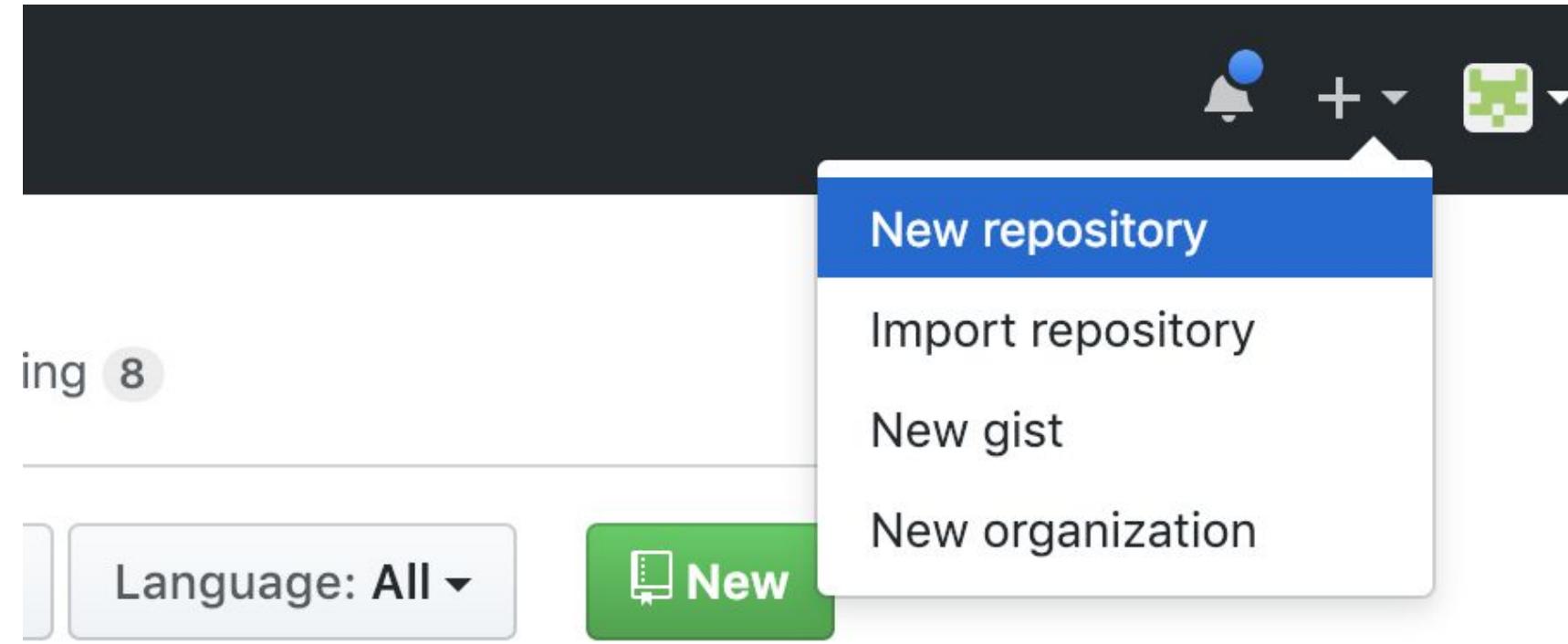


## Unit 04 | Github 사용



## Unit 04 | Github 사용

### New Repository 생성



## Unit 04 | Github 사용

# New Repository 생성

### Create a new repository

A repository contains all the files for your project, including the revision history.

Owner



YunhoJung ▾

Repository name

tobigs\_git



Great repository names are short and memorable. Need inspiration? How about [legendary-broccoli](#).

Description (optional)

Public

Anyone can see this repository. You choose who can commit.

Private

You choose who can see and commit to this repository.

Initialize this repository with a README

This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: None ▾

Add a license: None ▾



**Create repository**

## Unit 04 | Github 사용

### 원격 저장소와 연결

The screenshot shows a GitHub repository page for 'YunhoJung / tobigs\_git'. The top navigation bar includes 'Watch 0', 'Star 0', and 'Fork 0'. Below the header are tabs for 'Code', 'Issues 0', 'Pull requests 0', 'Projects 0', 'Wiki', 'Insights', and 'Settings'. A large central box contains instructions for repository setup:

- Quick setup — if you've done this kind of thing before**: Includes links for 'Set up in Desktop' (selected), 'HTTPS', 'SSH', and the repository URL 'git@github.com:YunhoJung/tobigs\_git.git'. A green circle highlights the 'Set up in Desktop' button, and a green arrow points from it to a handwritten note '3단계' (Step 3) next to a clipboard icon.
- ...or create a new repository on the command line**: Provides a shell script for initializing a new repository:

```
echo "# tobigs_git" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin git@github.com:YunhoJung/tobigs_git.git
git push -u origin master
```
- ...or push an existing repository from the command line**: Provides a shell script for pushing an existing repository:

```
git remote add origin git@github.com:YunhoJung/tobigs_git.git
git push -u origin master
```

## Unit 04 | Github 사용

### 원격 저장소와 연결

- 원격저장소를 관리할 수 있는 명령어 git remote를 이용하여 연결
  - **git remote add origin 사용자명@호스트:/원격/저장소/경로** 실행
  - **(git clone 사용자명@호스트:/원격/저장소/경로)**
  - origin은 원격저장소 이름 지정해주는 것일 뿐
- 이제 사용할 이름이라. 그냥 origin으로 쓰면 됨.
- origin 사용하고

## Unit 04 | Github 사용

### 로컬 저장소 → 원격 저장소으로 push

→ master 를 origin 에 올리자

```
→ tobigs_git git:(master) git push -u origin master
Counting objects: 22, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (17/17), done.
Writing objects: 100% (22/22), 1.82 KiB | 622.00 KiB/s, done.
Total 22 (delta 4), reused 0 (delta 0)
remote: Resolving deltas: 100% (4/4), done.
To github.com:YunhoJung/tobigs_git.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
```

## Unit 04 | Github 사용

### 원격 저장소 → 로컬 저장소 pull(fetch & merge)

- 원격저장소의 내용을 로컬 저장소로 업데이트(다른 사람의 수정한 것을 받기 위해)
- **git pull origin master(브랜치명) 실행**
- **origin의 내용이 master로 반영(fetch)되고 브랜치를 병합(merge)**



Cf) git pull을 할 때는 깃허브의 유저이름과 비밀번호를 쳐야 하는 경우가 많다. 매번 비밀번호를 입력하기 싫다면

**git config --global credential.helper 'store --file 경로'** 하면 해당 경로에 비밀번호가 저장된 파일이 생성된다.

단. 파일로 저장되는 만큼 보안에 취약하기 때문에 주의해야 합니다.

## Unit 04 | Github 사용

### 더 알면 좋은 내용 - 많이 쓰이는 패턴

- fork(타인 원격 저장소 -> 자신의 원격 저장소 복사)
- git clone(원격 저장소 -> 로컬 저장소 내용 복사)
- 브랜치 생성
- 작업 수행 후 add/commit/push
- pull request — *이제 깃헙은 머지하는거 아니야!*  *слияние !*
- 원격 저장소 관리자가 변경내역 확인 후 merge 여부 결정
- 코드 통기화 후 해당 브랜치 삭제

## Unit 04 | Github 사용

더 알면 좋은 내용 - <https://www.gitignore.io/>



## Unit 04 | Github 사용

### 더 알면 좋은 내용 - ssh 공개키 설정

<https://git-scm.com/book/ko/v2/Git-%EC%84%9C%EB%B2%84-SSH-%EA%B3%B5%EA%B0%9C%ED%82%A4-%EB%A7%8C%EB%93%A4%EA%B8%B0>

## Unit 04 | Github 사용

**로컬 저장소/로컬 저장소 - 원격 저장소/원격 저장소  
에서 발생하는 일들을 구분해서 이해하는 것이 중요하다!**

## Unit 05 | 과제

### 〈 필수 과제 〉

1. 팀장이 Github에 새로운 repository를 만들고 파일을 하나 생성한다.
2. 모두 각자의 이름 branch를 생성해 **하나의 파일**을 팀원 모두 한번 이상 수정 후 master에서 merge한다.
3. master를 push한다.

### 〈제출 방식〉

제출은 datamarket에 팀원 중 한명이 각 조이름과 작업한 github주소 올리기.

ex) 1조. <https://github.com/aaa/TobigsTest> 입니다!

ex) 본인의 이름 branch, 윤호 -> YunHo (O), 철수 -> ChrisKim (X)

## Unit 05 | 과제

### 〈 필수 과제 〉

#### 팀 배정(팀장, 팀원, 팀원)

1조 – 강수민, 권혜민, 김대웅, 홍지은

2조 – 김유민, 손창호, 심은선

3조 – 유기윤, 이도연, 이소라

4조 – 이영전, 임지수, 임채빈

5조 – 전용진, 정혜인, 한채연

6조 – 장유영, 장청아, 최세영

## Unit 05 | 과제

### 〈 선택 과제 〉

1. 자신의 Github에 새로운 repository를 만든다.
2. 10주 정규 세미나 동안 하게 되는 과제들을 자신의 Github에 계속 업로드한다.
3. commit 단위를 잘 고려하면서 과제를 수행한다.



→ 잘 생각하자

## Unit 05 | 과제

### 〈Reference〉

- <https://www.youtube.com/watch?v=hFJZwOfme6w&index=1&list=PLuHgQVnccGMA8iwZwrGyNXCGy2LAAsTXk>
- <https://git-scm.com/book/ko/v2/>
- <https://moon9342.github.io/git-github>
- <https://appendto.com/wp-content/uploads/2015/06/Screen-Shot-2015-06-24-at-8.37.13-PM-1024x663.png>
- [http://www.datamarket.kr/xe/board\\_jPWY12/48448](http://www.datamarket.kr/xe/board_jPWY12/48448)

# Q & A

들어주셔서 감사합니다.