Git / Github (3)

GitHub Flow Review

GitHub 은 Pull-request 가 중심인 협업 워크플로 위주로 설계 되어 있음

다른 프로젝트에 **내가 만든 commit** 을 제출한다는 의미 **(실제 전송단위는 branch)**

1. 기존 프로젝트 (master) 에서 **Fork(복사)** 해온다
2. Clone 해서 **토픽 브랜치**를 만든다
3. 내가 만든 **commit** (뭔가 수정) 을 보낸다
4. 자신의 GitHub 프로젝트에 브랜치를 **Push** 한다
5. 기존 프로젝트 (master) 에 **Pull-request** 를 보낸다
6. 토론하면서 그에 따라 계속 커밋한다
7. 기존 프로젝트 (master) 소유자는 Pull-reques 를 검토후 , **Merge** 한다

Git/GitHub(1) Review

1) Git 과 GitHub 차이는 ?

- **Git** 은 각 컴퓨터 (**local**) 에 설치되어 소스코드 관리가 가능한 프로그램
- **GitHub** 는 **remote** 저장소가 있는 외부서버를 지칭
- **Git** 이라는 **Source Control** 방법을 Github 이 사용할 뿐

2) Commit 과 Push 차이는 ?

- **commit** 은 **local** 작업폴더에 history 를 쌓는 것으로 외부망 (**internet**) 필요없음
- **push** 는 **remote** 저장소 (GitHub 등) 에 history 를 쌓는 것이어서 외부망 (**internet**) 이 필요하다

Git/GitHub(2) Review

3) Push 와 Pull-request 의 차이는 ?

- **Push** 는 프로젝트를 fork 해온 **나의 원격 저장소에** commit 을 반영하여 파일을 변경
- **Pull-request** 는 **실제 프로젝트의 원격 저장소**에 commit 을 반영해달라고 요청하는 것
  (실제 commit 은 그 프로젝트 관리자가 accept 해줘야 반영이 됨)

4) Rebase 과 Merge 차이는 ?

- **Rebase** 는 현재 작업중인 ‘branch’ 와 ‘master’ 를 새로운 master 로 합치기 전에 branch 이력까지 통합
- **Merge** 는 각 branch 마다 서로 다른 이력을 가진 채로 통합

Clone & Pull & Fetch

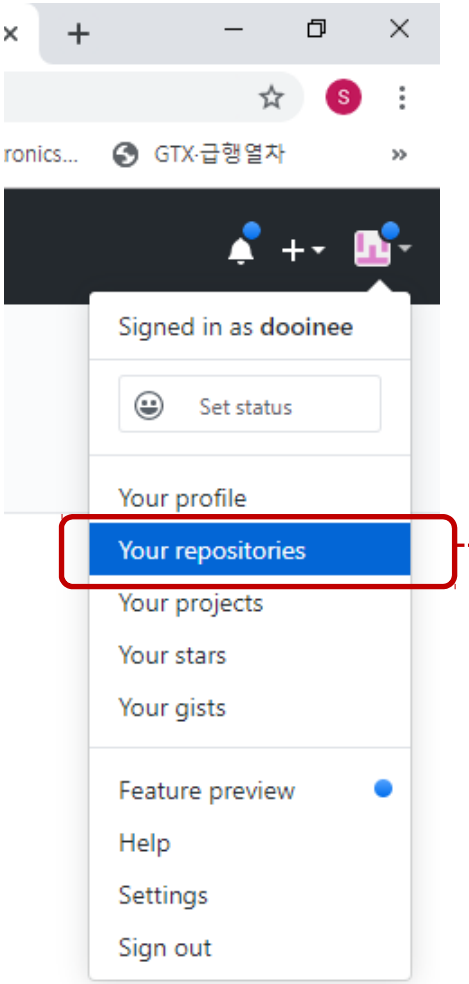
Clone – 원격 저장소 복제하기

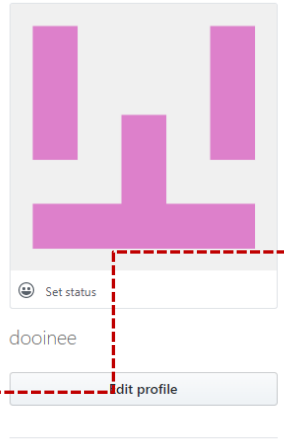
- **원격 저장소를 웹에서 통째로 복제해와 내 PC 에서 직접 작업**
- **변경 이력도 함께 로컬 저장소에 복제되어 오므로 , 원래 원격 저장소와 똑같이 이력을 참조하고 commit 을 진행할 수 있음 .**

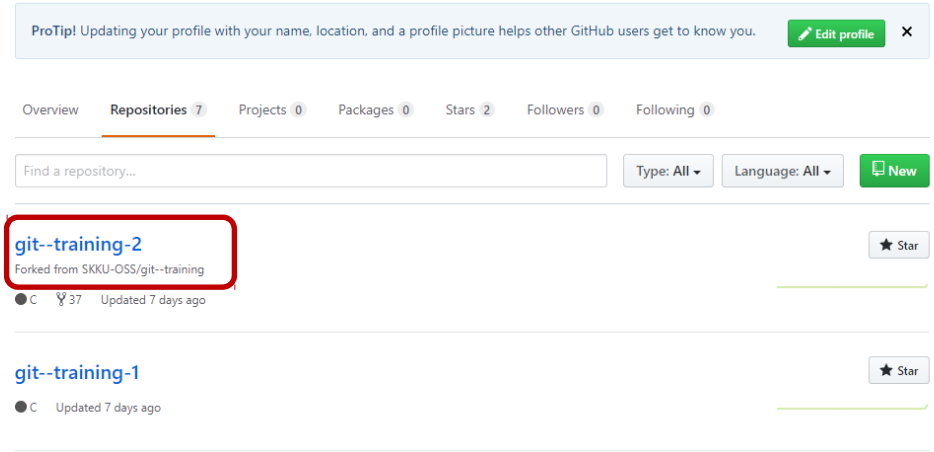
출처 : source code management system(Git) – Taeung Song,NIPA OpenFrotierLab(2016)

Clone(1) – 저번 시간 repository 삭제

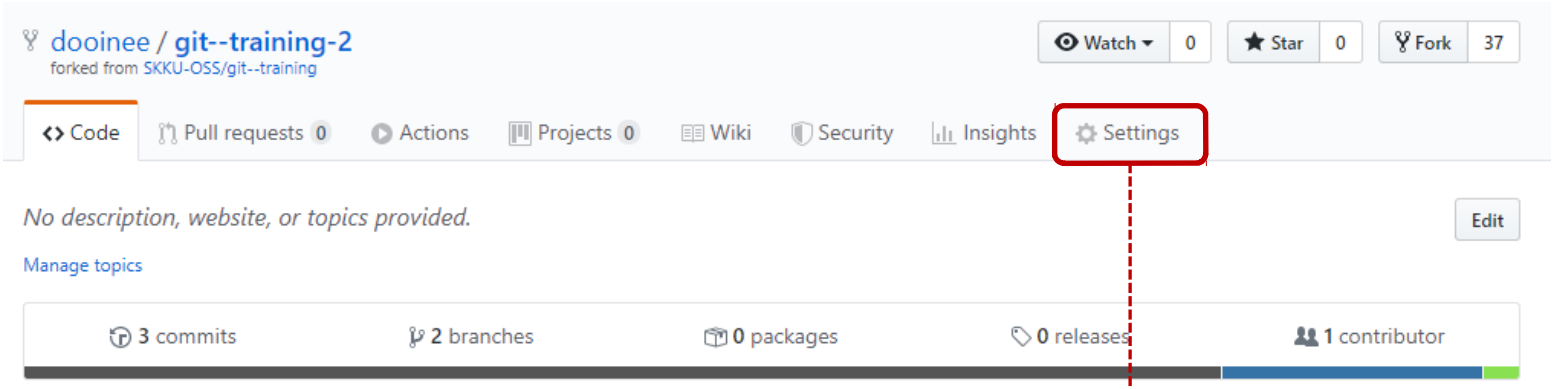
저번 시간에 fork 해온 repository 를 삭제하자 !



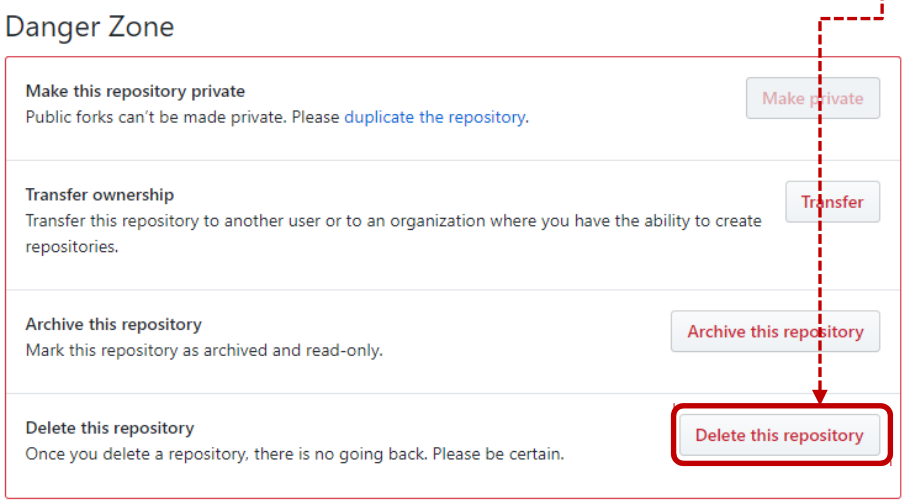




Clone(1) – 저번 시간 repository 삭제



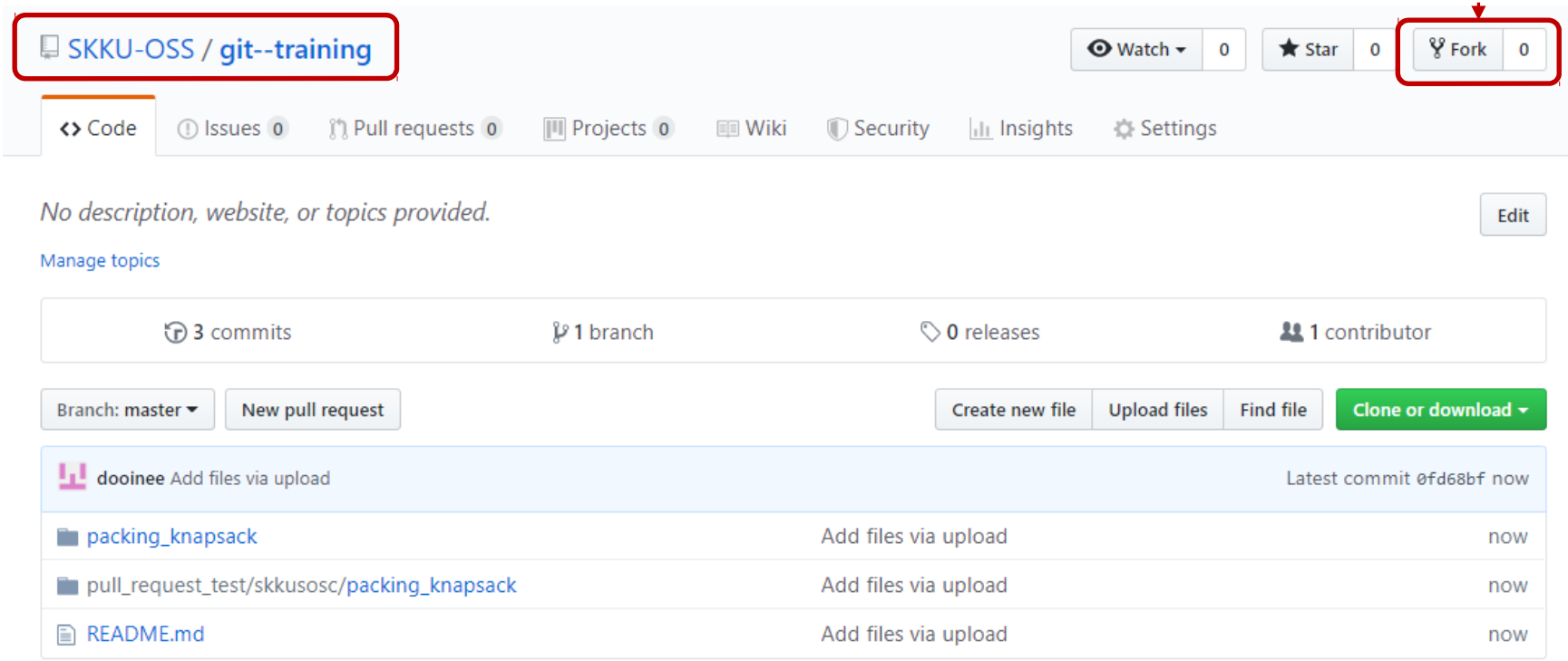
Settings 를 누른 후 맨 아래로 스크롤을 내리면 delete 단추가 있다 .



Clone(2) – 다시 fork 하기

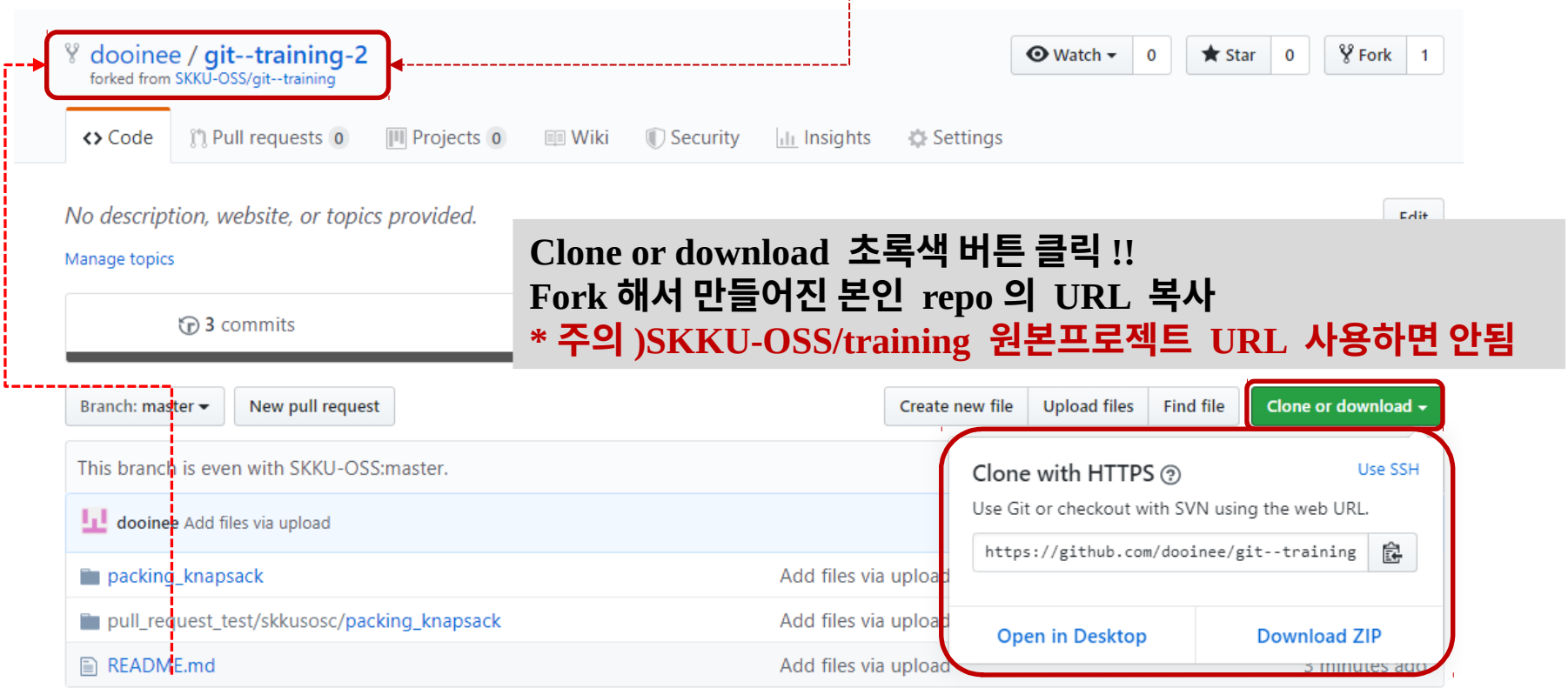
- 주의 fork 는 본인 프로젝트를 대상으로 하는게 아니다 . 아래 url 로 들어가자

https://github.com/SKKU-OSS/git--training 가서 **Fork** 버튼 누르자



Clone(3) 다시 fork 하기

Fork 가 되면 내 원격저장소가 추가 된다



Clone or download 초록색 버튼 클릭 !!
Fork 해서 만들어진 본인 repo 의 URL 복사
*** 주의)SKKU-OSS/training 원본프로젝트 URL 사용하면 안됨**

필수) forked from SKKU-OSS/git--training 표시 확인하기

Clone(4) – 기본 설정

1) Git-bash 혹은 터미널 실행 후에

1-1) 미리 캐치 저장되어 있을지 모를 계정정보 삭제 **(처음 설치시 생략 가능)**

```
$ git config  --global user.email –-unset-all user.name
```

```
$ git config  –-global user.name –-unset-all user.email
```

2) 나의 GitHub 계정 이메일 **(GitHub 계정 이메일)** 과 이름 **(본인 영문이름 or 닉네임)** 을 적자

```
$ git config  --global user.email “ 본인메일 @gmail.com”
```

```
$ git config  –-global user.name “ 본인이름 or 닉네임  skkusosc”
```

3) 제대로 설정되었는지 확인하기

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

```
user.email=dooinee@gmail.com
user.name=suin skkusosc
core.repositoryformatversion=0
core.filemode=true
core.bare=false
core.logallrefupdates=true
```

Clone(5) – Clone 하기

1) (git-bash/ 터미널에서) 최초경로 HOME 경로로 이동하자

(report-card 작업하던 폴더에서 벗어나기)

```
$ cd ~
```

2) clone 으로 fork 한 repo 받아오기 **" 아까 fork 한 repo 에서 복사한 URL"**

```
$ git clone https://github.com/dooinee/git--training-1.git
```

3) clone 한 프로젝트 폴더로 이동하기 (만약 프로젝트명이 git—training-1 이면 그 이름으로 이동)

```
$ cd git--training-1
```

4) 작업할 , 토픽 브랜치 (develop) 따로 만들기

```
$ git checkout –b develop
```

5) pull_request_test 폴더로 이동하자

```
$ cd pull_request_test
```

Clone(6) – push 하기

6) **내 이름으로 된 (skkusosc 대신)** 폴더 만들고

```
$ mkdir suin; cd suin
```

7) myfile.txt 열어서 (아래 간단한 글 (만) 작성 해보자)

쉽게 배우는 Git 명령어

```
$ git add myfile.txt
```

8) 준비된 파일 commit

```
$ git commit –sm “my first commit”
```

9) 내가 fork 한 repo 의 develop 브랜치로 push (주의 : master 아님)

```
$ git push origin develop
```

Clone(6) – push 하기

10) develop 브랜치와 master 브랜치 병합

```
$ git checkout master
```

```
$ cd ..
```

왜냐하면 master 브랜치에는 'suin' 디렉토리가 없기 때문에 현재 디렉토리에서 벗어나주어야 함

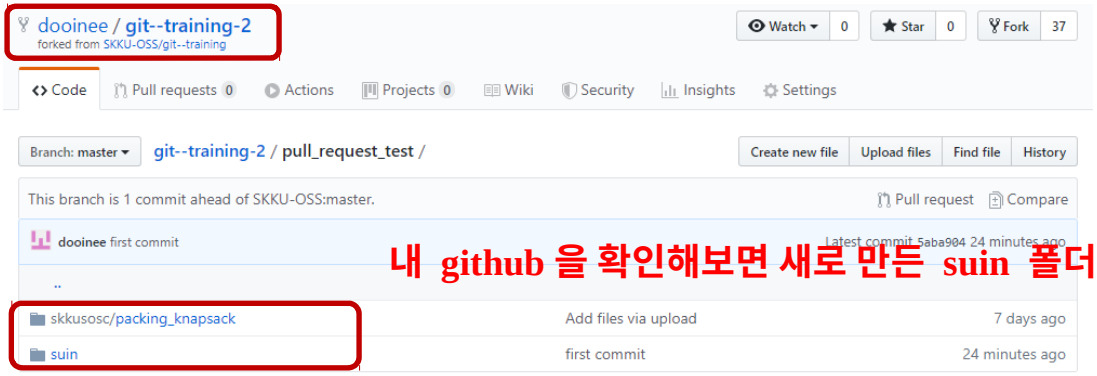
```
$ git merge develop
```

```
$ git branch –d develop
```

필요없는 develop 브랜치 삭제

11) 내가 fork 한 repo 로 합쳐진 'master' branch 를 push

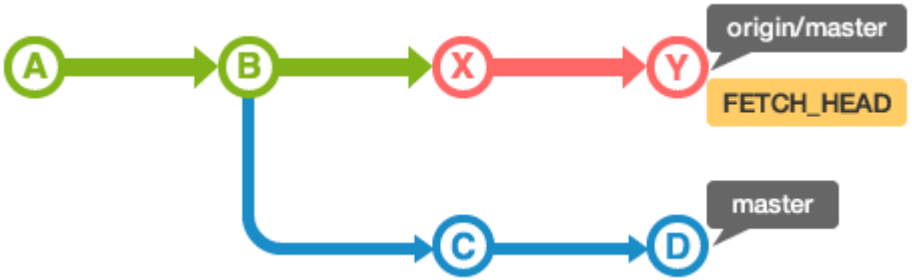
```
$ git push origin master
```



내 github 을 확인해보면 새로 만든 suin 폴더가 반영된 것을 알 수 있다 .

Fetch – 원격 저장소 가져오기

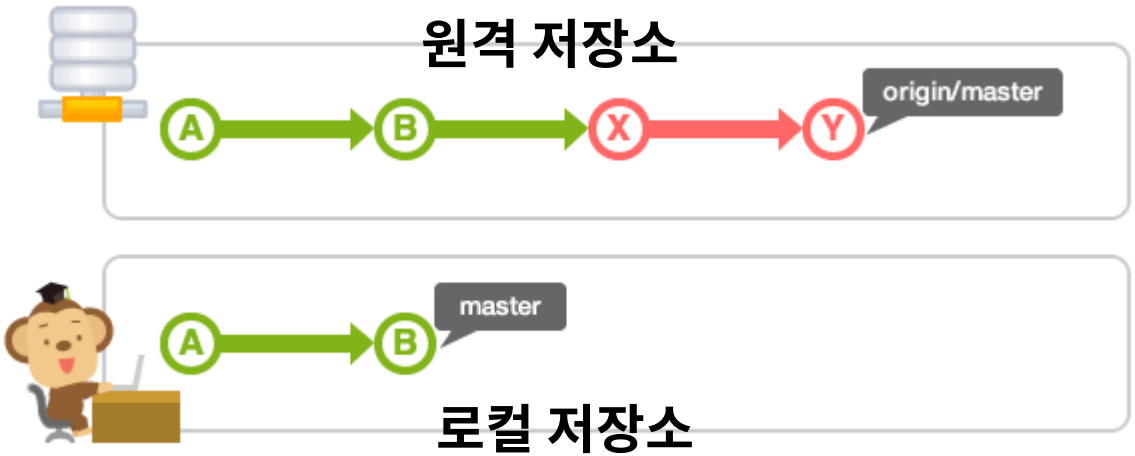
- 단순히 원격 저장소의 내용을 확인만 하고 로컬 데이터와 병합은 하고 싶지 않은 경우
- fetch 를 실행하면 원격 저장소의 최신 이력 확인 가능 . 이 때 가져온 최신 커밋 이력은 이름 없는 브랜치로 로컬에 가져오게 됨 . 이 브랜치는 ‘FETCH_HEAD’ 의 이름으로 체크아웃 가능



- 이 상태에서 원격 저장소의 내용을 로컬 저장소의 master 에 통합하고 싶은 경우 , ‘FETCH_HEAD’ 브랜치를 merge 하거나 pull 한다 .

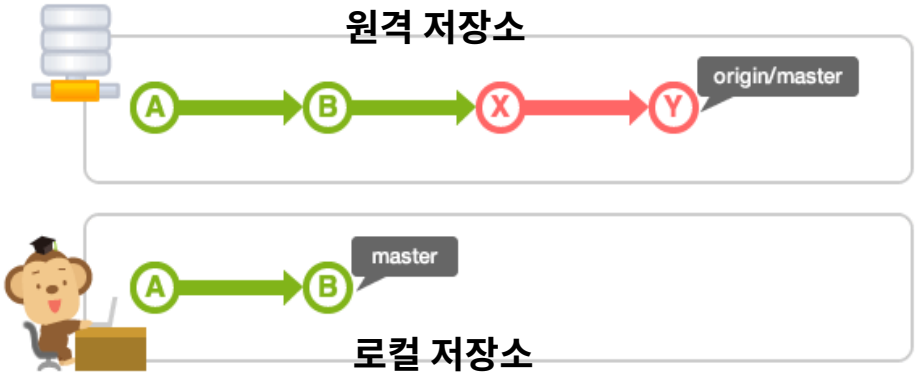
Pull – 원격 저장소 가져와 병합하기

- 원격 저장소를 공유해 여러 사람이 함께 작업을 하면 , 모두가 같은 원격 저장소에 push 함 .
- 그럴 경우 다른 사람이 원격 저장소에 올려놓은 (push) 변경 내용을 내가 작업하던 로컬 저장소에 적용할 필요가 있음 .
- 한마디로 pull 은 (fetch + merge)



Pull(1) – 원격 저장소 가져와 병합하기

- **상황 1. 로컬 저장소의 모든 변경 사항이 반영되어 있는 원격 저장소**



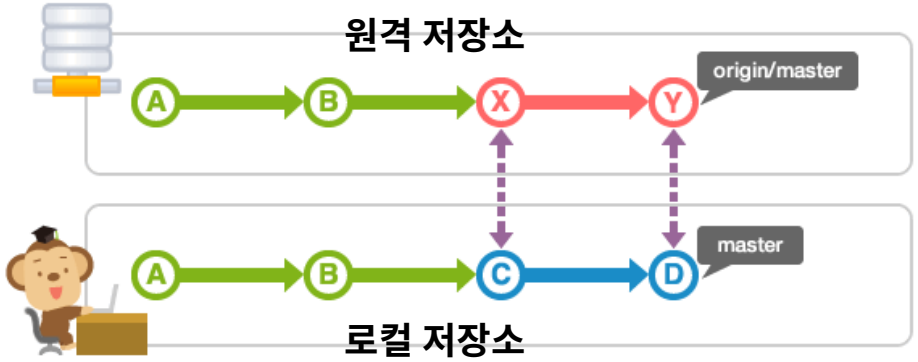
- **Fast-forward 병합**

(1) origin 원격 저장소의 내용을 master 로 복사

```
$ git pull origin master
```

Pull(2) – 원격 저장소 가져와 병합하기

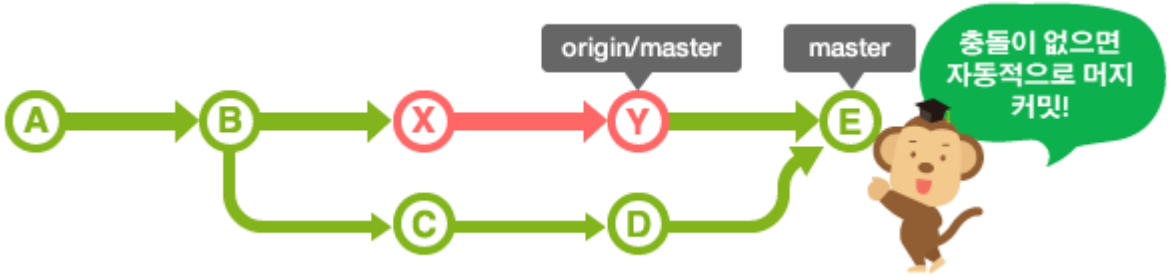
- **상황 2. 로컬 저장소의 ‘master’ 브랜치에서도 변경 사항이 생긴 경우**



- **Non fast-forward 병합**

(1) 충돌을 제거해주고 병합 시작

```
$ git pull origin master
```



Pull Test (1) – 실습

1) 자신의 git 계정에 ‘pull_test’ 라는 repository 만들기

2) ‘pull_test’ 의 origin 에서 clone 해오기

```
$ git clone ‘repository address’
```

3) local 의 clone 해온 pull_test 폴더에 , 아이캠퍼스 pull_test 폴더의 3 가지 파일 추가 후 push (report_card.c / report_card.pdf / text.txt)

```
$ git add .
```

```
$ git commit –m “Add file”
```

```
$ git push origin master
```

Pull Test (2) – 실습

4) local 에서 ‘push_test1’ / ‘push_test2’ 폴더 생성

```
$ mkdir push_test1
```

```
$ mkdir push_test2
```

5) 각각 ‘push_test1’ / ‘push_test2’ 폴더에 origin 으로부터 clone

```
$ git clone ‘repository address’ at ‘push_test1’ folder
```

```
$ git clone ‘repository address’ at ‘push_test2’ folder
```

Pull Test (3) – 실습

6) ‘push_test1’ 의 ‘ report_card.c’ 맨 뒤에 printf(“push push baby 1”) 추가 후 origin 에 푸시

```c
#include<stdio.h>

void main(){
  printf("This is Report Cards.c\n");
  printf("what is the difference ? \n");
  printf("Math: 180 \n");
  printf("English: 100 \n");
  printf("Sum grade: 190 \n");
  printf("Average grade: 95 \n");
  printf("Push Push Baby: 1");
}
```

```
$ git add .
```

```
$ git commit –m “Add::push push baby 1”
```

```
$ git push origin master
```

Pull Test (4) – 실습

7) ‘push_test2’ 의 ‘ report_card.c’ 맨 뒤에 printf(“push push baby 2”) 추가 후 origin 에 푸시

```c
#include<stdio.h>

void main(){
  printf("This is Report Cards.c\n");
  printf("what is the difference ? \n");
  printf("Math: 180 \n");
  printf("English: 100 \n");
  printf("Sum grade: 190 \n");
  printf("Average grade: 95 \n");
  printf("Push Push Baby: 2");
}
```

$ git add .

$ git commit –m “Add::push push baby 2”

$ git push origin master

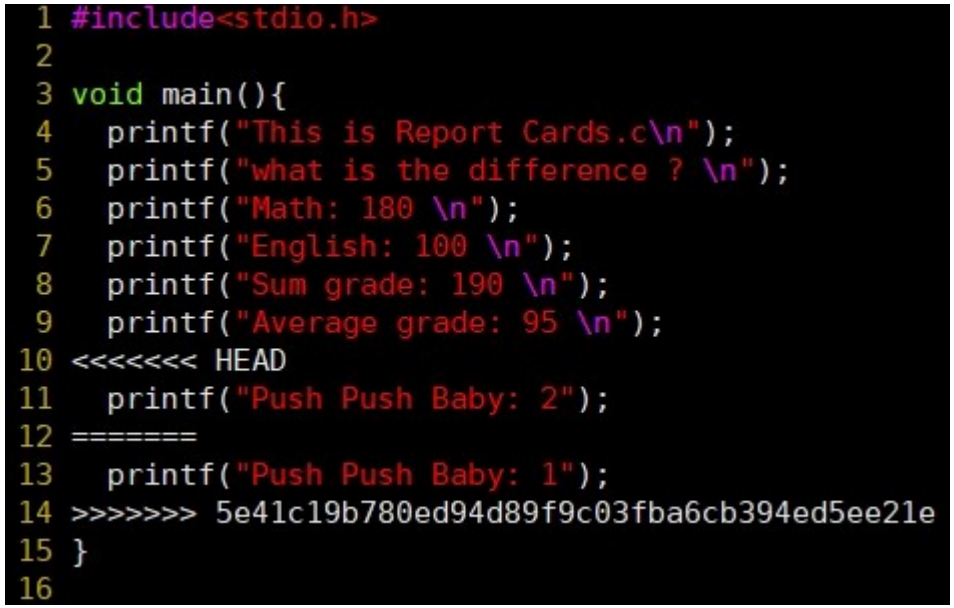
```
 ! [rejected]        master -> master (fetch first)
error: failed to push some refs to 'https://github.com/testyslee/test.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

Pull Test (5) – 실습

```
$ git pull origin master
```

8) ‘push_test2’ 의 ‘report_card.c’ 파일 vim 으로 열어보기

```
$ vim report_card.c
```



```
 1 #include<stdio.h>
 2
 3 void main(){
 4   printf("This is Report Cards.c\n");
 5   printf("what is the difference ? \n");
 6   printf("Math: 180 \n");
 7   printf("English: 100 \n");
 8   printf("Sum grade: 190 \n");
 9   printf("Average grade: 95 \n");
10 <<<<<<< HEAD
11   printf("Push Push Baby: 2");
12 =======
13   printf("Push Push Baby: 1");
14 >>>>>>> 5e41c19b780ed94d89f9c03fba6cb394ed5ee21e
15 }
16
```

Pull Test (6) – 실습

9) ‘push_test2’ 의 ‘ report_card.c’ 파일 수정 및 저장 후 git add / commit / push 실행

```c
#include<stdio.h>

void main(){
  printf("This is Report Cards.c\n");
  printf("what is the difference ? \n");
  printf("Math: 180 \n");
  printf("English: 100 \n");
  printf("Sum grade: 190 \n");
  printf("Average grade: 95 \n");
  printf("push push baby: 2");
  printf("Push Push Baby: 1");
}
```

```
$ git add .
```

```
$ git commit –m “Merge::push push baby”
```

```
$ git push origin master
```