

Git을 사용하여 협업을 효율적으로 해보기!

이재현(20기)

Git을 사용하여 협업을 효율적으로 해보기!

깃 / 깃허브 / GUI툴을 사용하여 협업을 할 수 있는 워크
플로우를 재밌게 익히게 될거예요~

이재현(20기)

Case 1



ESC_final_project

어떤 파일 하나가 있다.

Case 1



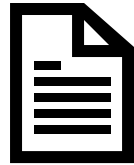
ESC_final_project

이 파일을 계속해서 수정하는데 기록을 남기고 싶다.

Solution



ESC_final_project
_2020_01_30_11



ESC_final_project
_2020_01_30_12



ESC_final_project
_2020_01_30_13



ESC_final_project
_2020_01_30_14

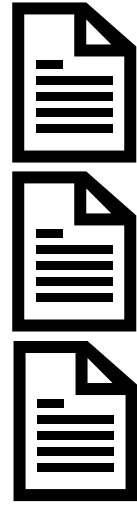


ESC_final_project
_2020_01_30_15



파일 이름 뒤에 날짜를 새겨서 저장을 하자

Solution



너무 많아지면?

Solution



너무 많아지면?



앞에 10개를 삭제하고 뒤 5개만 살리기?

누가 실수해서 10번째 파일에서 코드를 실수로 지웠으면?

불안하니깐 일단 다 남기기?

안볼거면서 왜 남김?

Case 2

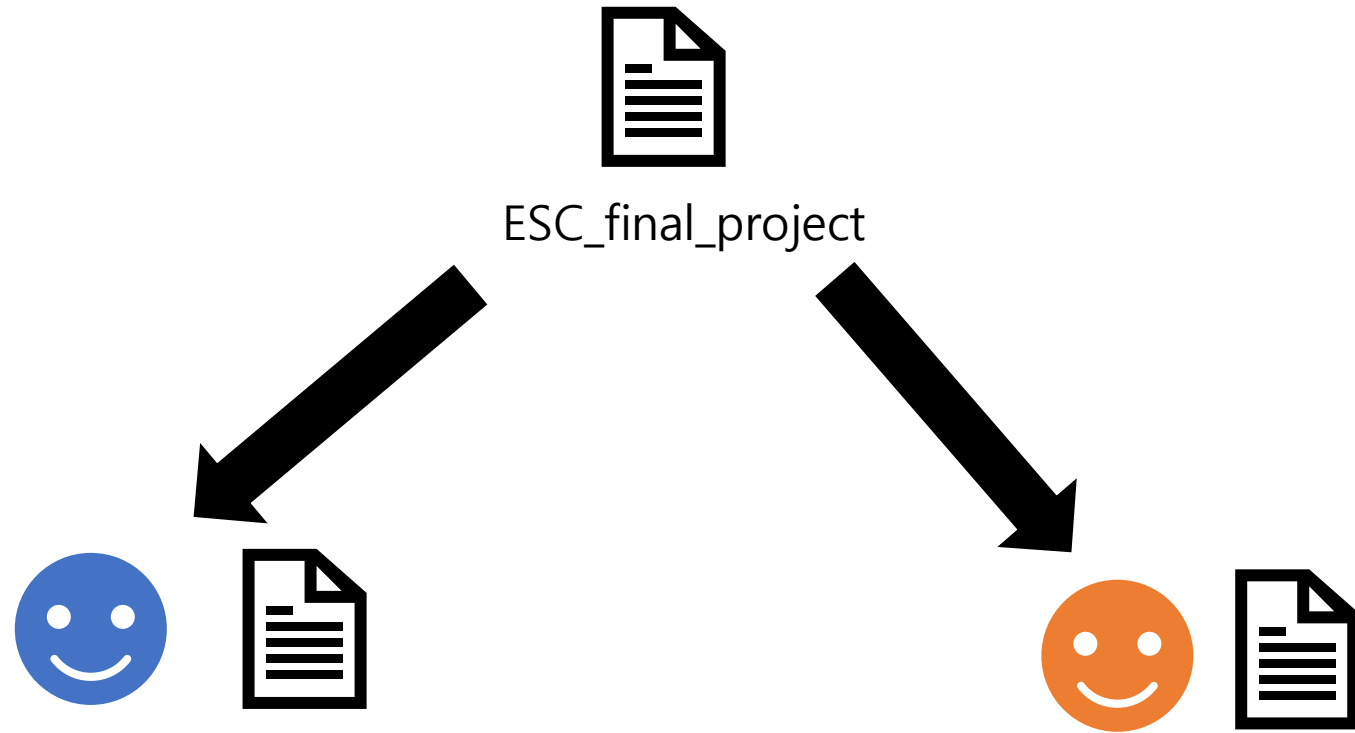


ESC_final_project



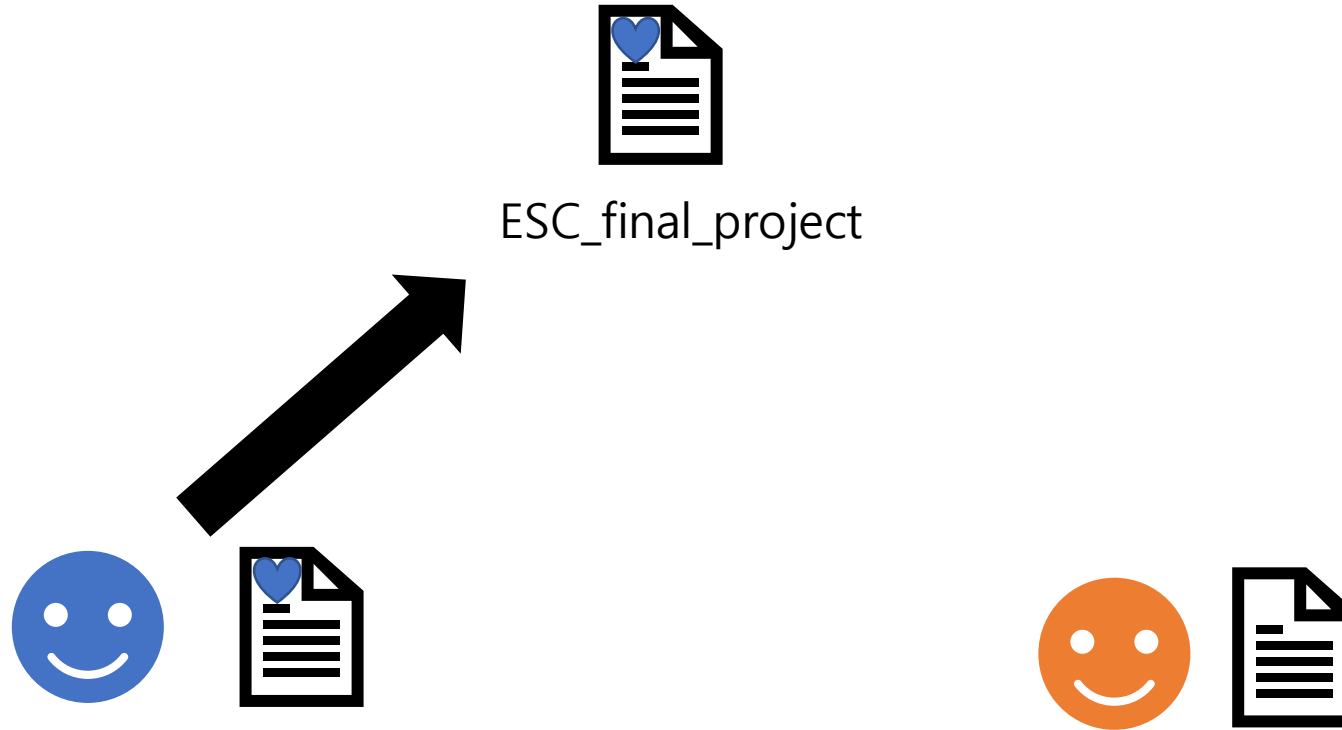
어떤 파일 하나가 서버에 올려져있다.

Case 2



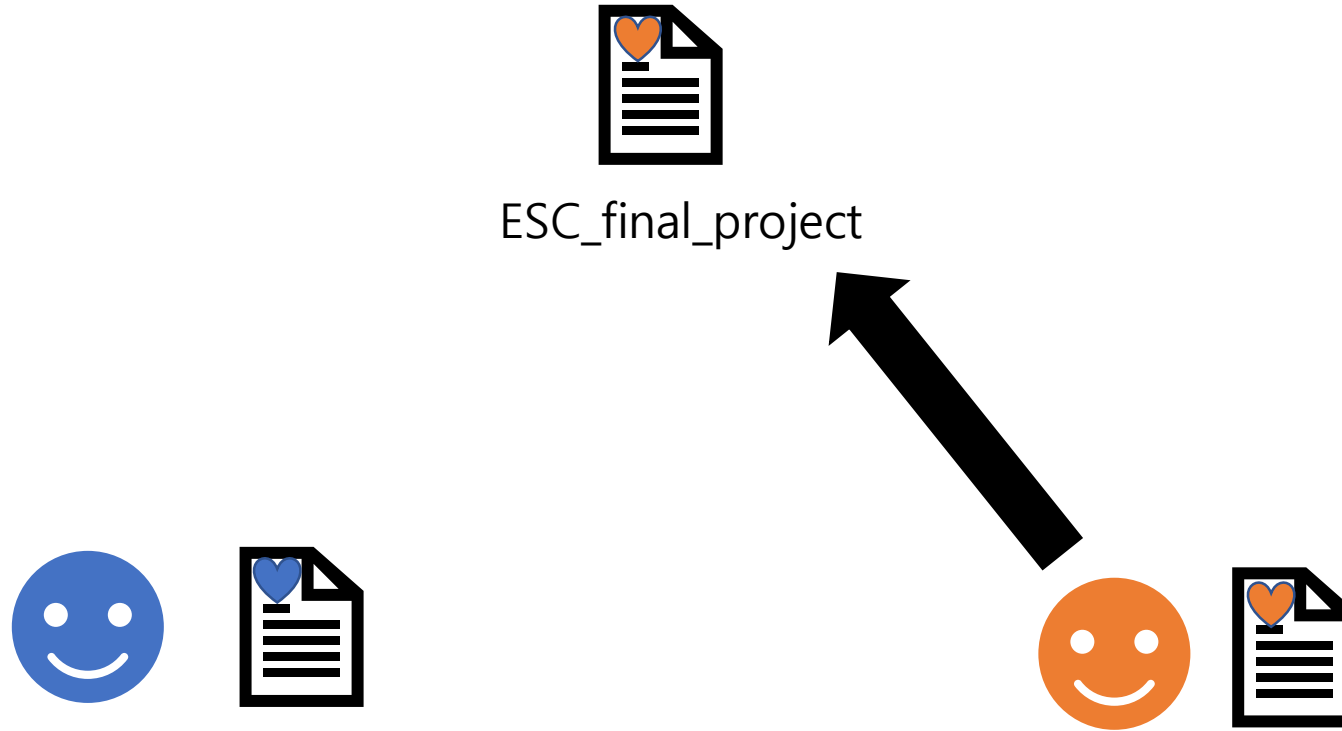
파랑이와 주황이가 다운로드를 받았다.

Case 2



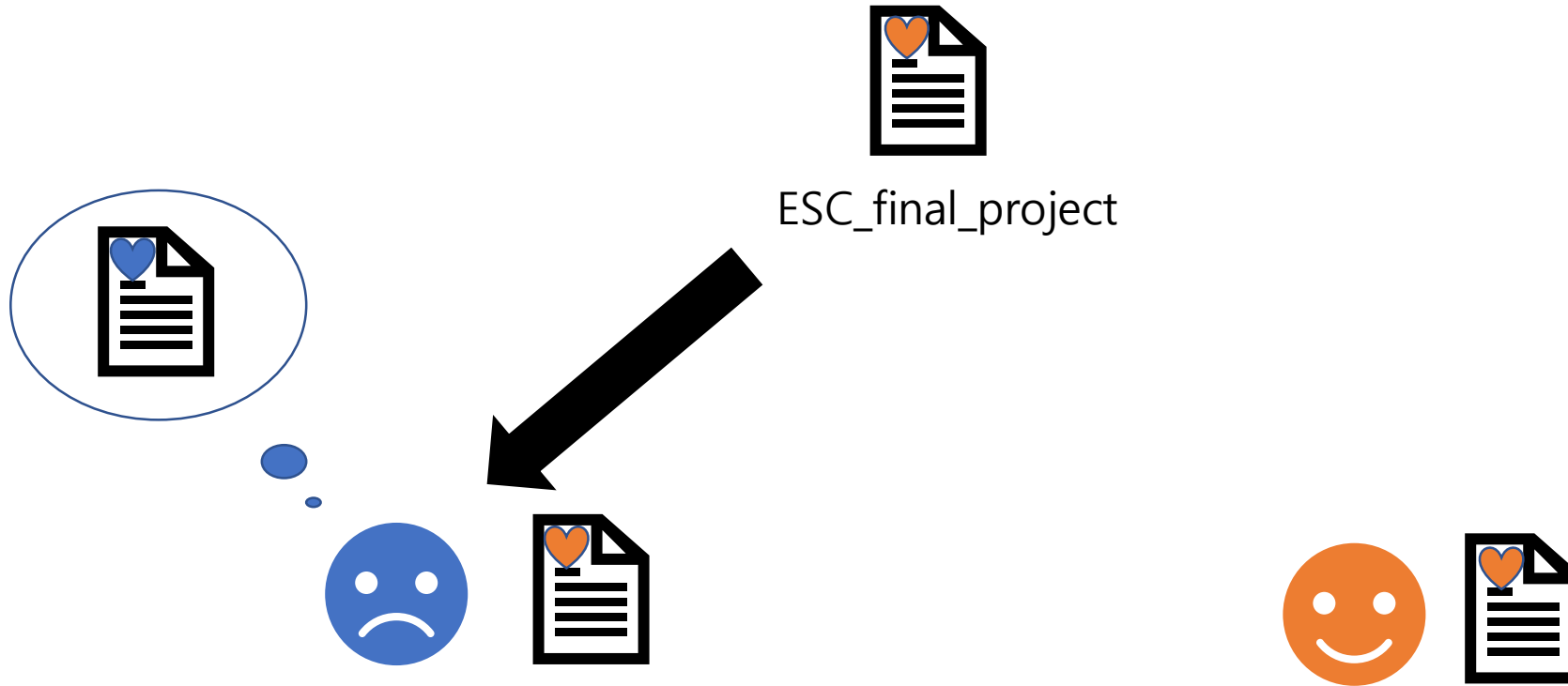
파랑이가 수정을 하여 서버에 업로드를 하였다.

Case 2



주황이가 수정을 하여 서버에 업로드를 하였다.

Case 2



파랑이 : 내 파일 어디감?



리누스 토발즈 : 리눅스 커널과 깃을 최초로 개발한 사람



분산 버전 관리 시스템

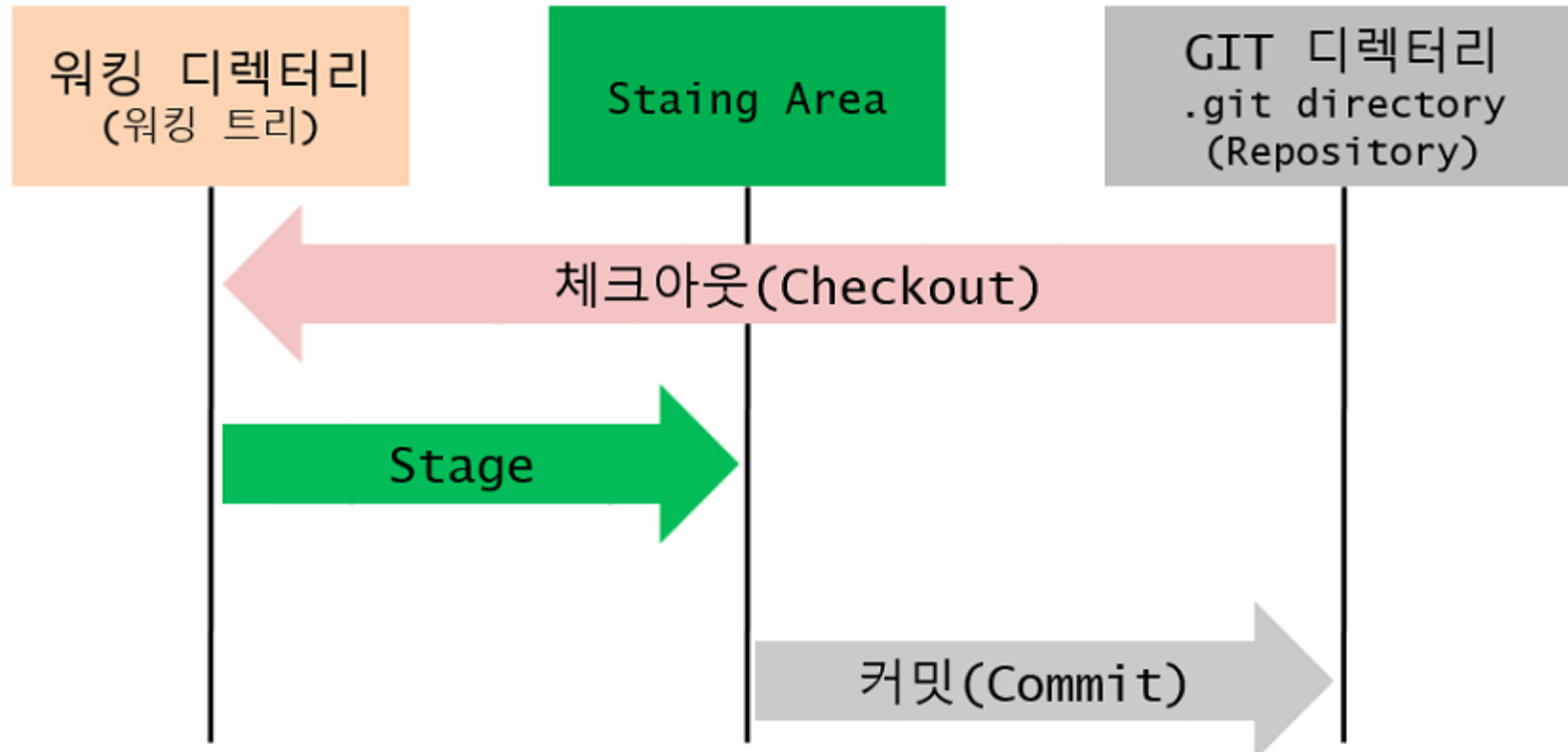
분산 : 여러명에서 동시에

버전 관리 : 버전관리를 할 수 있게

도와주는 프로그램

Git local

Git 프로젝트의 세 가지 단계



- 워킹 디렉터리 - 프로젝트를 진행하는 실제 작업 공간
- Staging Area - 워킹 디렉터리에서 작업한 내역을 Git 디렉터리로 커밋 하기 위해 담아두는 장바구니
- Git 디렉터리 - .git 이라는 이름의 디렉터리, Git 프로젝트에 대한 모든 정보를 담고 있는 핵심 디렉터리

git init : Git Repository 생성

```
MASTER@DESKTOP-49OPGLB MINGW64 /d/ESC/Test
$ git init
Initialized empty Git repository in D:/ESC/Test/.git/

MASTER@DESKTOP-49OPGLB MINGW64 /d/ESC/Test (master)
$ ls -al
total 8
drwxr-xr-x 1 MASTER 197121 0 1월 28 12:46 ./
drwxr-xr-x 1 MASTER 197121 0 1월 28 12:45 ../
drwxr-xr-x 1 MASTER 197121 0 1월 28 12:46 .git/
```

TMI : master branch는 git init과 동시에 만들어진 브랜치
(브랜치는 뒤에서 정의함)

새 파일 작성

```
MASTER@DESKTOP-490PGLB MINGW64 /d/ESC/Test (master)
$ vim test1.txt

MASTER@DESKTOP-490PGLB MINGW64 /d/ESC/Test (master)
$ cat test1.txt
안녕하세요?

MASTER@DESKTOP-490PGLB MINGW64 /d/ESC/Test (master)
$ ls -al
total 9
drwxr-xr-x 1 MASTER 197121 0 1월 28 18:20 ./
drwxr-xr-x 1 MASTER 197121 0 1월 28 18:17 ../
drwxr-xr-x 1 MASTER 197121 0 1월 28 18:20 .git/
-rw-r--r-- 1 MASTER 197121 17 1월 28 18:20 test1.txt

MASTER@DESKTOP-490PGLB MINGW64 /d/ESC/Test (master)
$ |
```

git status : Git Repo 상태 확인

```
MASTER@DESKTOP-490PGLB MINGW64 /d/ESC/Test (master)
$ git status
on branch master

No commits yet

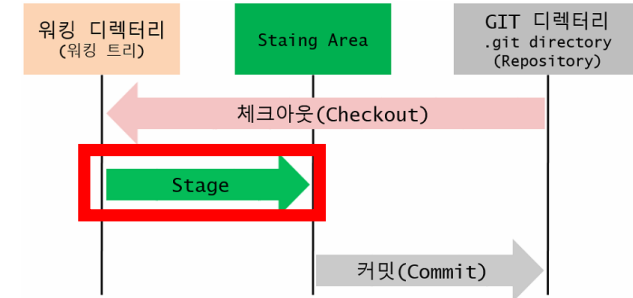
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    test1.txt

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

TMI : Untracked files는 git이 추적하지 않는 파일

git add : Staging Area로 옮기기



```
MASTER@DESKTOP-490PGLB MINGW64 /d/ESC/Test (master)
$ git add test1.txt

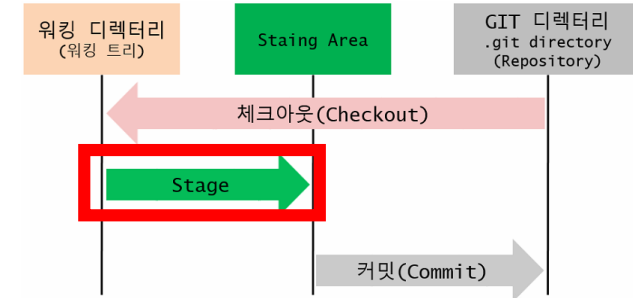
MASTER@DESKTOP-490PGLB MINGW64 /d/ESC/Test (master)
$ git status
on branch master

No commits yet

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

        new file:   test1.txt
```

파일을 좀 더 수정해보자



```
안녕하세요?  
여러분  
~  
~  
~  
~  
~  
~  
~  
~  
~  
~  
~  
~  
~  
~  
~  
~  
~  
~  
~  
~  
test1.txt[+] [unix] (13:27 28/01/2020) 2,7-5 모두
```

git status : Git Repo 상태 확인

```
MASTER@DESKTOP-490PGLB MINGW64 /d/ESC/Test (master)
$ git status
On branch master

No commits yet

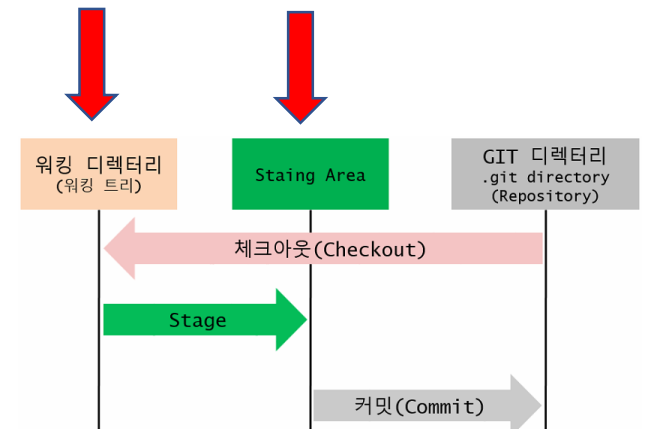
Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   test1.txt

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:   test1.txt
```

Staging Area와
워킹 디렉터리가
다른 상황



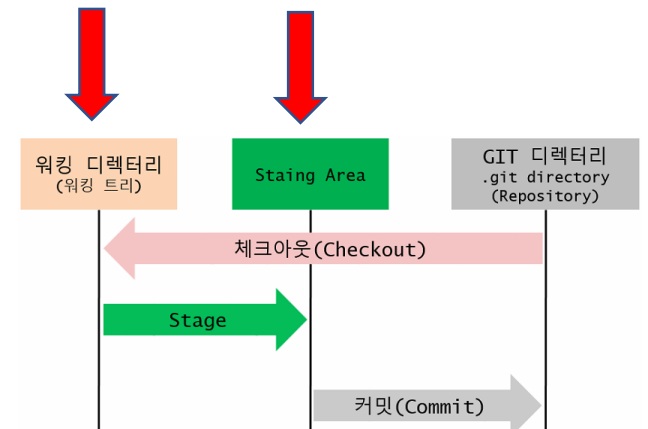
git diff : 현재 워킹 디렉토리와 Staging Area의 차이를 보여줌

```
MASTER@DESKTOP-49OPGLB MINGW64 /d/ESC/Test (master)
$ git diff
diff --git a/test1.txt b/test1.txt
index 250ff0f..8e9048e 100644
--- a/test1.txt
+++ b/test1.txt
@@ -1,2 @@
안녕하세요?
+여러분
```

git status : 파일이 변경됐다는 사실

git diff : 어떤 내용이 변경됐는지

Staging Area와
워킹 디렉터리가
다른 상황



git add : Staging Area로 옮기기

```
MASTER@DESKTOP-490PGLB MINGW64 /d/ESC/Test (master)
$ git add test1.txt

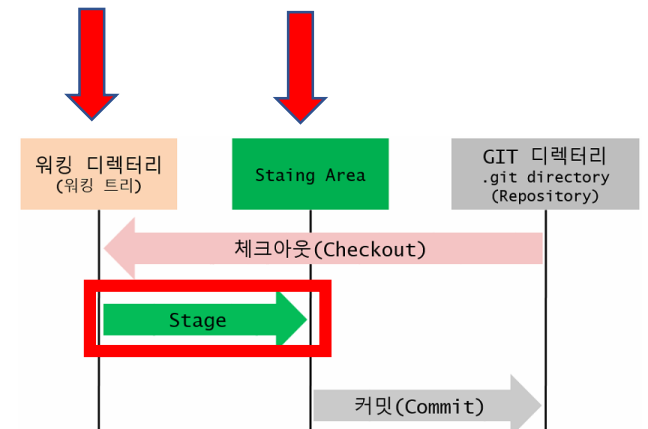
MASTER@DESKTOP-490PGLB MINGW64 /d/ESC/Test (master)
$ git status
on branch master

No commits yet

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

        new file:   test1.txt
```

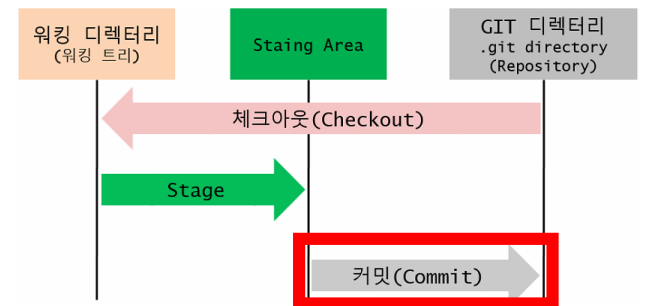
Staging Area와
워킹 디렉터리가
다른 상황



git commit : 파일 변경 사항들 기록

```
MASTER@DESKTOP-49OPGLB MINGW64 /d/ESC/Test (master)
$ git commit -m 'write a greeting message'
[master (root-commit) 8bb5ccf] write a greeting message
1 file changed, 2 insertions(+)
create mode 100644 test1.txt
```

1. 뭐 했는 지를 적기
2. 명령문으로 (첫 글자는 대문자)

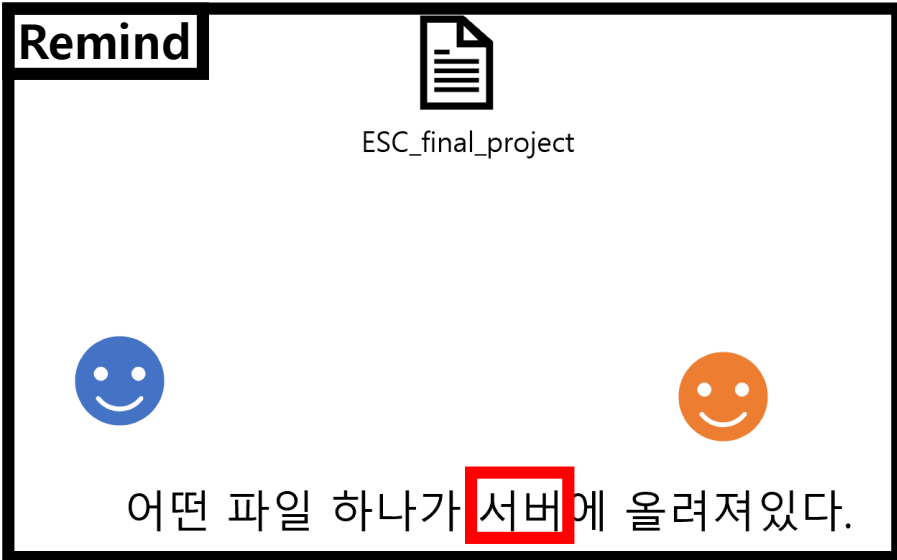


git log: 커밋 히스토리 조회

```
MASTER@DESKTOP-49OPGLB MINGW64 /d/ESC/Test (master)
$ git log
commit 8bb5ccf55f833691ed9031d6e84da8ccbe175082 (HEAD -> master)
Author: JaeHyun Lee <leequant761@gmail.com>
Date: Tue Jan 28 13:52:26 2020 +0900

    write a greeting message
```

TMI : 'HEAD'라는 특수한 포인터는 지금 작업하는 브랜치를 가리킨다.



깃허브는 분산 버전 관리 툴인 깃을 사용하는 프로젝트를 지원하는 웹호스팅 서비스이다.

깃이 설치된 서버 역할을 해주어서 **협업**을 하기 위해서 필요한 존재

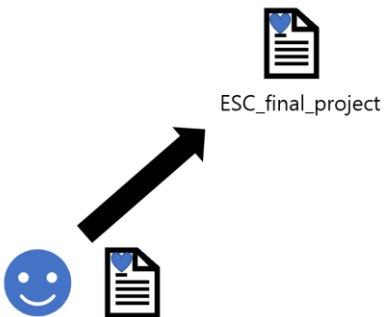
git push: 로컬 → 원격으로 밀어넣기

```
MASTER@DESKTOP-49OPGLB MINGW64 /d/ESC/Test (master)
$ git push
fatal: No configured push destination.
Either specify the URL from the command-line or configure a remote repository using

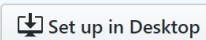
    git remote add <name> <url>

and then push using the remote name

    git push <name>
```



Quick setup — if you've done this kind of thing before



or

HTTPS

SSH

https://github.com/leequant761/Git_ESC.git

Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

...or create a new repository on the command line

```
echo "# Git_ESC" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/leequant761/Git_ESC.git
git push -u origin master
```



...or push an existing repository from the command line

```
git remote add origin https://github.com/leequant761/Git_ESC.git
git push -u origin master
```



...or import code from another repository

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

[Import code](#)

```
MASTER@DESKTOP-490PGLB MTNGW64 /d/ESC/Test (master)
$ git remote add origin https://github.com/leequant761/Git_ESC.git
MASTER@DESKTOP-490PGLB MTNGW64 /d/ESC/Test (master)
$ git push -u origin master
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 257 bytes | 128.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/leequant761/Git_ESC.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
```

1

2

1 : remote repo를 https://~~~ 로 설정하며 origin(별칭)으로 부르겠다.

2 : master 브랜치를 origin 서버에 push!

leequant761 / Git_ESC

Unwatch

1

Star

0

Fork

0

Code

Issues 0

Pull requests 0

Actions

Projects 0

Wiki

Security

Insights

Settings

No description, website, or topics provided.

Edit

Manage topics

2 commits

1 branch

0 packages

0 releases

1 contributor

Branch: master

New pull request

Create new file

Upload files

Find file

Clone or download

leequant761 Create README.md

Latest commit d3c0817 now

README.md

Create README.md

now

test1.txt

Write a greeting message

39 minutes ago

README.md



Git_ESC

안녕하세요

haha

```
print(1)
```

Git branch

프로그램에 새로운 기능을 추가하고 싶다.

프로그램에 새로운 기능을 추가하고 싶다.

master 브랜치는 현재 서비스가 진행 중인 상황이다.

어떻게 하면 될까?

master 브랜치에 영향을 안받는 공간에서 코드를 작성하고 테스트를 해봐야한다.

Solution

<input type="checkbox"/> 이름	수정한 날짜	유형	크기
final_project	2020-01-28 오후 3:13	파일 폴더	
final_project_test	2020-01-28 오후 3:13	파일 폴더	



저장소를 똑같이 복사해서 테스트까지 하면 해결!

Solution



<input type="checkbox"/> 이름	수정한 날짜	유형	크기
final_project	2020-01-28 오후 3:13	파일 폴더	
final_project_test	2020-01-28 오후 3:13	파일 폴더	

용량이 너무 크다면? 100GB짜리 폴더 복사하면 굉장히 오래걸림

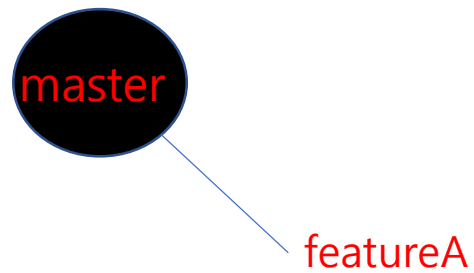
기능추가, 보수, 리팩토링 등등 코드를 변경할 때 마다
이렇게 독립적인 공간을 만들면 매우~ 비효율적임

git pull : remote → local repo

```
MASTER@DESKTOP-49OPGLB MINGW64 /d/ESC/Test (master)
$ git pull
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
From https://github.com/leequant761/Git_ESC
   8e33d46..e39fc0f  master    -> origin/master
Updating 8e33d46..e39fc0f
Fast-forward
 README.md | 6 ++++++
 1 file changed, 6 insertions(+)
 create mode 100644 README.md
```

git branch : 브랜치(독립적인 공간) 생성

```
MASTER@DESKTOP-490PGLB MINGW64 /d/ESC/Test (master)  
$ git branch featureA
```



git checkout : 브랜치 갈아타기

```
MASTER@DESKTOP-490PGLB MINGW64 /d/ESC/Test (master)
$ git branch featureA

MASTER@DESKTOP-490PGLB MINGW64 /d/ESC/Test (master)
$ git checkout featureA
Switched to branch 'featureA'

MASTER@DESKTOP-490PGLB MINGW64 /d/ESC/Test (featureA)
$ |
```

TMI: 커밋 해쉬를 적어서 이전의 스냅샷으로 옮겨갈 수도 있다

git checkout 8bb5ccf55f833691ed9031d6e84da8ccbe175082

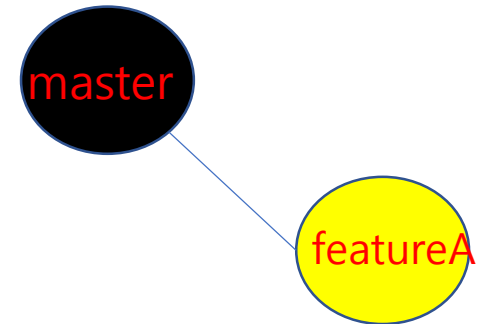
브랜치 A에서 파일 작성하고 커밋

```
MASTER@DESKTOP-49OPGLB MINGW64 /d/ESC/Test (featureA)
$ vim test2.txt

MASTER@DESKTOP-49OPGLB MINGW64 /d/ESC/Test (featureA)
$ cat test2.txt
깃을 공부해봅시다

MASTER@DESKTOP-49OPGLB MINGW64 /d/ESC/Test (featureA)
$ git add test2.txt

MASTER@DESKTOP-49OPGLB MINGW64 /d/ESC/Test (featureA)
$ git commit -m 'Add featureB'
[featureA 0544d32] Add featureB
1 file changed, 1 insertion(+)
create mode 100644 test2.txt
```

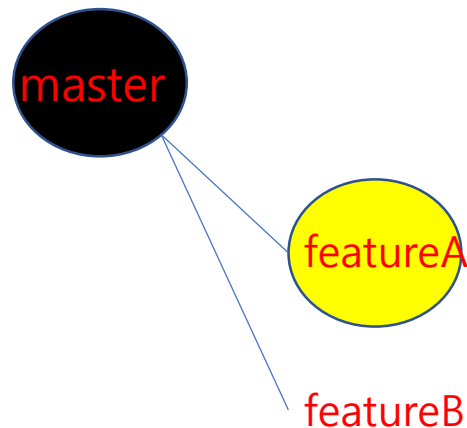


git branch : 브랜치(독립적인 공간) 생성

```
MASTER@DESKTOP-490PGLB MINGW64 /d/ESC/Test (featureA)
$ git checkout master 브랜치를 생성할 때 어느 브랜치에서 생성하는 지 주의!
Switched to branch 'master'
Your branch is up to date with 'origin/master'.

MASTER@DESKTOP-490PGLB MINGW64 /d/ESC/Test (master)
$ git branch featureB

MASTER@DESKTOP-490PGLB MINGW64 /d/ESC/Test (master)
$ git checkout featureB
Switched to branch 'featureB'
```



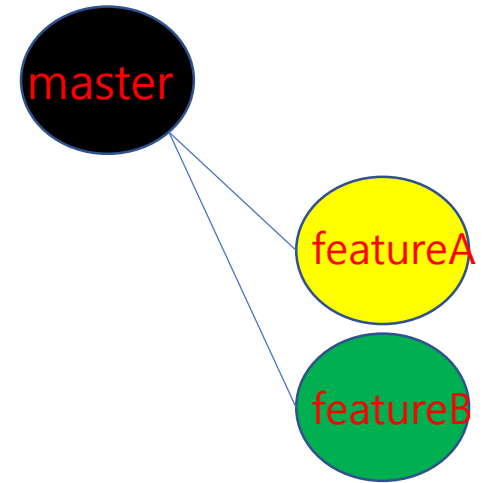
브랜치 B에서 파일 작성하고 커밋

```
MASTER@DESKTOP-49OPGLB MINGW64 /d/ESC/Test (featureB)
$ vim test3.txt

MASTER@DESKTOP-49OPGLB MINGW64 /d/ESC/Test (featureB)
$ cat test3.txt
재 밋 죠 ?

MASTER@DESKTOP-49OPGLB MINGW64 /d/ESC/Test (featureB)
$ git add test3.txt

MASTER@DESKTOP-49OPGLB MINGW64 /d/ESC/Test (featureB)
$ git commit -m 'Add featureB'
[featureB 474238b] Add featureB
1 file changed, 1 insertion(+)
create mode 100644 test3.txt
```



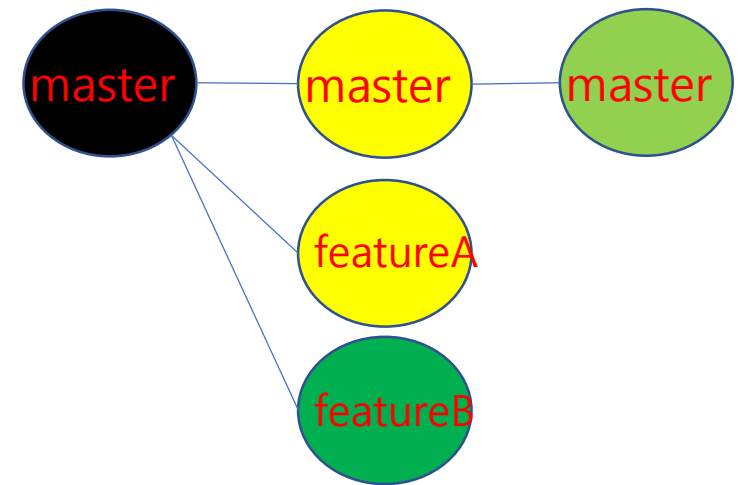
git merge : 현재 HEAD가 가르키고 있는 브랜치에 feature 브랜치를 병합

```
MASTER@DESKTOP-490PGLB MINGW64 /d/ESC/Test (featureB)
$ git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.

MASTER@DESKTOP-490PGLB MINGW64 /d/ESC/Test (master)
$ git merge featureA
Updating e39fc0f..3067811
Fast-forward
 test2.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 test2.txt

MASTER@DESKTOP-490PGLB MINGW64 /d/ESC/Test (master)
$ git merge featureB
Merge made by the 'recursive' strategy.
 test3.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 test3.txt

MASTER@DESKTOP-490PGLB MINGW64 /d/ESC/Test (master)
$ ls
README.md test1.txt test2.txt test3.txt
```



HEAD : 지금 작업하는 브랜치를 가르키는 포인터

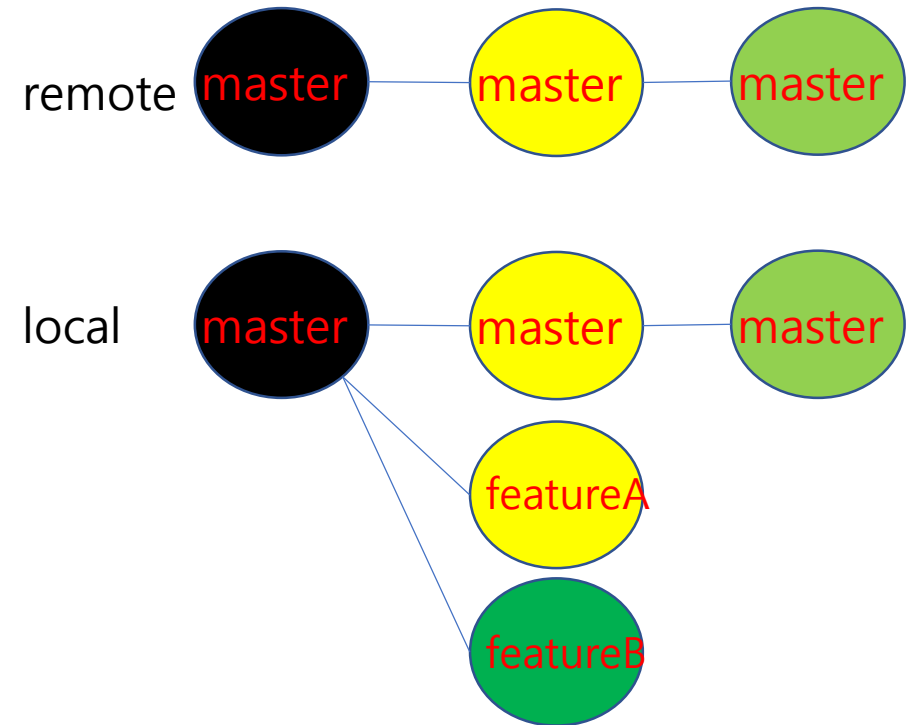
master : git init과 동시에 만들어진 브랜치

fast-forward : 헤드를 옮기기만 해도 되는 경우

git push: 로컬 → 원격으로 밀어넣기

```
MASTER@DESKTOP-490PGLB MINGW64 /d/ESC/Test (master)
$ git push
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 4 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (5/5), 501 bytes | 501.00 KiB/s, done.
Total 5 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 1 local object.
To https://github.com/leequant761/Git_ESC.git
    d3c0817..dfbce69  master -> master

MASTER@DESKTOP-490PGLB MINGW64 /d/ESC/Test (master)
$ |
```



git push (협업할 브랜치)

```
MASTER@DESKTOP-490PGLB MINGW64 /d/ESC/Test (master)
$ git checkout featureA
Switched to branch 'featureA'

MASTER@DESKTOP-490PGLB MINGW64 /d/ESC/Test (featureA)
$ git push origin featureA
Total 0 (delta 0), reused 0 (delta 0)
remote:
remote: Create a pull request for 'featureA' on GitHub by visiting:
remote:   https://github.com/leequant761/Git_ESC/pull/new/featureA
remote:
To https://github.com/leequant761/Git_ESC.git
 * [new branch]      featureA -> featureA
```

Branch: master ▾ New pull request

Switch branches/tags

Find or create a branch...

Branches Tags

✓ master default

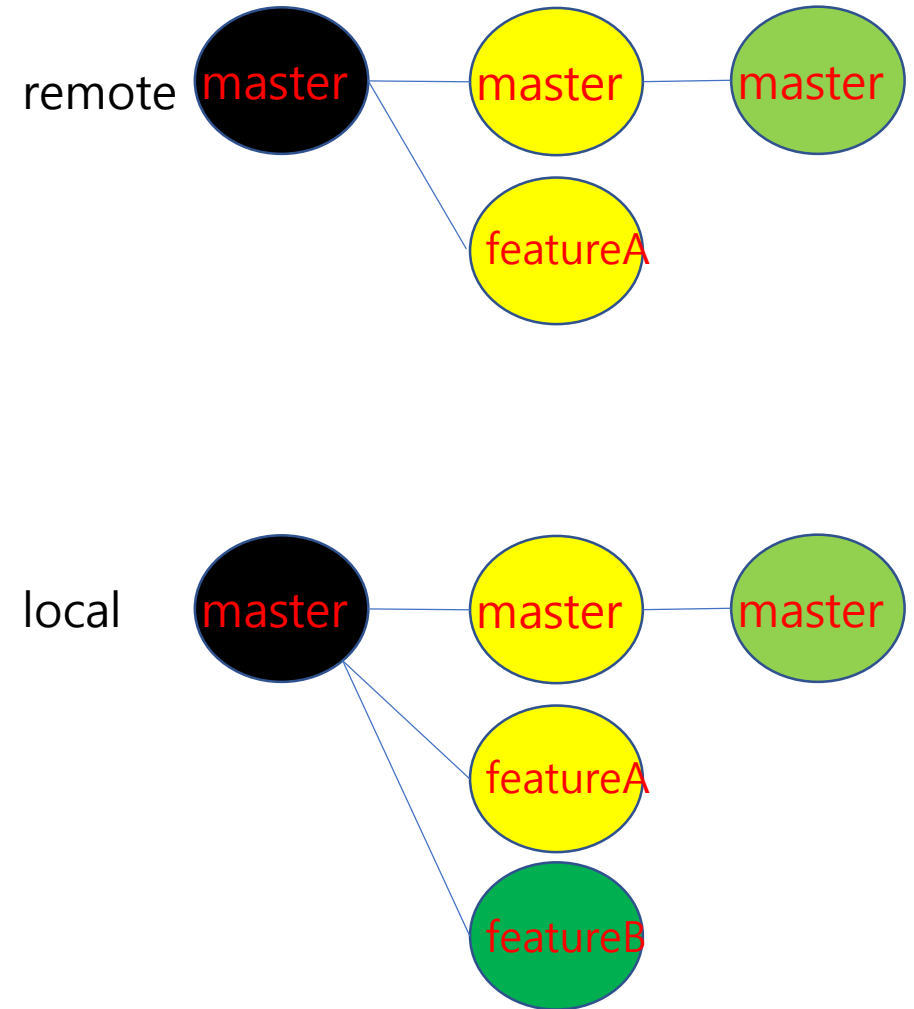
featureA

Create README.md

Write a greeting message

Add featureA

Add featureB



git branch -d

```
MASTER@DESKTOP-490PGLB MINGW64 /d/ESC/Test (master)
$ git branch -d featureA
Deleted branch featureA (was b21f597).

MASTER@DESKTOP-490PGLB MINGW64 /d/ESC/Test (master)
$ git branch -d featureB
Deleted branch featureB (was 474238b).
```



작업이 끝난 **feature** 브랜치는 삭제하도록 하자.

git clone: 서버 → 로컬로 다운로드(with .git/)

```
MASTER@DESKTOP-490PGLB MINGW64 /d/ESC
$ git clone https://github.com/ctgk/PRML
Cloning into 'PRML'...
remote: Enumerating objects: 27, done.
remote: Counting objects: 100% (27/27), done.
remote: Compressing objects: 100% (24/24), done.
remote: Total 3299 (delta 8), reused 10 (delta 3), pack-reused 3272
Receiving objects: 100% (3299/3299), 22.66 MiB | 6.91 MiB/s, done.
Resolving deltas: 100% (2021/2021), done.
```

Git GUI

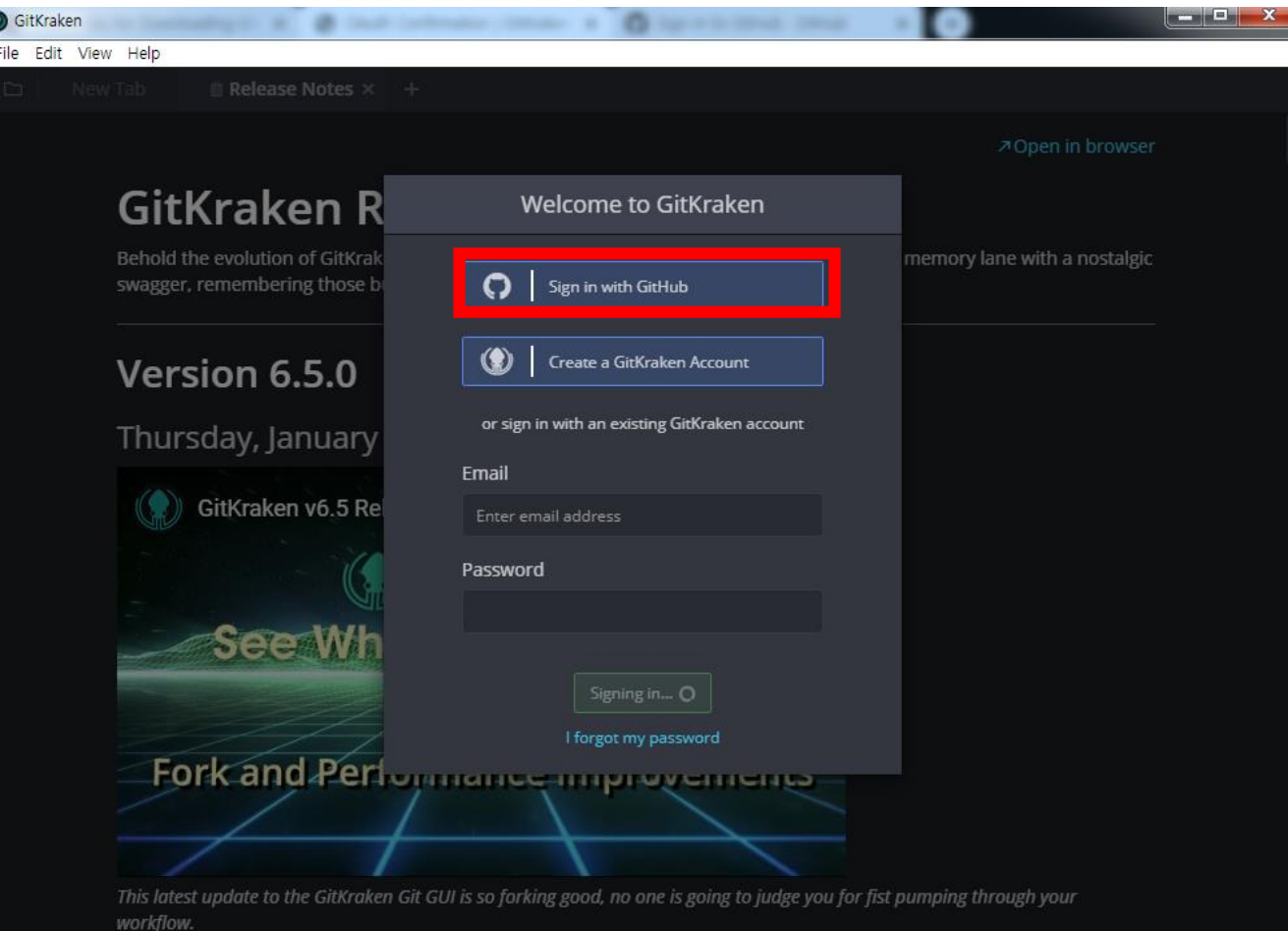
출처 : <https://cupjoo.tistory.com/9>

하고 싶은 건 많은만큼 명령어도 너~무 많다.

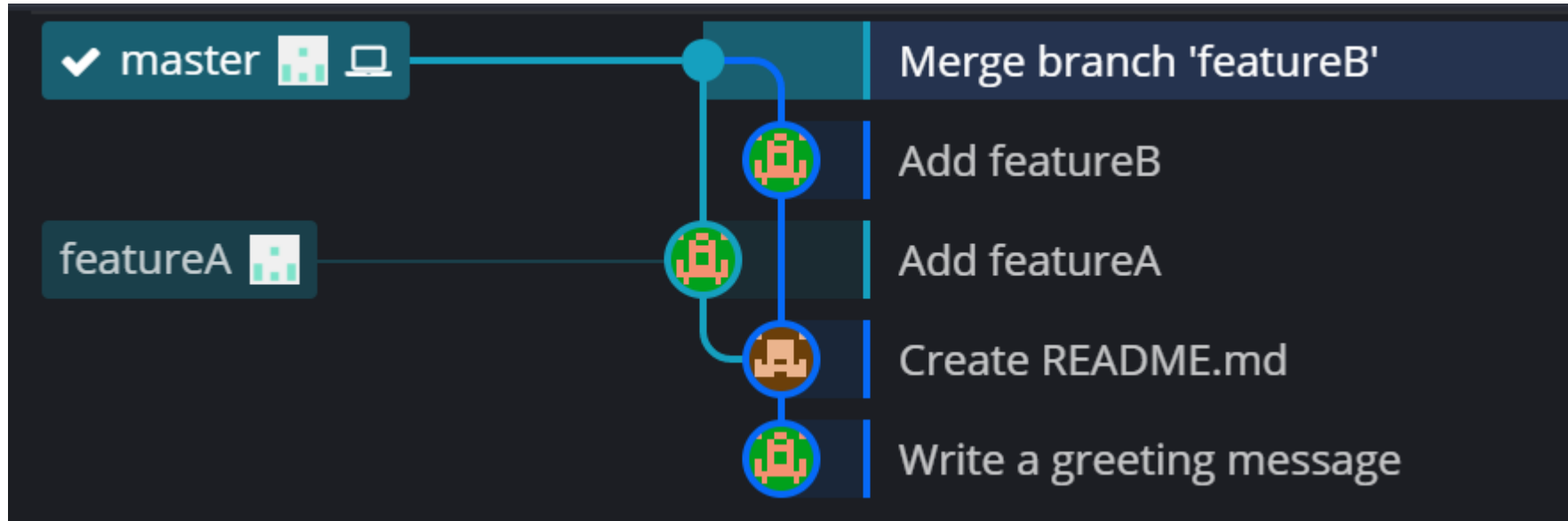
Command Line Interface는 태생적으로 불편하다.



계정 동기화



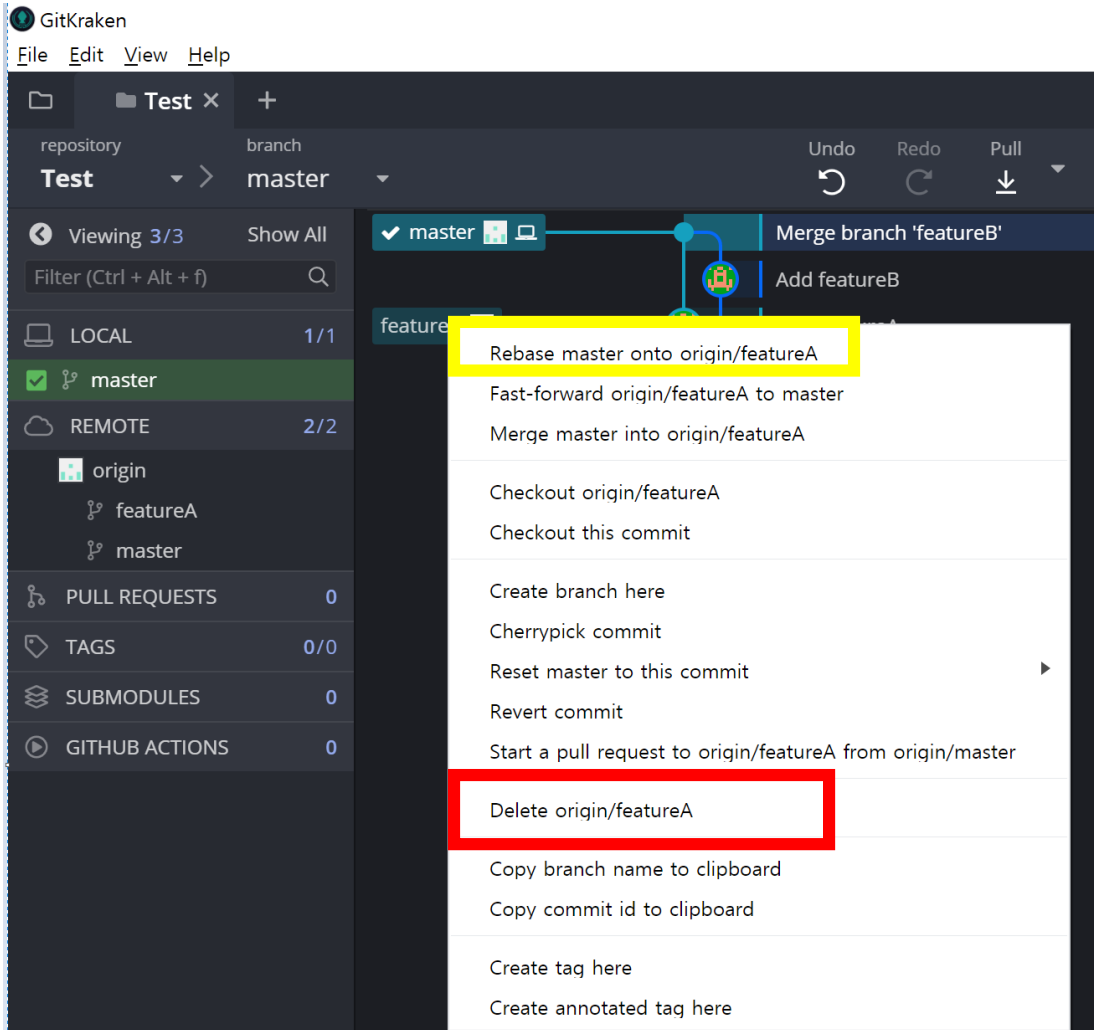
Git Repo에서 열어보기



로컬 repo에서는 feature 브랜치를 다 지웠지만 remote repo에서는 feature 브랜치가 살아있는 상황

삭제하고 싶다!

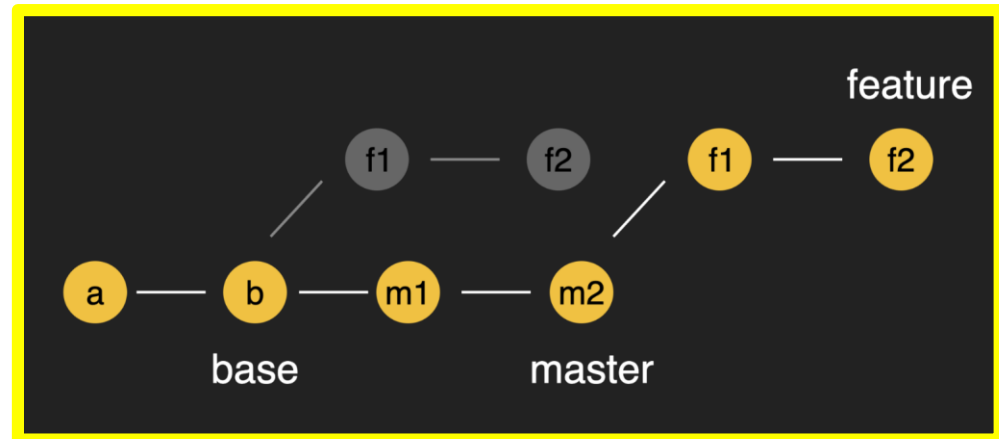
브랜치 삭제



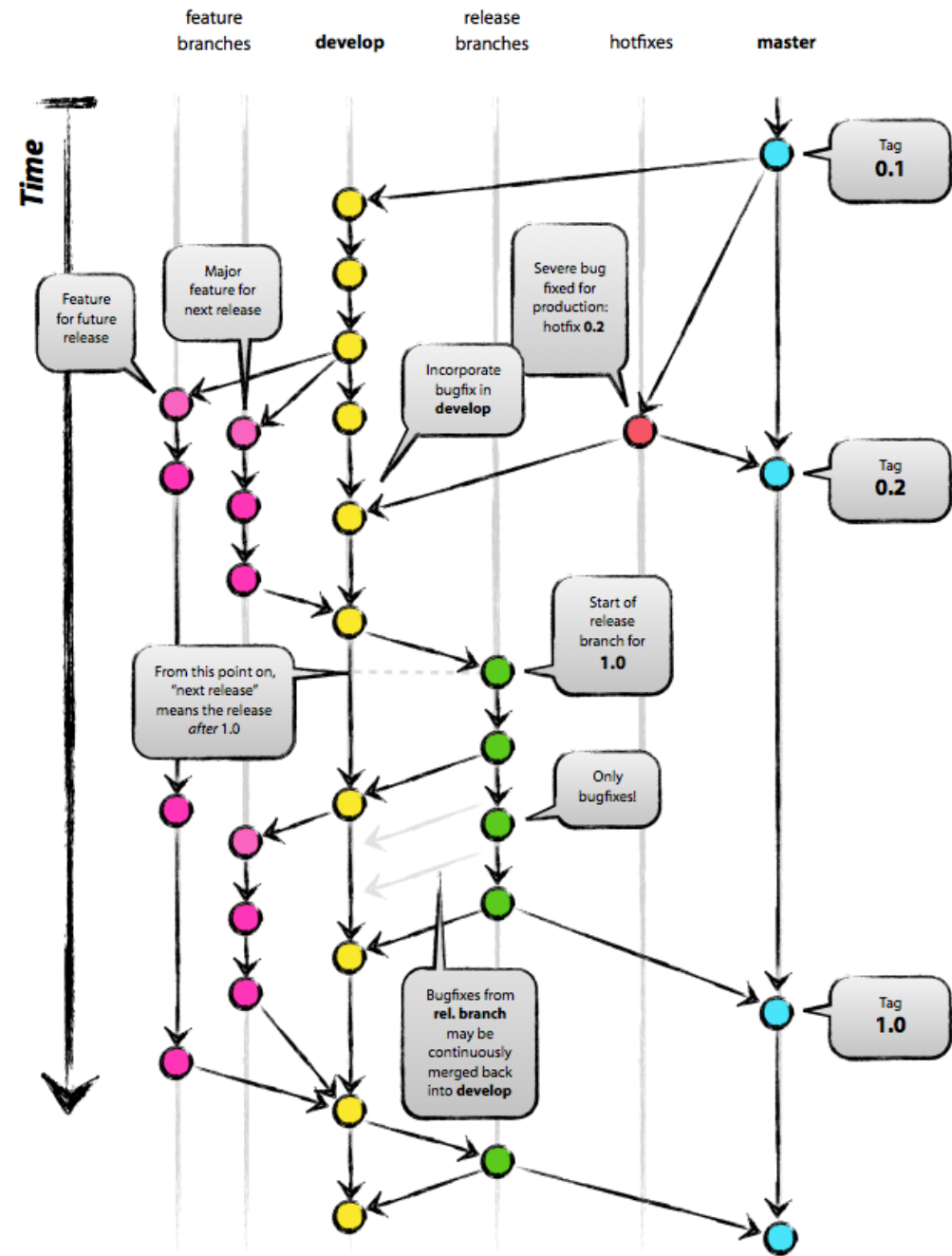
로컬 repo에서는 브랜치를 다 지웠지만 remote repo에서는
브랜치가 살아있는 상황

origin(remote repo 별칭)에서 **브랜치 삭제한다**

정말 상관 없는 정보인데 그림 한방에 설명 가능해서 넣음
TMI : Rebase 는 현재 base가 마스터 브랜치의 마지막 커밋에서
뒤쳐져있다. 이 때 base를 최신으로 맞추고 싶다면!



Git flow



1. master branch

제품으로 출시될 수 있는 브랜치 배포 이력을 관리하기 위해 사용. 즉, 배포 가능한 상태만을 관리한다.

2. develop Branch (평소에는 이 브랜치를 기반으로 협업 개발)

다음 출시 버전을 개발하는 브랜치
feature 개발을 위한 브랜치들을 병합하기 위해 사용.
즉, 모든 기능이 추가되고 버그가 수정되어 배포 가능한 안정적인 상태라면 develop 브랜치를 'master' 브랜치에 병합(merge)한다.

3. Feature branch (기능을 개발하는 브랜치)

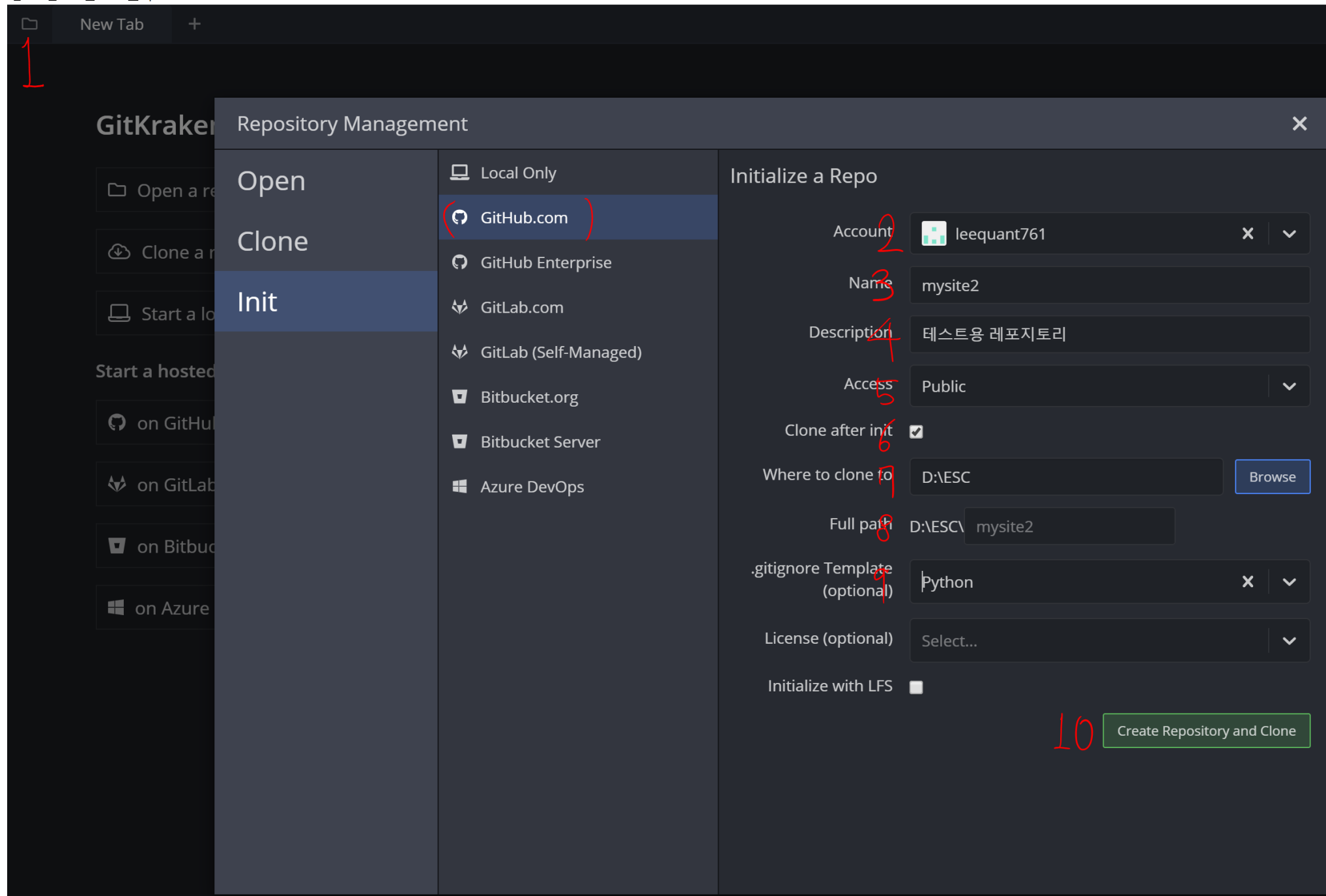
feature 브랜치는 새로운 기능 개발 및 버그 수정이 필요할 때마다 'develop' 브랜치로부터 분기한다.

feature 브랜치에서의 작업은 기본적으로 공유할 필요가 없기 때문에, 자신의 로컬 저장소에서 관리한다.

개발이 완료되면 'develop' 브랜치로 병합(merge)하여 다른 사람들과 공유한다.

<https://gmlwjd9405.github.io/2018/05/11/types-of-git-branch.html>

CLI로 관리 어려워 보임



TMI

9를 Python으로 두면
.gitignore가 생긴다.

이 파일에 담긴 패턴의
파일명은 추적되지 않음

Develop 브랜치 생성하기

The image shows the GitKraken application interface with two overlapping windows. The left window displays the 'File' menu with the 'Preferences...' option highlighted. The right window shows the 'Exit Preferences' dialog box with the 'Git Flow' option selected, indicated by a red '1'. Below the 'Git Flow' option, the 'Commit Template' and 'LFS' options are visible. The right window also shows the 'Git Flow' configuration panel with the 'Branches' section. The 'Master' branch is set to 'master', and the 'Develop' branch is set to 'develop'. The 'Prefixes' section shows 'Feature' as 'feature/' and 'Release' as 'release/'. At the bottom of the 'Branches' section, there is a green button labeled 'Initialize Git Flow', which is highlighted with a red '2'.

GitKraken

File Edit View Help

New Tab Ctrl+T

Clone Repo Ctrl+N

Init Repo Ctrl+I

Open Repo Ctrl+O

Open Terminal Alt+T

Open in File Manager Alt+O

Preferences... Ctrl+콤마

Check for Update (Last checked an hour ago)

Sign into a Different Account

Exit Alt+F4

Tags 0/0

Submodules 0

GitHub Actions 0

GitKraken

File Edit View Help

mysite2

Exit Preferences

lee JaeHyun Lee leequant761@gmail.com

General

Profiles

Authentication

UI Preferences

GPG Preferences

Editor Preferences

Repo-Specific Preferences

mysite2

Git Flow 1

Commit Template

LFS

Git Flow

Branches

Master master

Develop develop

Prefixes

Feature feature/

Release release/

Hotfix hotfix/

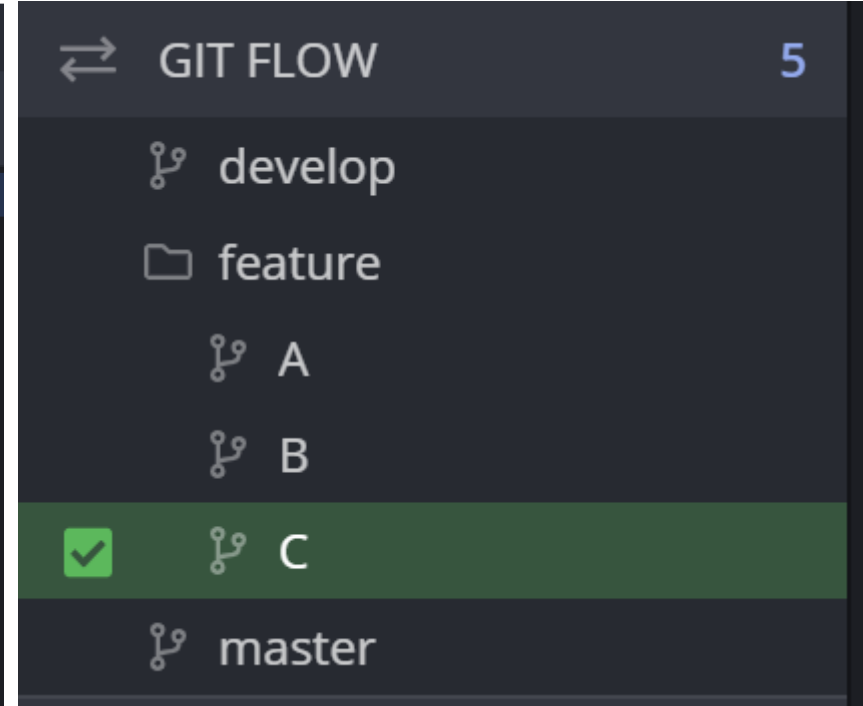
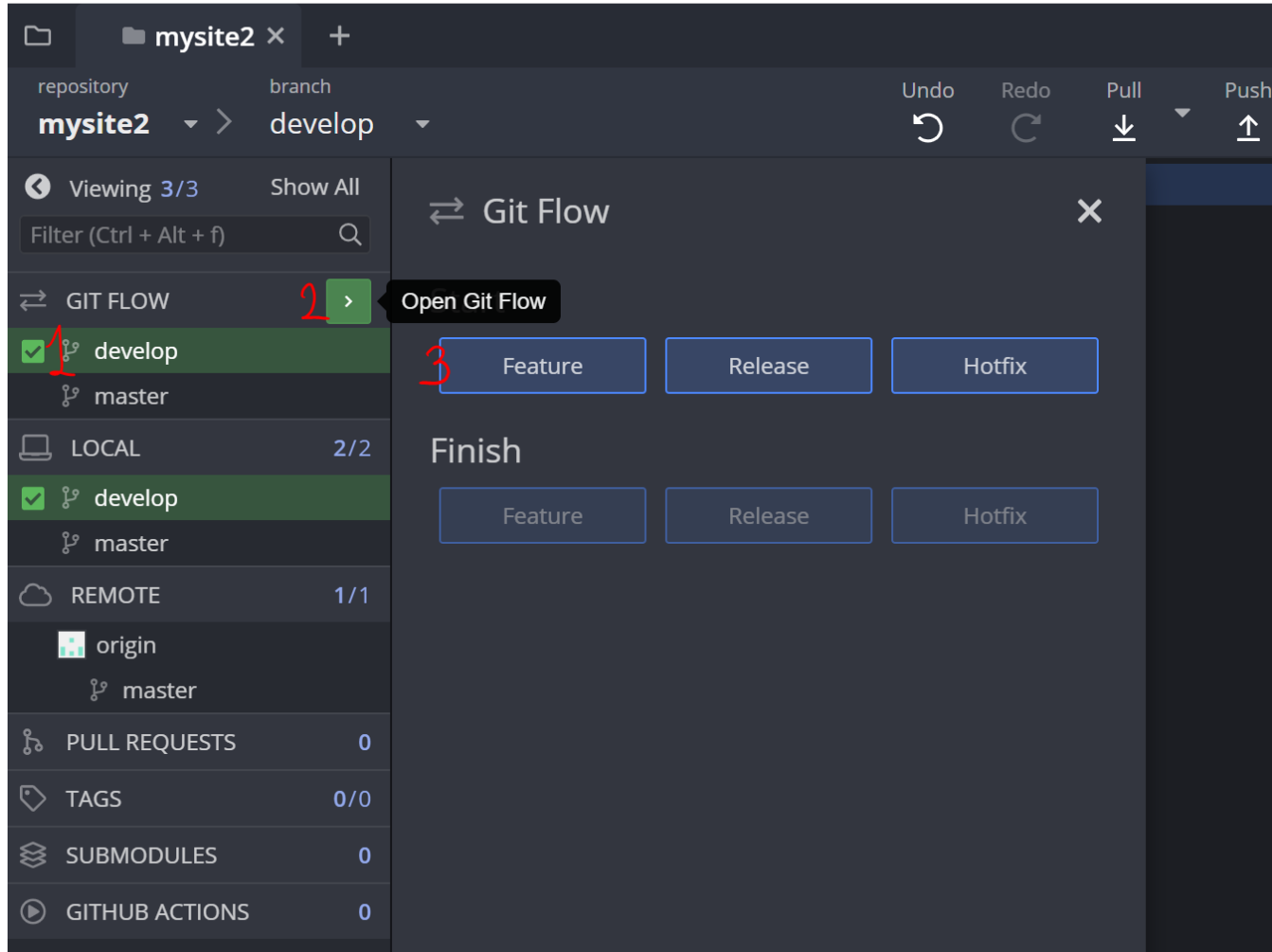
Version Tag

Initialize Git Flow 2

Develop 브랜치에서 Feature 브랜치 생성

GitKraken

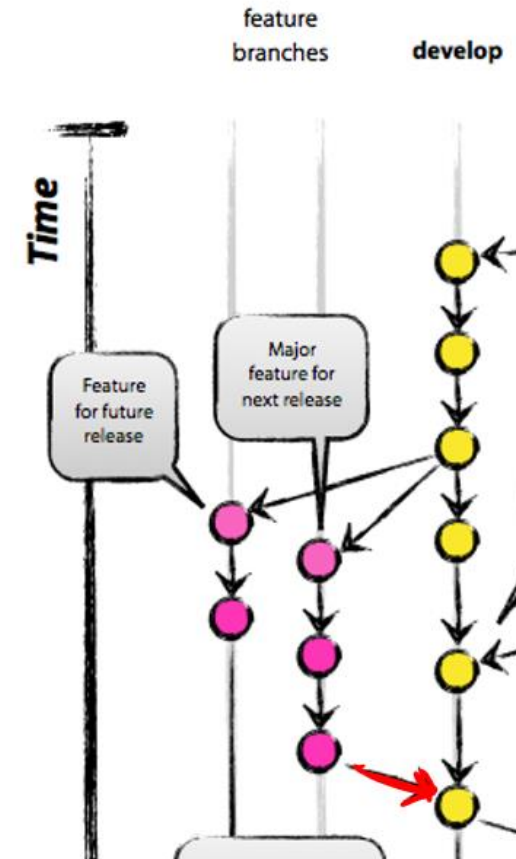
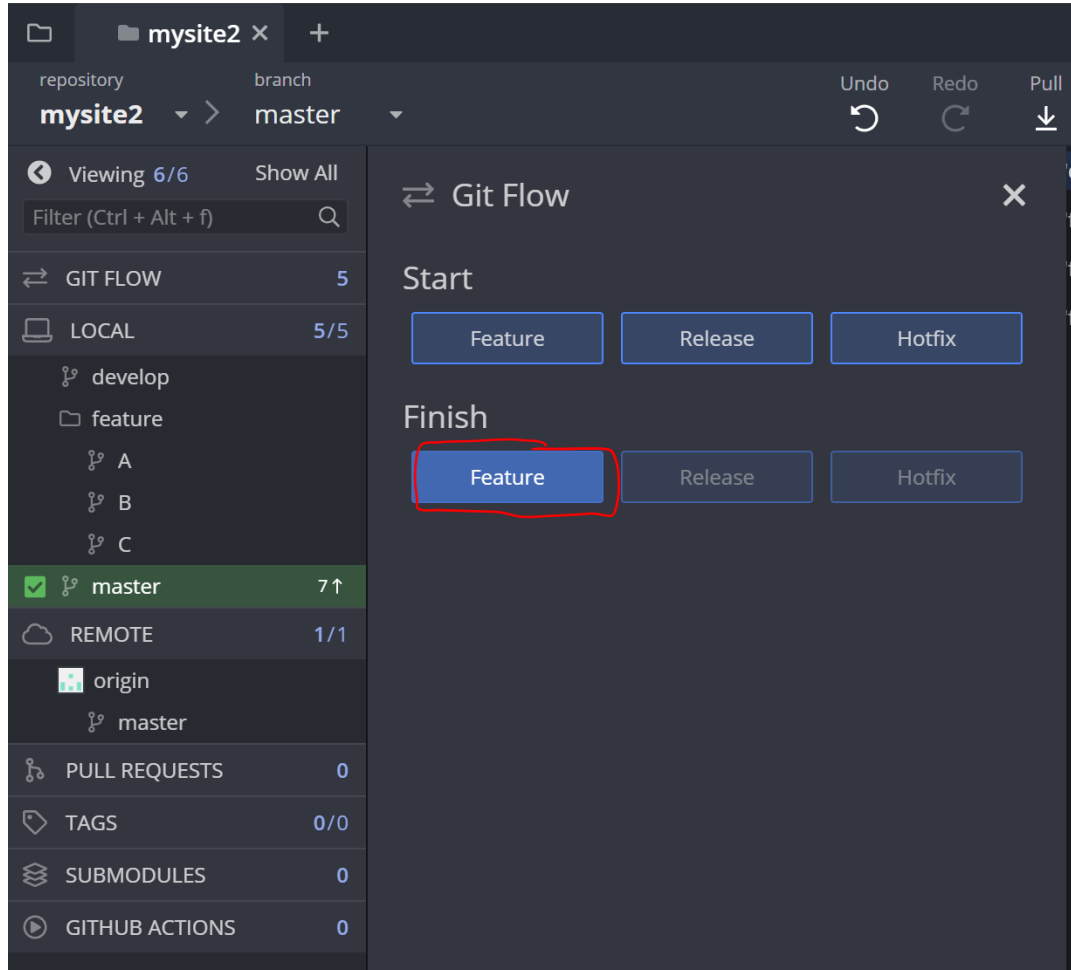
File Edit View Help

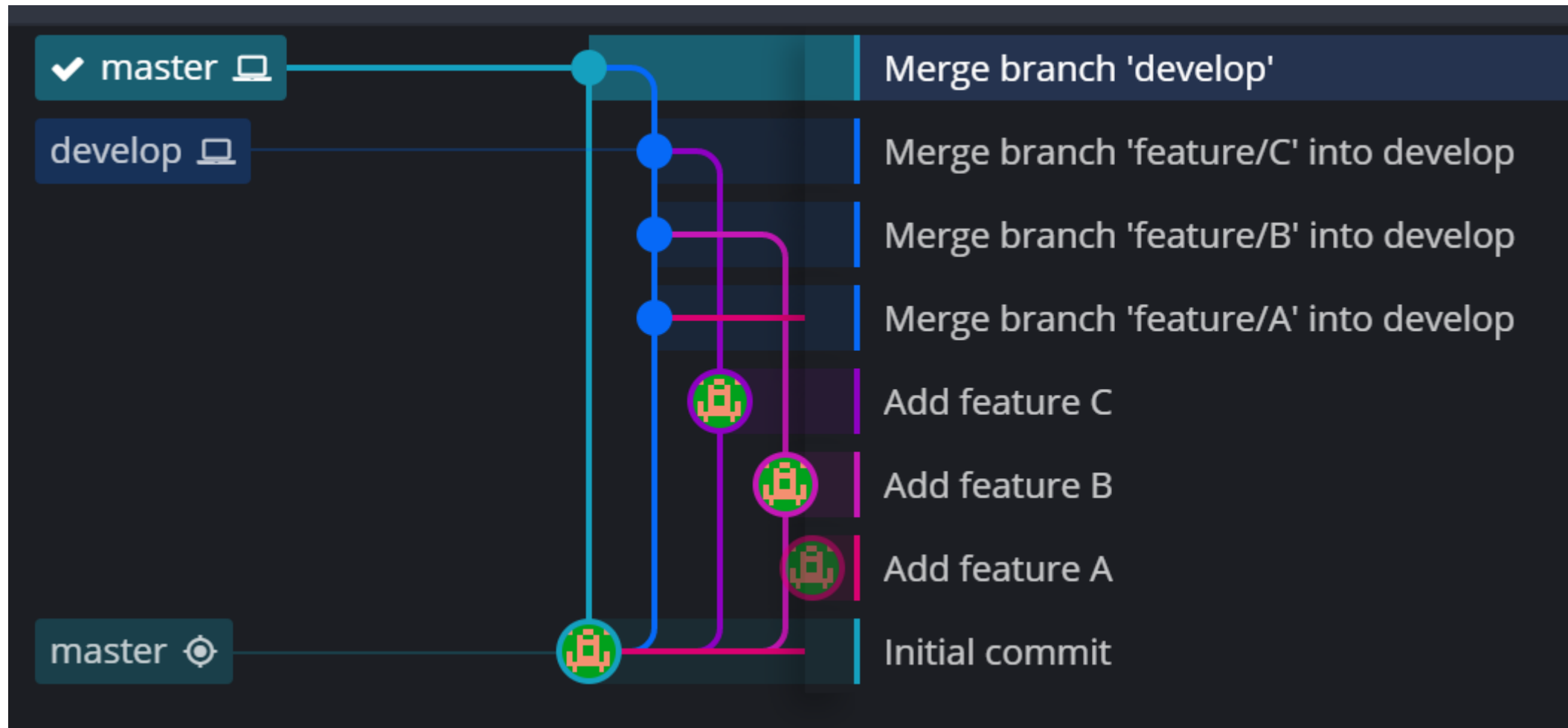


Feature 브랜치 머지하고 삭제하기

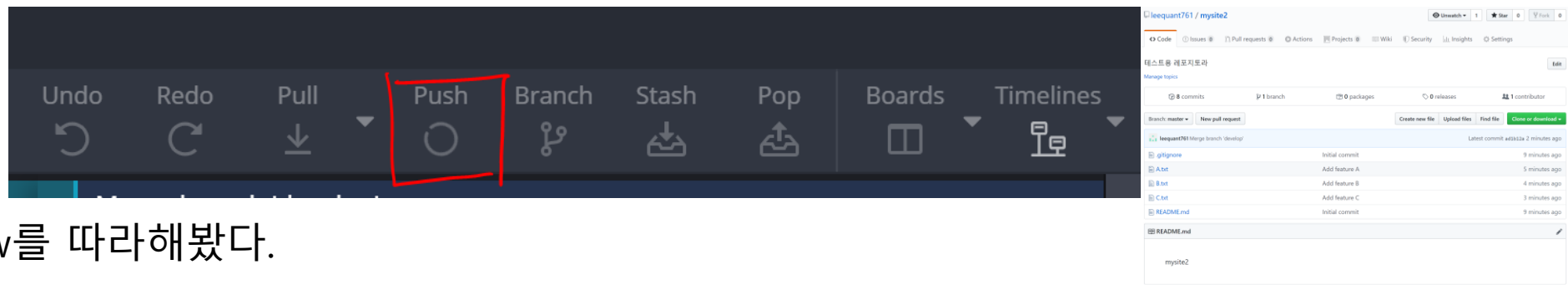
GitKraken

File Edit View Help





master 브랜치
develop 브랜치
feature 브랜치



를 생성해서 Git Flow를 따라해봤다.

사실 혼자서 작업하는 데는 여기까지 과정만 따라해도 별 문제가 없다.

하지만 여러 개발자들이 동시에 Develop 또는 Master 브랜치에 접근하는 경우
코드가 엄청나게 꼬일 수 있다!

사실 혼자서 작업하는 데는 여기까지 과정만 따라해도 별 문제가 없다.

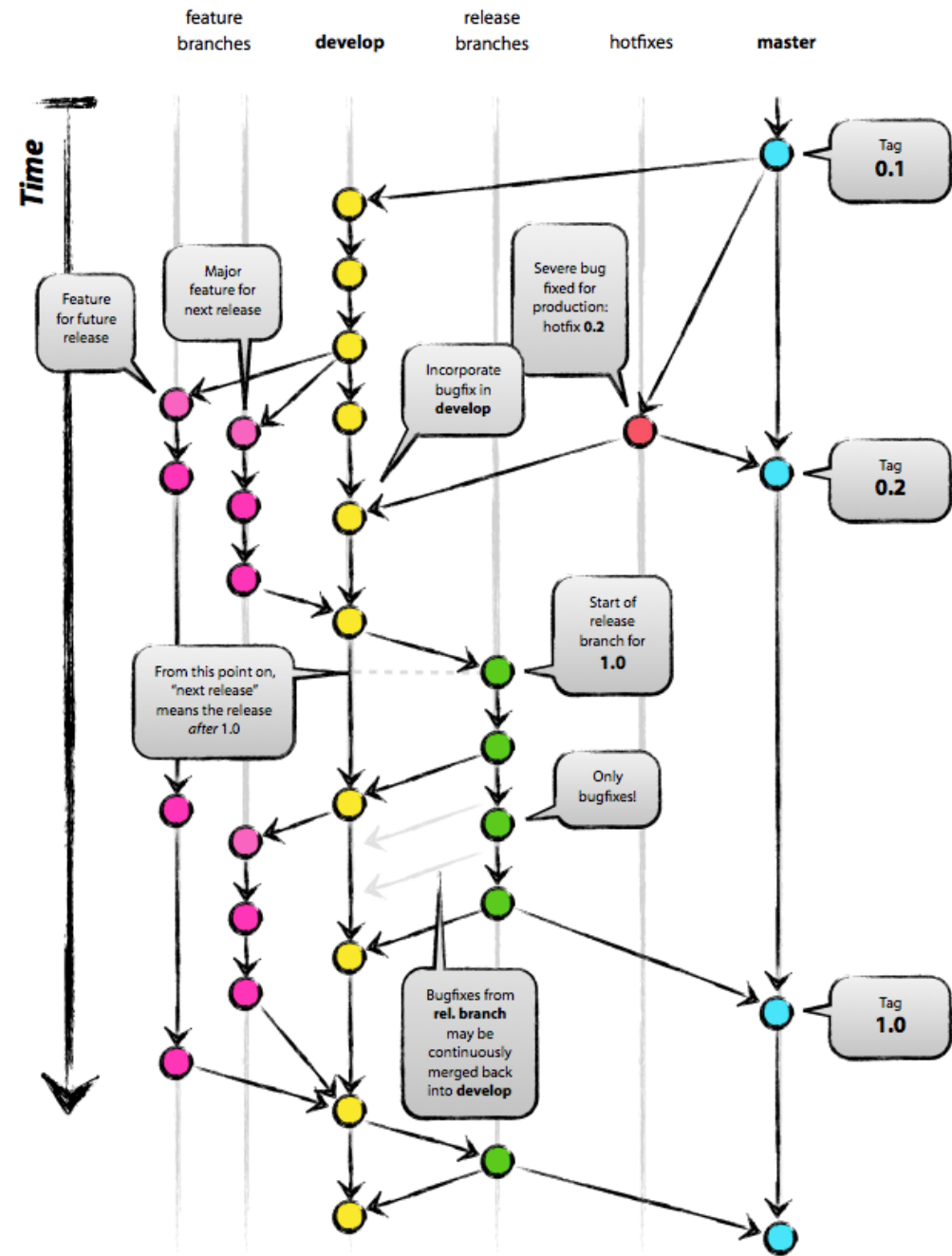
하지만 여러 개발자들이 동시에 Develop 또는 **Master 브랜치**에 접근하는 경우
코드가 엄청나게 꼬일 수 있다!

따라서 우리는 각 Branch에 대해 Pull Request를 날리면

허가 받은 관리자만이 해당 Request를 허가하고 Merge를 진행
할 수 있게 설정해야 한다.

브랜치 권한 수정하기

Remind



1. master branch

제품으로 출시될 수 있는 브랜치 배포 이력을 관리하기 위해 사용. 즉, 배포 가능한 상태만을 관리한다.

2. develop Branch (평소에는 이 브랜치를 기반으로 협업 개발)

다음 출시 버전을 개발하는 브랜치
feature 개발을 위한 브랜치들을 병합하기 위해 사용.
즉, 모든 기능이 추가되고 버그가 수정되어 배포 가능한 안정적인 상태라면 develop 브랜치를 'master' 브랜치에 병합(merge)한다.

3. Feature branch (기능을 개발하는 브랜치)

feature 브랜치는 새로운 기능 개발 및 버그 수정이 필요할 때마다 'develop' 브랜치로부터 분기한다.

feature 브랜치에서의 작업은 기본적으로 공유할 필요가 없기 때문에, 자신의 로컬 저장소에서 관리한다.

개발이 완료되면 'develop' 브랜치로 병합(merge)하여 다른 사람들과 공유한다.

<https://gmlwjd9405.github.io/2018/05/11/types-of-git-branch.html>

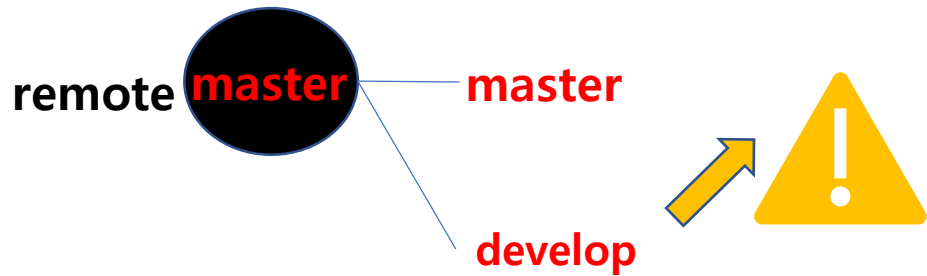
협업을 위해서

깃허브에

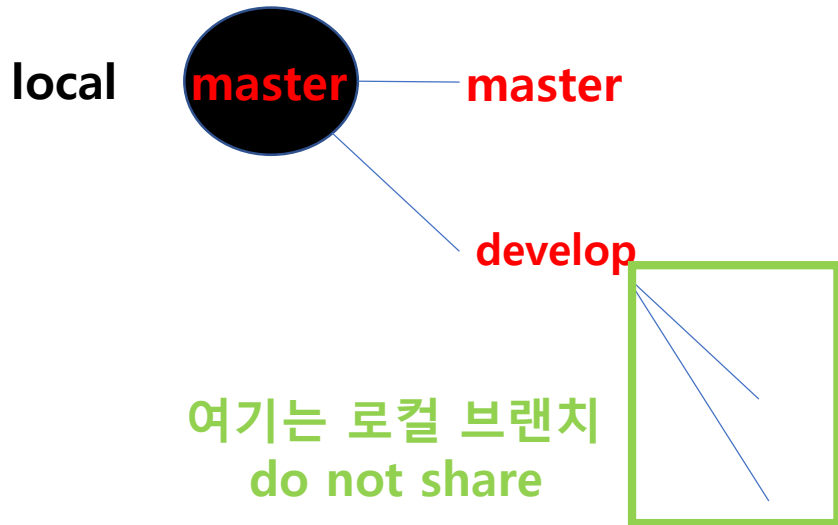
master branch와 develop branch를

두면 되겠구나!

이런 플로우로 작업을 하고 싶다!

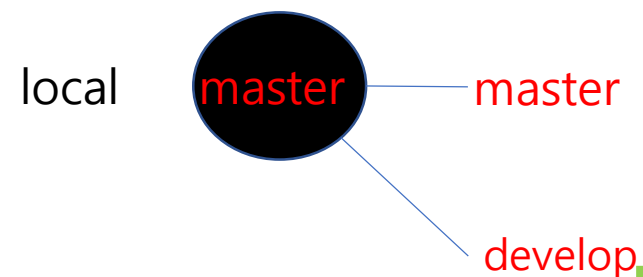
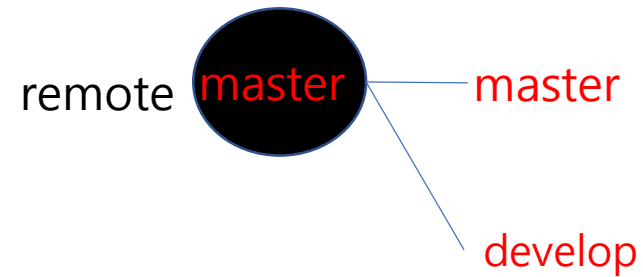


master commit을 위해
Develop을 merge를 하고 싶을 때는 허가 필요

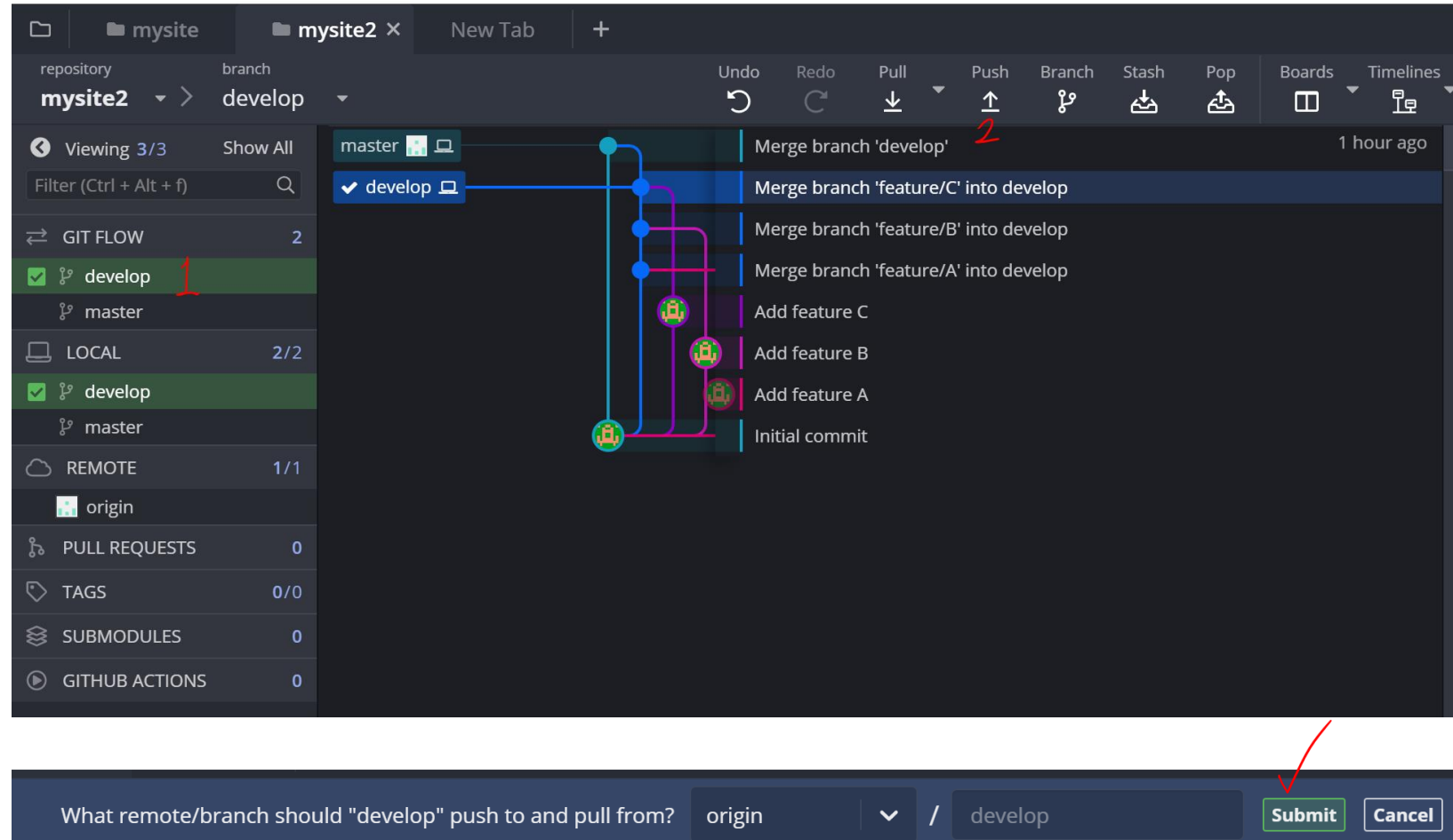
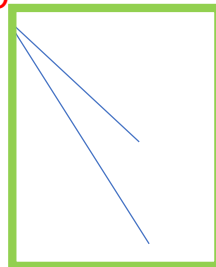


이는 수많은 협업 플로우 중에 하나의 예시일 뿐! 오해하지말자!

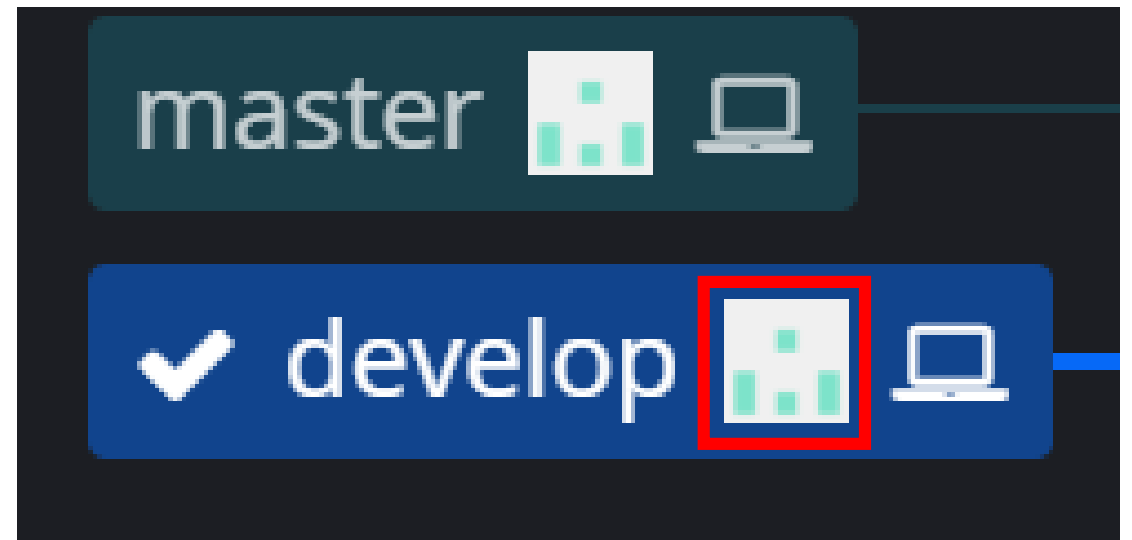
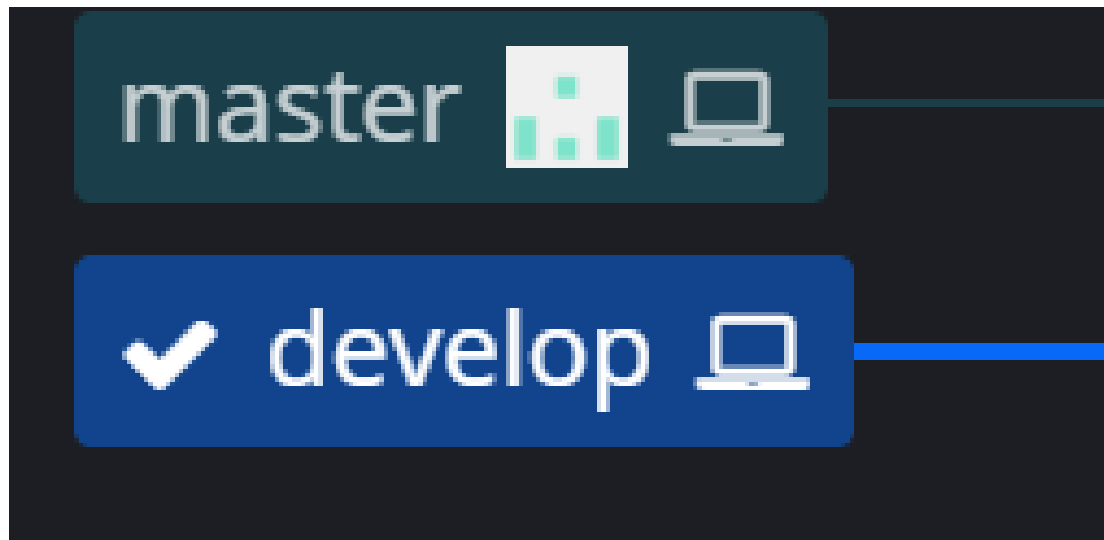
우선 서버에 협업을 위해 develop 브랜치를 origin으로 올리자!



여기는 로컬 브랜치
do not share



Before & After





master 브랜치에 커밋 쌓는 것은 함부로 할 수 없다! 허락을 받자!

leequant761 / mysite2

Unwatch 1 Star 0 Fork 0

Code Issues 0 Pull requests 0 Actions Projects 0 Wiki Security Insights Settings 1

Options Collaborators Branches 2 Webhooks Notifications Integrations & services Deploy keys Secrets Actions

Moderation Interaction limits

Default branch

The default branch is considered the "base" branch in your repository, against which all pull requests and code commits are automatically made, unless you specify a different branch.

master Update

Branch protection rules

Define branch protection rules to disable force pushing, prevent branches from being deleted, and optionally require status checks before merging. New to branch protection rules? [Learn more.](#)

No branch protection rules defined yet.

Add rule 5

Branch protection rule

Branch name pattern

master 4

Protect matching branches

- ☒ **Require pull request reviews before merging**
When enabled, all commits must be made to a non-protected branch and submitted via a pull request with the required number of approving reviews and no changes requested before it can be merged into a branch that matches this rule.
Required approving reviews: 1
- ☐ **Dismiss stale pull request approvals when new commits are pushed**
New reviewable commits pushed to a matching branch will dismiss pull request review approvals.
- ☐ **Require review from Code Owners**
Require an approved review in pull requests including files with a designated code owner.

오너는 바로 커밋을 할 수도 있지만 마찬가지로 Pull request로 하길 바람

옆자리 학회원의 mysite2 깃허브에 초대를 받아보세요~

Options

Collaborators 1

Branches


Webhooks

Notifications

Integrations & services

Collaborators

Push access to the repository

 leequant761 ×

Search by username, full name or email address
You'll only be able to find a GitHub user by their email address if they've chosen to list it publicly. Otherwise, use their username instead.

2

3 Add collaborator

옆자리 학회원의 repo에 들어가서 master에 커밋을 시도해보자!

leequant761 / mysite2

Watch 1 Star 0 Fork 0

Code Issues 0 Pull requests 0 Actions Projects 0 Wiki Security Insights

테스트용 레포지토리

8 commits 2 branches 0 packages 0 releases 1 contributor

Branch: master New pull request

Create new file Upload files Find file Clone or download

leequant761 Merge branch 'develop' Latest commit ad1b12a 3 hours ago

.gitignore	Initial commit	4 hours ago
A.txt	Add feature A	4 hours ago
B.txt	Add feature B	3 hours ago
C.txt	Add feature C	3 hours ago
README.md	Initial commit	4 hours ago

README.md

mysite2

mysite2 / D.txt Cancel

Edit new file Preview

1 D 기능 추가

Commit new file

Add feature D

Add an optional extended description...



☐ You can't commit to `master` because it is a [protected branch](#).

☒ Create a **new branch** for this commit and start a pull request. [Learn more about pull requests](#).

subaccount-jh-patch-1

Commit new file Cancel

응 안돼

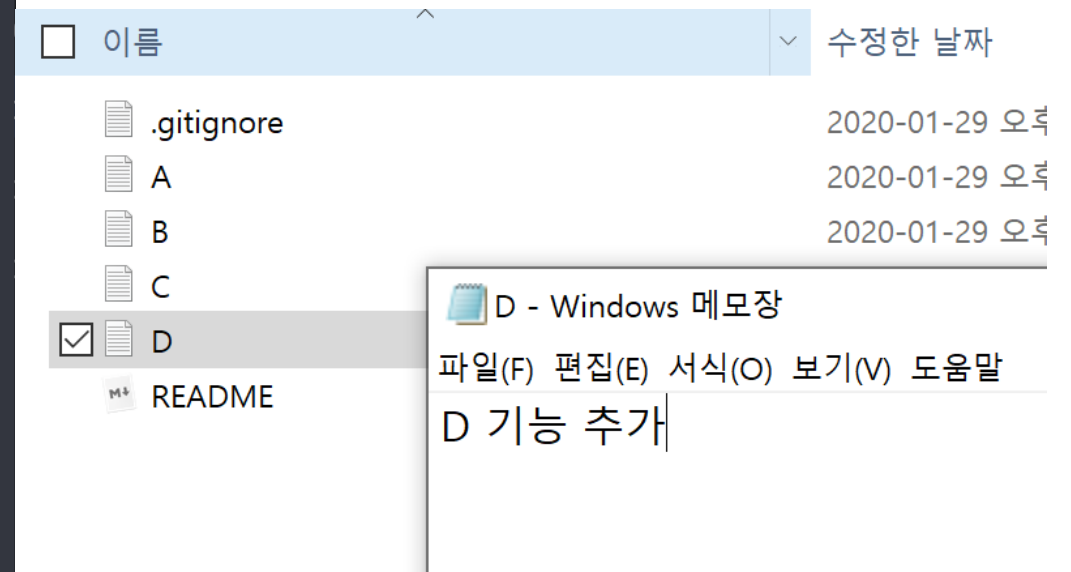
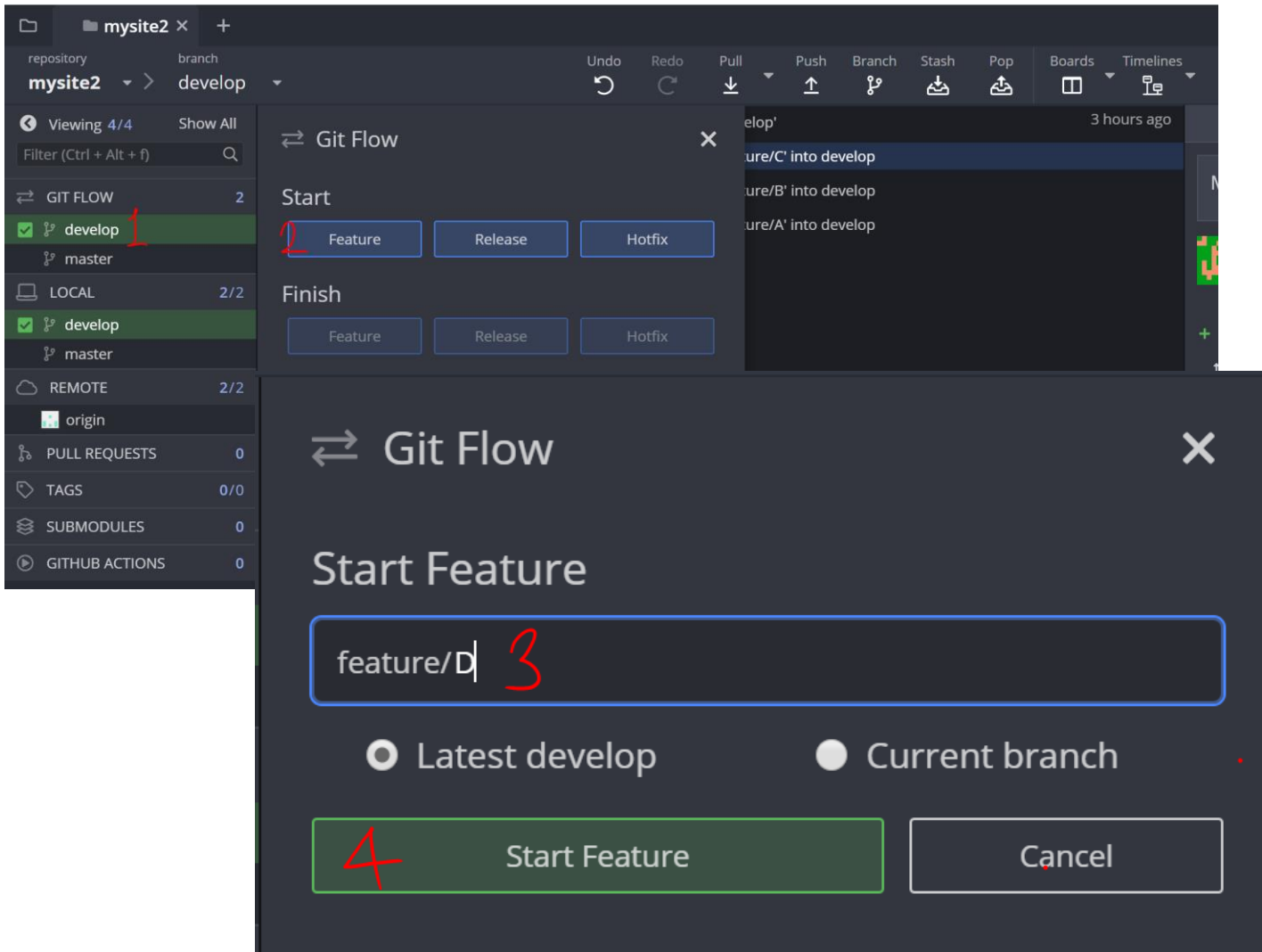
- ☐  You can't commit to `master` because it is a [protected branch](#).
- ☒  Create a new branch for this commit and start a pull request. [Learn more about pull requests.](#)

 `subaccount-jh-patch-1`

로컬에서 개발을 해서 옆자리 학회원의 master에 커밋을 해보자

우선 옆자리 학회원의 깃허브 주소를 clone하자!

로컬에서 개발을 해서 옆자리 학회원의 master에 커밋을 해보자



로컬에서 협업계정으로 작업을 해보자

5를 누르고 커밋까지!

The screenshot displays the Visual Studio Code Git interface with the following components:

- Top Bar:** Includes icons for Undo, Redo, Pull, Push, Branch, Stash, Pop, Boards, and Timelines. The user profile 'lee' (leequant761@gmail.com) is shown on the right.
- Commit History:** A list of recent commits including 'Merge branch 'develop'', 'Merge branch 'feature/C' into develop', 'Merge branch 'feature/B' into develop', 'Merge branch 'feature/A' into develop', and 'Add feature C'. A red circle highlights the 'Add feature C' commit.
- Commit Details:** A panel showing the selected commit '1 file change in working directory' with commit hash '0caa08' and author 'JaeHyun Lee'.
- Git Flow Panel:** A sidebar on the left with a 'Git Flow' section containing 'Start' (Feature, Release, Hotfix) and 'Finish' (Feature, Release, Hotfix) buttons. A red circle highlights the 'Feature' button in the 'Finish' section.
- Repository Explorer:** A sidebar on the right showing the 'mysite2' repository with branches 'develop' and 'master'. The 'master' branch is selected and highlighted with a red circle.
- Context Menu:** A right-click menu is open over the 'master' branch, listing actions such as 'Pull (fast-forward if possible)', 'Push', 'Set Upstream', 'Rebase master onto develop', 'Merge develop into master' (highlighted with a red circle), 'Checkout develop', 'Create branch here', and 'Cherrypick commit'.

엥? 로컬 마스터까지 머지는 되네?

그럼 Push하면 어떻게 될까?

엥? 로컬 마스터까지 머지는 되네?

그럼 Push하면 어떻게 될까?



Push Failed: master



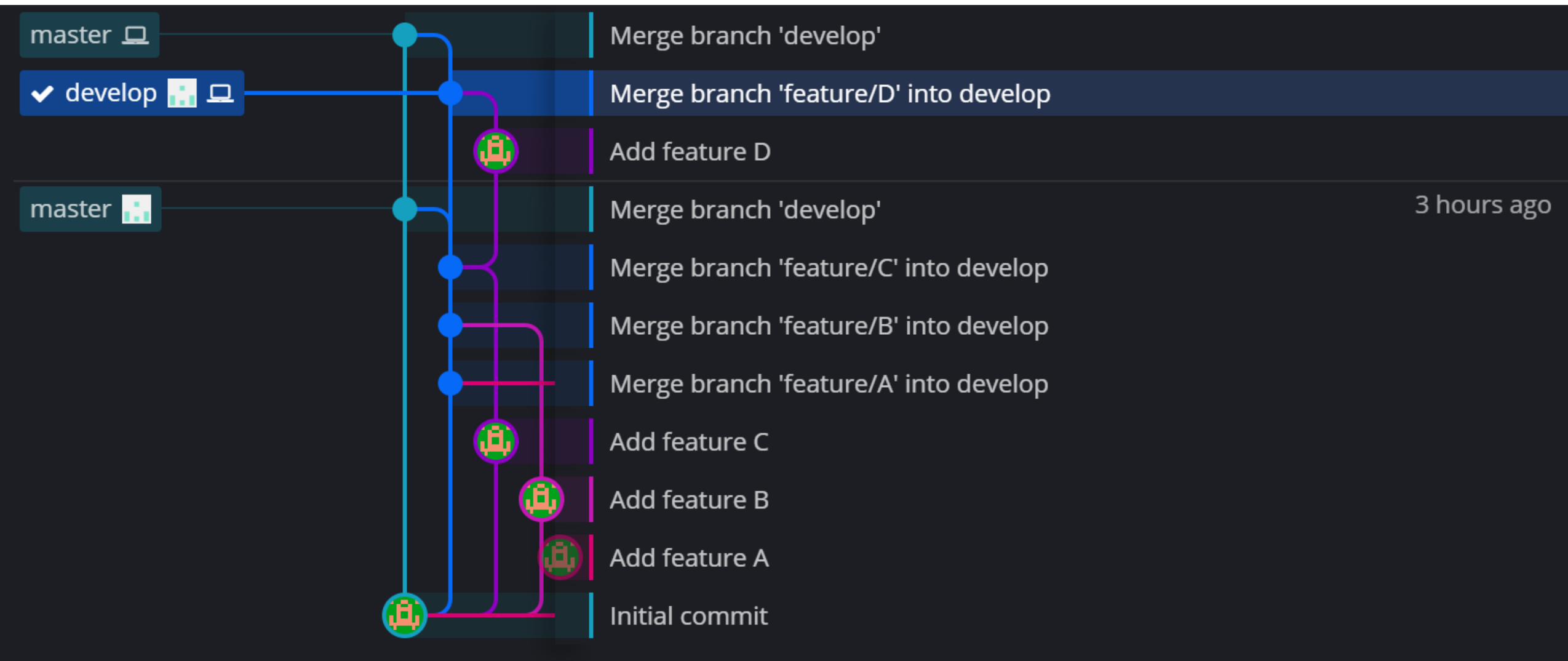
Push failed on refs/heads/master:
protected branch hook declined

origin/master는 보호된 상태구나~

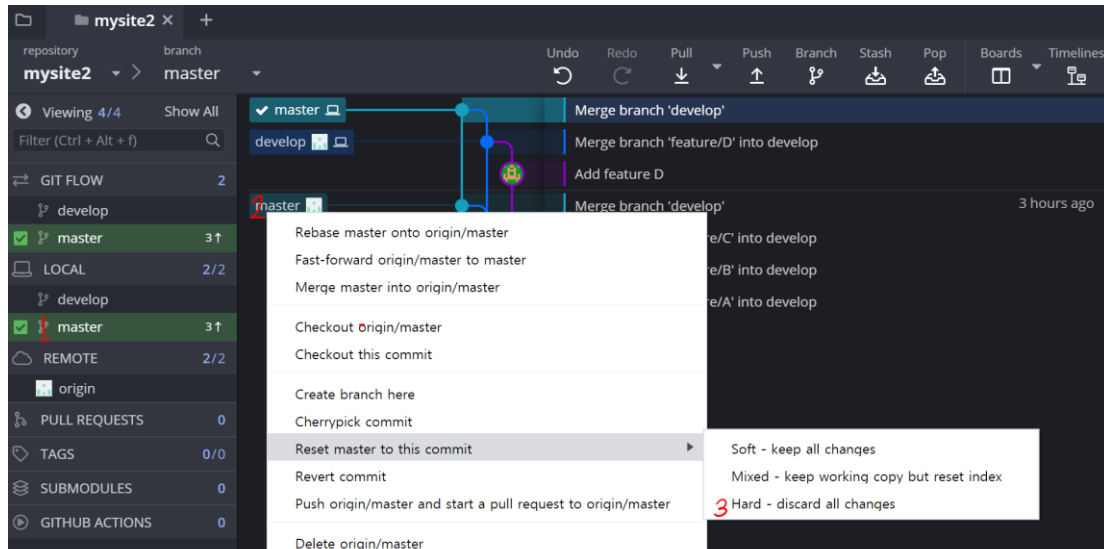
origin/master에 어떻게 머지함?

Pull Request!

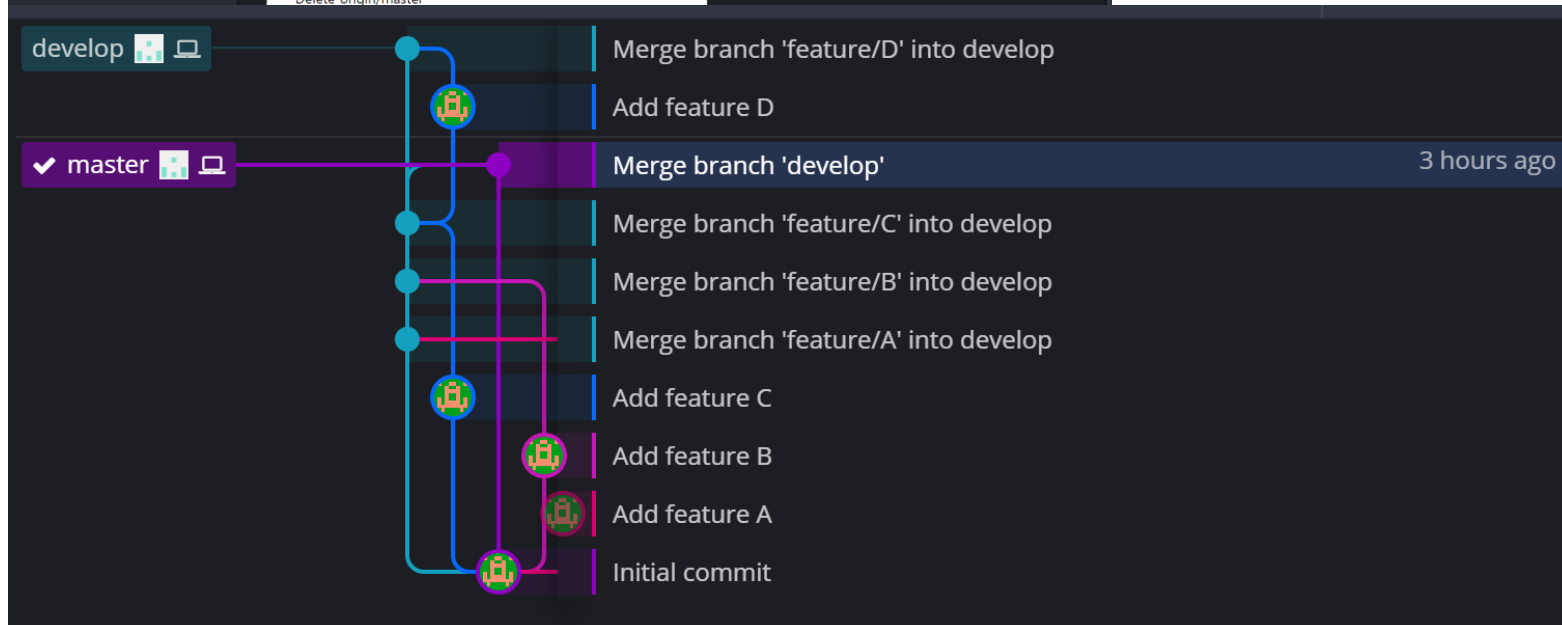
Step 1 : local의 develop브랜치를 서버로 push



Step 2 : 로컬 master브랜치에서 reset으로 방금 커밋을 취소하자

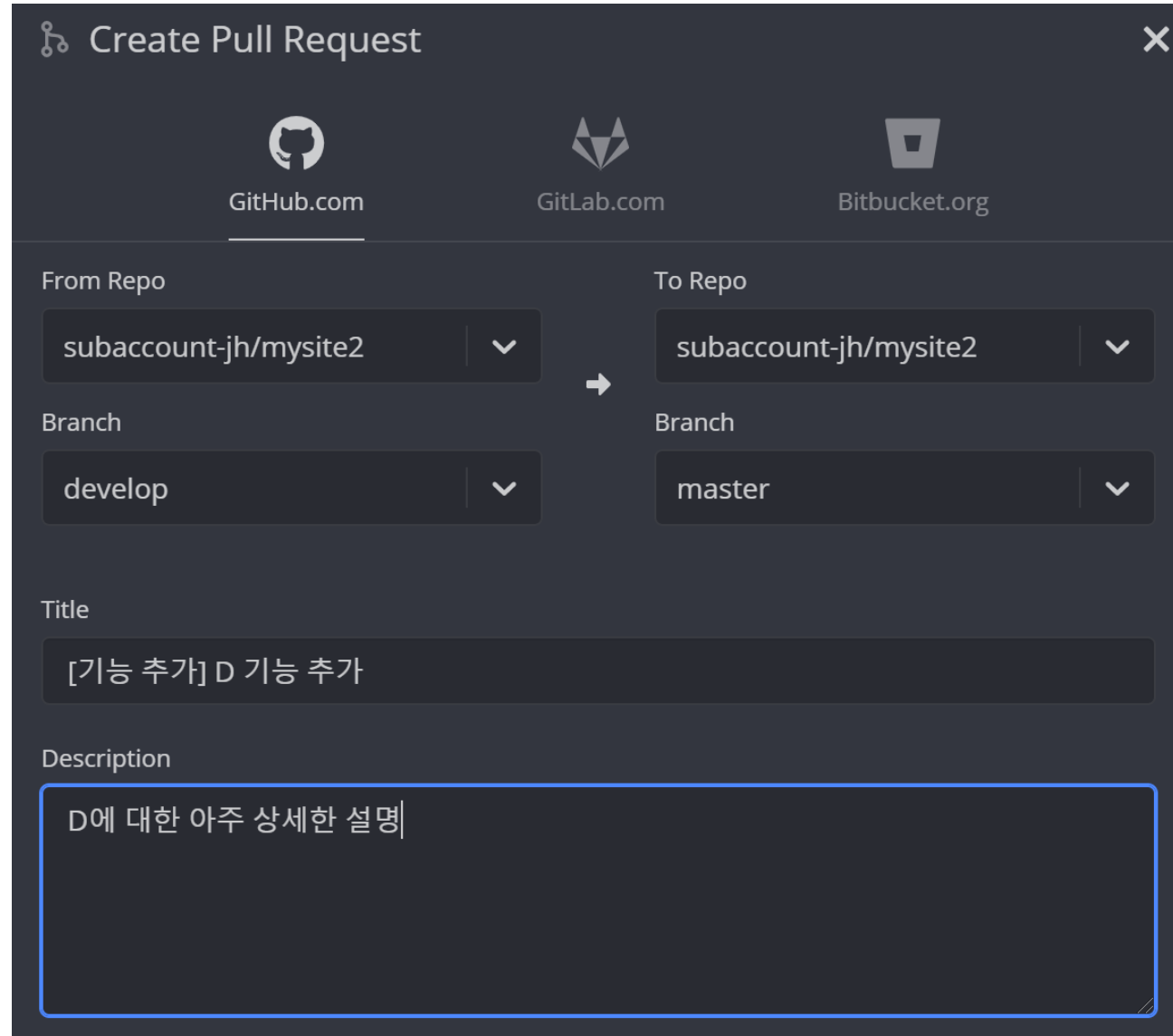
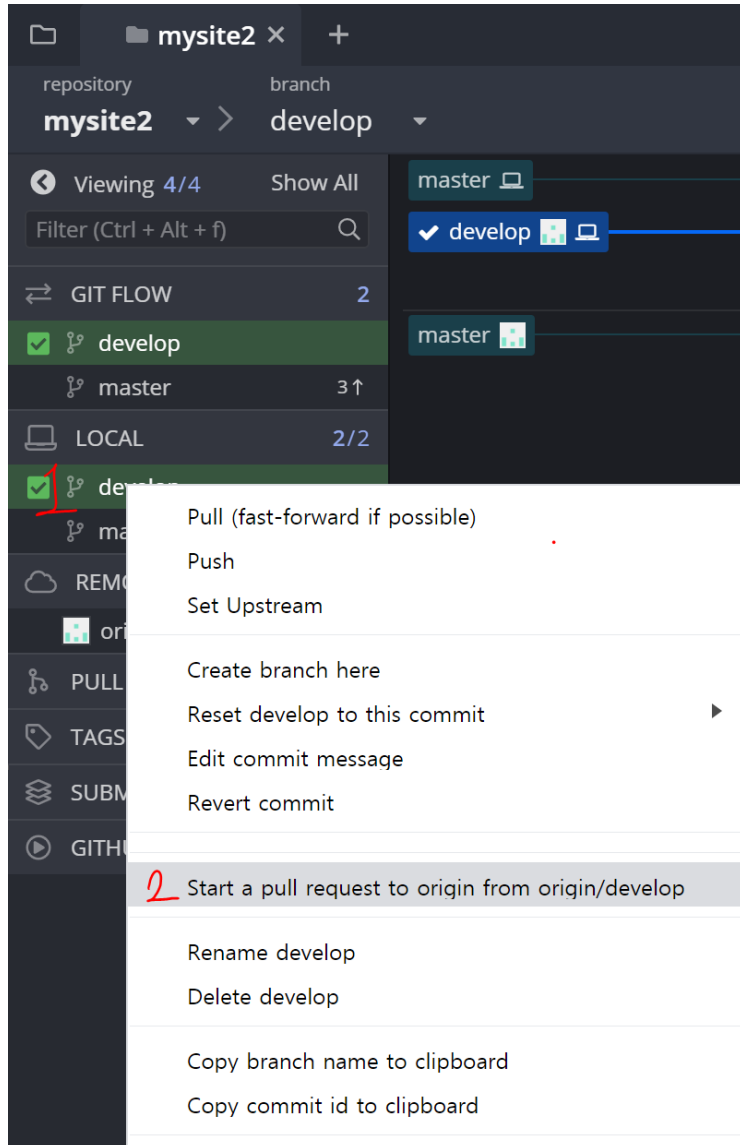


Step2는 pull request랑은 관련 없는데
Reset은 많이 쓰니까 언급한 것

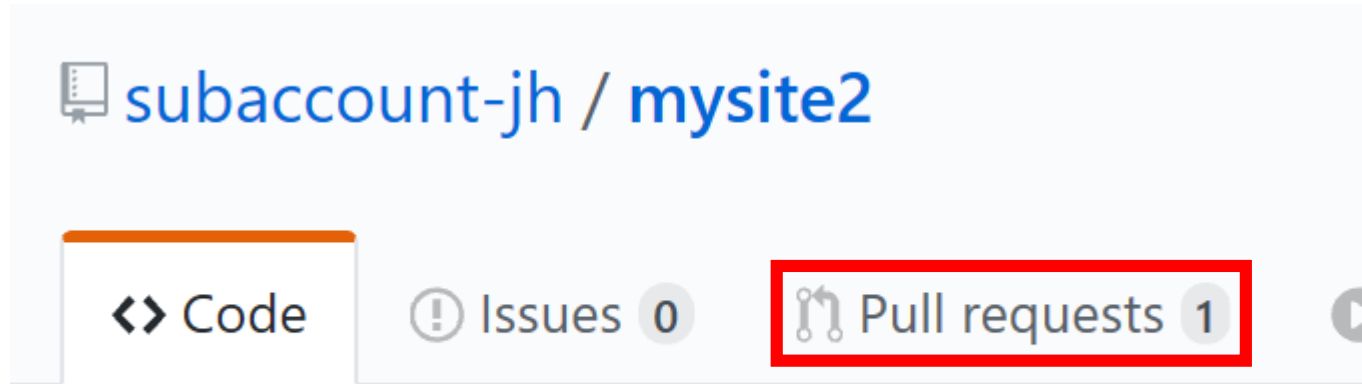


← 현재 상황

Step 3



Step 4 깃허브에 접속해서 옆자리 학회원의 작업물에 코멘트달기




Step 4 깃허브에 접속해서 옆자리 학회원의 작업물에 코멘트달기

Conversation 0

Commits 2

Checks 0


Files changed 1




leequant761 commented 4 minutes ago

Collaborator + 😊 ...

D에 대한 아주 상세한 설명




leequant761 added 2 commits 33 minutes ago



Add feature D


5d8f7ac



Merge branch 'feature/D' into develop

de92636

Add more commits by pushing to the **develop** branch on **subaccount-jh/mysite2**.



×

Review required

Add your review

At least 1 approving review is required by reviewers with write access. [Learn more](#).

×

Merging is blocked

Merging can be performed automatically with 1 approving review.

As an administrator, you may still merge this pull request.

Merge pull request

▼

You can also [open this in GitHub Desktop](#) or view [command line instructions](#).


Step 4 깃허브에 접속해서 코멘트달기


[기능 추가] D 기능 추가 #1


[Edit](#)

 Open leequant761 wants to merge 2 commits into `master` from `develop` 

 Conversation 0

 Commits 2

 Checks 0

 Files changed 1

+1 -0 

Changes from all commits ▾ File filter... ▾ Jump to... ▾ ⚙ ▾

0 / 1 files viewed ⓘ

Review changes ▾

▼ 1  D.txt 

... @@ -0,0 +1 @@

1 + D 기능 추가



Write

Preview

AA B i “ <> 🔗 ☰ ≡ ☰ ☑ @ 📌 ↶

Leave a comment

Attach files by dragging & dropping, selecting them. 

☐ Comment

Submit general feedback without explicit approval.

☒ Approve

Submit feedback and approve merging these changes.

☐ Request changes

Submit feedback that must be addressed before merging.

Submit review

끝

이재현(20기)