



Github Tutorial for ARGOS

Edit by Mani (Team Director)



Git? Github?

Git

- 분산형 버전 관리 프로그램으로서 개발협업과 프로젝트의 버전과 이력관리를 위한 도구(Version Control System)

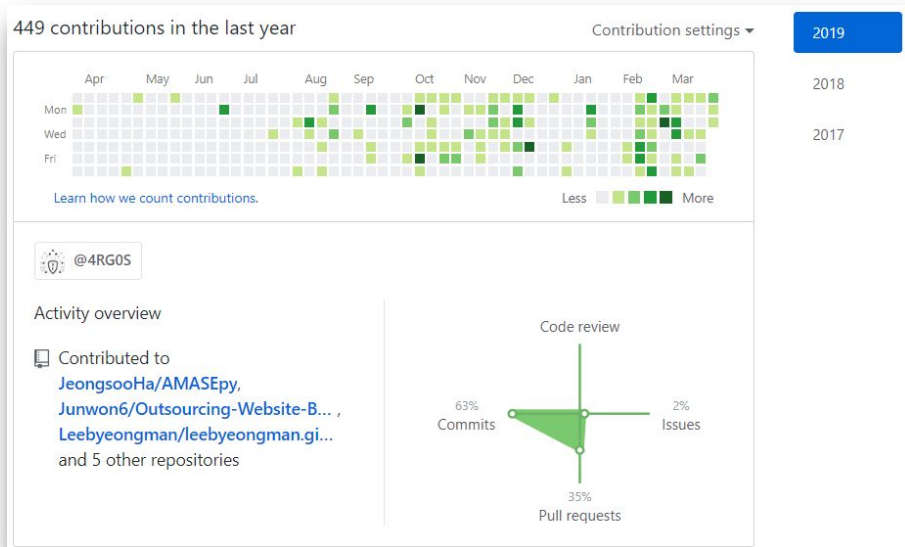
Github (Git + hub)

- **Git**의 원격 저장소 서버를 제공하는 서비스
- 로컬 **pc**에서 원격 서버에 저장되어있는 프로젝트를 다운받거나 자신이 수정한 소스를 업로드 할 수 있다.



왜 Git을 사용해야하죠?

- 효과적인 협업
- 손쉬운 개발 및 테스트 환경 구축
- 효율적인 배포관리
- 취업해야겠지?
- 18년도부터 졸업 요건에 들어감
- 잔디를 심는 즐거움 ㅋㅋ



[컴퓨터공학과 졸업요건 내규]

제4조(프로그래밍역량) ①졸업대상자는 아래 제②항 및 제③항의 내용이 포함된 졸업보고서를 지도교수에게 제출한다.

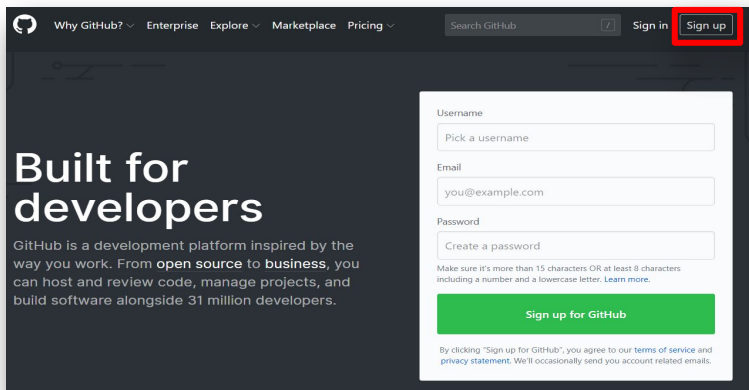
② 웹 기반 온라인 버전관리 시스템인 GitHub에 가입하고, 자신이 수행한 프로젝트를 포트폴리오로 구성하고 아이디를 졸업보고서에 기입한다. 다른 시스템을 이용한 경우에는 지도교수의 심의에 따라 인정여부를 결정한다.

③ 웹 기반 온라인 프로그래밍 테스트 시스템인 Leetcode에 가입하여 알고리즘 문제풀이 및 레벨업 후 아이디를 졸업보고서에 기입한다. 다른 시스템을 이용한 경우에는 지도교수의 심의에 따라 인정여부를 결정한다.



Github 가입해보자~(1/3)

- Github 아이디가 없는 사용자(있다면 p.7로 이동)



첫 페이지

Sign Up 페이지

- <https://github.com/>에 들어가면 나오는 첫 페이지예요. 빨간 박스를 누르면 회원가입을 할 수 있습니다.
- - 'Username'는 GitHub의 닉네임 (깃허브 아이디가 모예요? 물어보면 이 Username를 물어본거라 생각하면 돼요)
- - 'Email Address'는 GitHub의 로그인 아이디
- - 'Password'는 비밀번호를 입력
- 이렇게 정보를 입력하고 나서 **Create an account** 버튼을 클릭하세요



Github 가입해보자~(2/3)

Choose your plan

With tools developers love and the world's largest open source community, there's no wrong choice.

Free

The basics of GitHub for every developer

\$0

per month

Includes:

- ∞ Unlimited public and private repositories
- ✓ 3 collaborators for private repositories
- ✓ Issues and bug tracking
- ✓ Project management

Are you a student? Get access to the best developer tools for free with the [GitHub Student Developer Pack](#).

Pro

Pro tools for developers with advanced requirements

\$7

per month
[\(view in KRW\)](#)

Includes:

- ∞ Unlimited public and private repositories
- ∞ Unlimited collaborators
- ✓ Issues and bug tracking
- ✓ Project management
- ✓ [Advanced tools and insights](#)

계정 유형 선택

Welcome to GitHub

You'll find endless opportunities to learn, code, and create, @ByeongmanLee.

Completed
Set up a personal account

Step 2:
Choose your plan

Step 3:
Tailor your experience

How would you describe your level of programming experience?

☐ Totally new to programming ☐ Somewhat experienced ☐ Very experienced

What do you plan to use GitHub for? (check all that apply)

☐ School projects ☐ Research ☐ Design

☐ Project Management ☐ Development ☐ Other (please specify)

Which is closest to how you would describe yourself?

☐ I'm a student ☐ I'm a professional ☐ I'm a hobbyist

☐ Other (please specify)

What are you interested in?

사용자 정보 입력

1. 계정 유형은 무료로 선택해주세요
2. 공개, 비공개 repo가 제한이 없어요. (개꿀~)
3. 아래로 내리면 **Continue** 녹색 버튼이 있을 거예요, 클릭해주세요

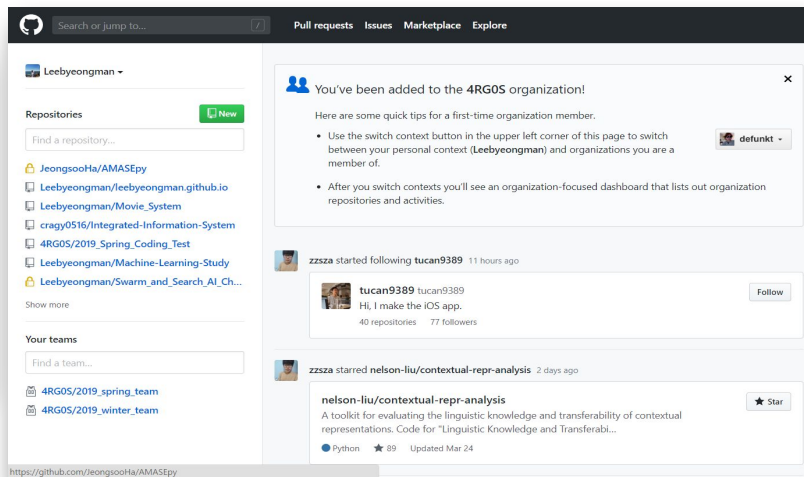
1. 사용자 정보 입력은 일단 skip을 하세요
2. 나중에 해도 상관없습니다 ...
ㅎ



Github 가입해보자~(3/3)



메일 인증



가입완료된 화면

- 그럼, 이메일로 가입한 계정이 맞는지 확인하는 메일이 발송될 거예요
- 빨간 상자를 클릭하면 인증이 된답니다.
- 인증 후 가입완료된 화면이에요.



bash 설치

Git bash 설치 블로그.. 이걸 링크로 대체할게요 ...ㅠㅠ

<https://gabii.tistory.com/entry/Git-Git-Bash-219-%EC%84%A4%EC%B9%98%ED%95%98%EA%B8%B0>



기본적인 명령어

커맨드창 다루기

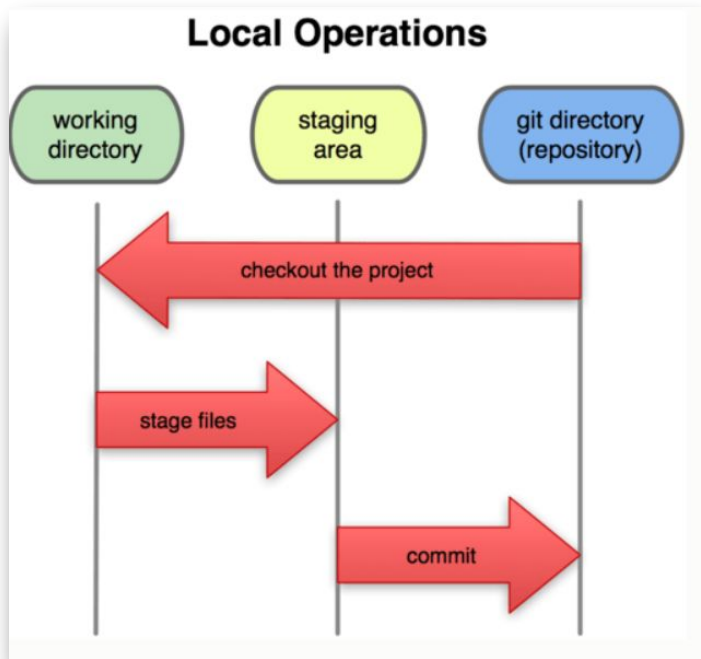
- 화면 초기화 : `Ctrl + L`
- 한 행의 처음과 끝 : `Ctrl + A`, `Ctrl + E`

명령어 키워드

- 목록 보기 : `ls` 또는 `dir`
- 파일의 내용 보기 : `cat`
- 특정 문자를 검색 : `grep`
- 디렉토리로 이동 : `cd`
- 디렉토리 생성 : `mkdir`
- 파일 삭제 : `rm`
- 파일 생성 : `touch`



Local Operations



Git 디렉토리는 Git이 프로젝트의 메타데이터와 객체 데이터베이스를 저장하는 곳을 말해요. Git 디렉토리가 Git의 핵심이에요. 다른 컴퓨터에 있는 저장소를 Clone 할 때 Git 디렉토리가 만들어져요.

워킹 디렉토리는 프로젝트의 특정 버전을 Checkout한 것이에요. Git 디렉토리는 지금 작업하는 디스크에 있고 그 디렉토리에 압축된 데이터베이스에서 파일을 가져와서 워킹 디렉토리를 만들어요.

Staging Area는 Git 디렉토리에 있어요. 단순한 파일이고 곧 커밋할 파일에 대한 정보를 저장해요. 종종 인덱스라고 불리기도 하지만, **Staging Area**라는 명칭이 표준이 되어가고 있어요.

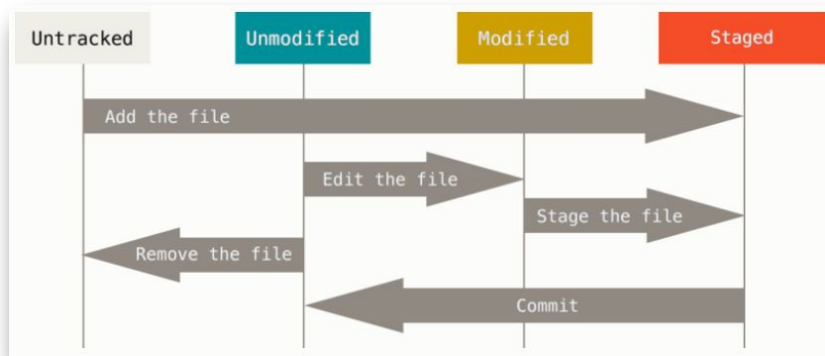
Git으로 하는 일은 기본적으로 아래와 같다:

- 워킹 디렉토리에서 파일을 수정한다.
- Staging Area에 파일을 Stage해서 커밋할 스냅샷을 만든다.
- Staging Area에 있는 파일들을 커밋해서 Git 디렉토리에 영구적인 스냅샷으로 저장한다.

※ Snapshot : git에서 각 버전을 저장하는 기능



Life Cycle



워킹 디렉토리의 모든 파일은 크게 **Tracked**(관리대상임)와 **Untracked**(관리대상이 아님)로 나눈다. **Tracked** 파일은 이미 스냅샷에 포함돼 있던 파일이다. **Tracked** 파일은 또 **Unmodified**(수정하지 않음)와 **Modified**(수정함) 그리고 **Staged**(커밋으로 저장소에 기록할) 상태 중 하나이다. 간단히 말하자면 **Git**이 알고 있는 파일이라는 것이다.

그리고 나머지 파일은 모두 **Untracked** 파일이다. **Untracked** 파일은 워킹 디렉토리에 있는 파일 중 스냅샷에도 **Staging Area**에도 포함되지 않은 파일이다. 처음 저장소를 **Clone** 하면 모든 파일은 **Tracked**이면서 **Unmodified** 상태이다. 파일을 **Checkout** 하고 나서 아무것도 수정하지 않았기 때문에 그렇다.

마지막 커밋 이후 아직 아무것도 수정하지 않은 상태에서 어떤 파일을 수정하면 **Git**은 그 파일을 **Modified** 상태로 인식한다. 실제로 커밋을 하기 위해서는 이 수정한 파일을 **Staged** 상태로 만들고, **Staged** 상태의 파일을 커밋한다. 이런 라이프사이클을 계속 반복한다.



Basics



Create New Repository(1/2)


- 빨간 박스로 되어 있는 부분을 작성해줘요
- **Public** : 모두에게 공유할 수 있고
- **Private** : 나 또는 내가 공유한 사람만 접근 할 수 있어요
- **README** 버튼을 눌러주세요
- 선택을 했다면 **Create repository** 버튼을 눌러주세요


Click!!


Leebyeongman ▾


Repositories


Find a repository...


 **New**


 JeongsooHa/AMASEpy


 Leebyeongman/leebyeongman.github.io

 Leebyeongman/Movie_System

 cragy0516/Integrated-Information-System

 4RG0S/2019_Spring_Coding_Test

 Leebyeongman/Machine-Learning-Study

 Leebyeongman/Swarm_and_Search_AI_Ch...

Show more



Create a new repository

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

Owner: Leebyeongman ▾

Repository name *: **Test** ✓

Great repository names are short and memorable. Need inspiration? How about **effective-parakeet**?

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



Create New Repository(2/2)

The screenshot shows the GitHub interface for a new repository named 'Test' by user 'Leebyeongman'. At the top, there are buttons for 'Watch' (0), 'Star' (0), and 'Fork' (0). Below this is a navigation bar with 'Code' selected, and links for 'Issues' (0), 'Pull requests' (0), 'Projects' (0), 'Wiki', 'Insights', and 'Settings'. A message states 'No description, website, or topics provided.' with an 'Edit' button. Below this, statistics show '1 commit', '1 branch', '0 releases', and '1 contributor'. A 'Branch: master' dropdown and a 'New pull request' button are present. A 'Clone or download' button is highlighted in green. A dropdown menu is open, showing options to 'Clone with HTTPS' (with a 'Use SSH' link), 'Open in Desktop', 'Open in Visual Studio', and 'Download ZIP'. The HTTPS URL is 'https://github.com/Leebyeongman/Test.git'. The repository content shows an 'Initial commit' with a 'README.md' file. The file content is 'Test'.

새로운 **repo**가 생긴 것을 확인할 수 있어요

만들기 정말 쉽죠?



git config (최초 1회 실행만 해요)

Git을 설치하고 나면 Git의 사용 환경을 적절하게 설정해줘야 해요.

설정된 내용은 Git을 업그레이드해도 유지되고 언제든지 다시 바꿀 수 있는 명령어가 있습니다.

바로 `git config`인데요.

오른쪽 사진은 `user.name`과 `email`를 설정한 결과예요

이와같이 작성하면 설정할 수 있습니다~

```
// git commit에 사용될 username
```

```
git config --global user.name "your_name"
```

```
// git commit에 사용될 email
```

```
git config --global user.email "your_email@example.com"
```

```
// 설정한 내용을 확인할 수 있다.
```

```
git config --list
```

```
ssey0@DESKTOP-TU585I3 MINGW64 ~/OneDrive/바탕 화면/argos/2019_Spring_Coding_Test/
$ git config --global user.name leebyeongman

ssey0@DESKTOP-TU585I3 MINGW64 ~/OneDrive/바탕 화면/argos/2019_Spring_Coding_Test/
$ git config --global user.email ssey0921@gmail.com

ssey0@DESKTOP-TU585I3 MINGW64 ~/OneDrive/바탕 화면/argos/2019_Spring_Coding_Test/
$ git config --list
core.symlinks=false
core.autocrlf=true
core.fscache=true
color.diff=auto
color.status=auto
color.branch=auto
color.interactive=true
help.format=html
rebase.autosquash=true
http.sslcainfo=C:/Program Files/Git/mingw64/ssl/certs/ca-bundle.crt
http.sslbackend=openssl
diff.astextplain.textconv=astextplain
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
credential.helper=manager
user.name=leebyeongman
user.email=ssey0921@gmail.com
core.repositoryformatversion=0
core.filemode=false
core.bare=false
core.logallrefupdates=true
core.symlinks=false
core.ignorecase=true
remote.origin.url=https://github.com/Leebyeongman/2019_Spring_Coding_Test.git
remote.origin.fetch=+refs/heads/*:refs/remotes/origin/*
branch.master.remote=origin
branch.master.merge=refs/heads/master
```



git init (현재 디렉토리를 로컬로 설정해줘요)

로컬저장소로 설정할 위치로 이동해서 `git init`을 입력하면

(master) 라고 뜨는 것을 확인 할 수 있어요

그럼 잘하고 있는 거예요

// 로컬저장소로 설정할 프로젝트 위치로 이동한다.

`cd test`

// 로컬저장소로 설정한다.

// (master) 브랜치로 보이면 성공한 것이다.

`git init`

// 만약 init을 취소하려면 아래의 명령어를 입력한다.

`rm -r .git`

```
MINGW64;C:/Users/ssey0/OneDrive/바탕 화면/argos/test

ssey0@DESKTOP-TU5B5I3 MINGW64 ~/OneDrive/바탕 화면 /argos
$ cd test/

ssey0@DESKTOP-TU5B5I3 MINGW64 ~/OneDrive/바탕 화면 /argos/test
$ git init
Initialized empty Git repository in C:/Users/ssey0/OneDrive/바탕 화면 /argos/test/.git/

ssey0@DESKTOP-TU5B5I3 MINGW64 ~/OneDrive/바탕 화면 /argos/test (master)
$ |
```



git remote (로컬저장소와 원격저장소를 연결해줘요)

이제 로컬저장소와 원격저장소를 연결해줘야해요

명령어는 다음과 같아요. **origin** 부분은 원하는 대로 작성하면

되는데 보통은 **origin**으로 사용해요

연결을 하고 연결 되었는지 확인해주세요

아주 잘하고 있습니다.

```
ssey0@DESKTOP-TU5B5I3 MINGW64 ~/OneDrive/바탕 화면 /argos/test (master)
$ git remote add origin https://github.com/Leebyeongman/Test.git

ssey0@DESKTOP-TU5B5I3 MINGW64 ~/OneDrive/바탕 화면 /argos/test (master)
$ git remote -v
origin https://github.com/Leebyeongman/Test.git (fetch)
origin https://github.com/Leebyeongman/Test.git (push)
```

// Github 원격저장소와 연결한다.

`git remote add origin` [자신의 Github 원격저장소 주소]

// 연결된 원격저장소 확인한다.

`git remote -v`



git status (로컬저장소의 현재 상태를 보여줘요)

본격적으로 파일을 올리기 위해서 해주는 작업이에요

a.py라는 파일이 빨간색으로 있는 것을 확인할 수 있어요

새로 만들어졌다는 것을 의미해요

이제 저 파일을 추가 해볼게요

// 파일의 상태를 확인한다.

git status

```
ssey0@DESKTOP-TU5B5I3 MINGW64 ~/OneDrive/바탕 화면 /argos/test (master)
$ vi a.py

ssey0@DESKTOP-TU5B5I3 MINGW64 ~/OneDrive/바탕 화면 /argos/test (master)
$ git status
On branch master

No commits yet

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

        a.py

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



git add (파일을 준비영역으로 옮겨줘요)

파일을 추가하면 이전에 빨간색이었던 부분이 초록색으로 바뀌었다는 것을 확인할 수 있어요.

이제 파일이 준비영역으로 옮겨졌다는 것이예요

// a.py 파일만 추가

`git add a.py`

// 워킹 디렉터리 내 모든 파일을 추가

`git add .`

// 명령 프롬프트에서 상호작용하면서 추가 (나갈땐 q를 입력)

`git add -i`

// 진행중인 파일일 경우, Staging Area에서 워킹 디렉터리로 옮겨온다.

`$git rm --cached a.html`

`$git rm -r --cached .`

```
ssey0@DESKTOP-TU5B5I3 MINGW64 ~/OneDrive/바탕 화면 /argos/test (master)
$ git add a.py

ssey0@DESKTOP-TU5B5I3 MINGW64 ~/OneDrive/바탕 화면 /argos/test (master)
$ git status
On branch master

No commits yet

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

        new file:   a.py
```



git commit (준비영역의 파일을 로컬저장소에 저장해주줘요)

commit을 사용해서 저장소에 저장을 해줘야하는데 보통은
두번째와 같은 방식을 많이 사용해요

// 에디터가 출력되고, 에디터에서 커밋 메시지 입력 후 저장하면 커밋됨
`git commit`

// 간단한 커밋 메시지를 입력후 커밋
`git commit -m "커밋 메시지"`

// Staging Area에 들어간 파일에 대해서만 (워킹 디렉터리는 적용 X)
`git commit -a -m "커밋 메시지"`

```
ssey0@DESKTOP-TU5B5I3 MINGW64 ~/OneDrive/바탕 화면 /argos/test (master)
$ git commit -m "first commit"
[master c1f1f5f] first commit
1 file changed, 1 insertion(+)
create mode 100644 a.py
```



git log (로컬저장소의 커밋 이력을 조회해줘요)

commit이 잘 올라갔는지 확인을 할 수 있어요

명령어는 `git log`를 입력하면 돼요

아까 커밋 메시지로 남긴 “first commit”을 확인할 수 있어요

// 커밋 이력 상세조회

`git log`

// 커밋 이력중 커밋ID, 타이틀 메시지만 조회

`git log --oneline`

// 모든 브랜치 커밋 이력 조회

`git log --oneline --decorate --graph --all`

// 특정 파일의 변경 커밋 조회

`git log -- a.html`

```
ssey0@DESKTOP-TU5B5I3 MINGW64 ~/OneDrive/바탕 화면 /argos/test (master)
$ git log
commit c1f1f5f94cdc925eb9cafd2b0057fcdc9eaffc57 (HEAD -> master)
Author: leebyeongman <ssey0921@gmail.com>
Date: Mon Mar 25 14:16:24 2019 +0900

    first commit

commit 22510642a01638bd14ba3fb668672d556f2c0512 (origin/master)
Author: ManiMani <33439998+Leebyeongman@users.noreply.github.com>
Date: Mon Mar 25 14:14:03 2019 +0900

    Initial commit
```



git push (원격저장소에 저장해줘요)

github 페이지에 들어가서 새로고침을 하면 추가한 a.py가
원격저장소에 올라간 것을 확인할 수 있어요

// 원격저장소에 저장한다.

git push -u origin master

// 에러 - ![rejected] master -> master (fetch first)

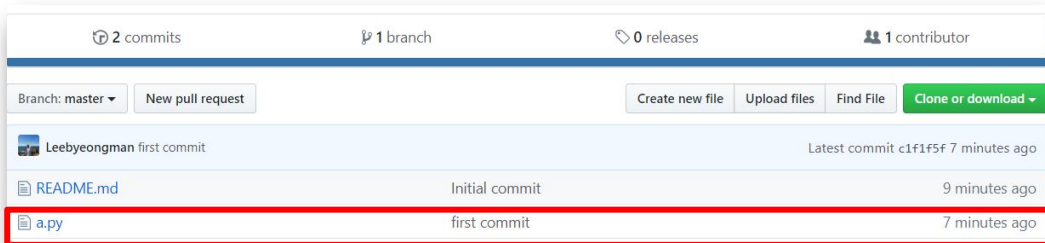
// 이미 변경된 파일이 원격저장소에 있을경우 발생

git pull origin master

// 에러 - ![rejected] master -> master (non-fast-forward)

git push origin +master

```
ssey0@DESKTOP-TU5B5I3 MINGW64 ~/OneDrive/바탕 화면 /argos/test (master)
$ git push -u origin master
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 273 bytes | 136.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/Leebyeongman/Test.git
 2251064..c1f1f5f master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
```





git pull (원격저장소에서 업데이트 된 파일을 가져와요)

// 원격저장소에서 업데이트 된 파일을 가져온다.

git pull origin master

```
ssey0@DESKTOP-TU5B5I3 MINGW64 ~/OneDrive/바탕 화면 /argos/test (master)
$ git pull origin master
From https://github.com/Leebyeongman/Test
* branch          master      -> FETCH_HEAD
```



Advanced



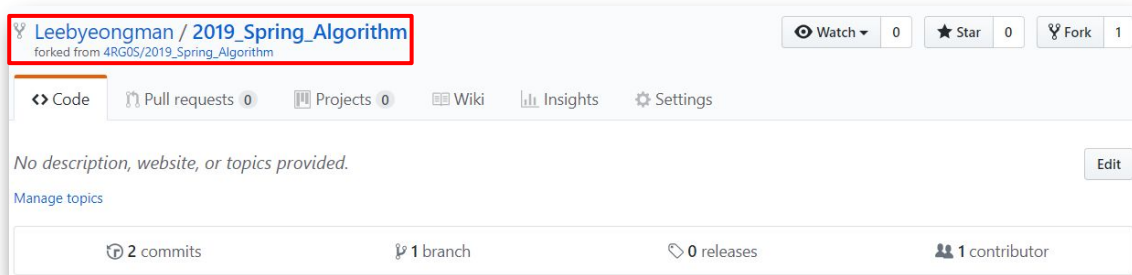
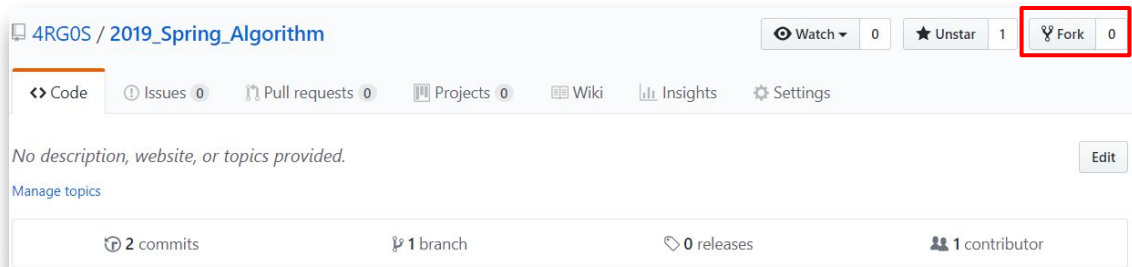
Pull request 절차

1. Fork
2. clone
3. remote 설정
4. branch 생성
5. 파일 수정 후 add, commit, push
6. Pull request 생성
7. Merge
8. Merge 이후 동기화
9. branch 삭제

※ 개인 **repo**를 관리할 줄 안다면 이제는 다른 사람의 **repo**에 기여하는 것도 알아야해요.
나중에 협업을 하다보면 **pull request**를 많이 사용하거든요



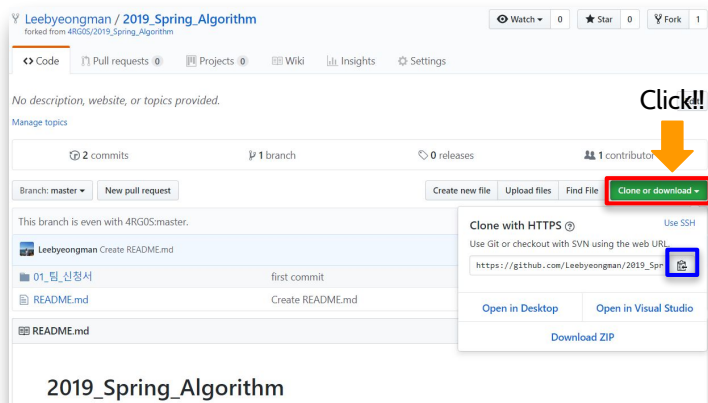
Fork (해당 프로젝트의 저장소를 자신의 저장소로 가져와요)



1. 위에 보이는 빨간 박스를 클릭해서 Fork를 떠줍니다.
2. 그러면 책을 포크로 찍은 그림이 나올 거예요
3. 포크를 뗐으면 아래와 같이 자기 원격 저장소에 **repo**가 생긴 것을 확인 할 수 있어요



clone



초록색 버튼을 누르고 파란 박스가 있는 버튼을 누르면 해당 깃 주소를 복사할 수 있어요~
이제 생성한 **repo**를 내 컴퓨터에 가져와볼게요~

먼저, **bash**를 켜주세요.
자기가 원하는 곳(바탕화면 권장 - 확인하기 쉬움)에 폴더(ex. **test**)를 하나 만들어주세요

원격저장소에서 로컬로 가져오기 위해서는 다음 명령어를 입력해주면 돼요

cd로 **clone**한 폴더로 들어가세요

ls를 입력하면 팀 신청서 폴더와 **README**가 있는 것을 확인할 수 있어요

```
ssey0@DESKTOP-TU5B5I3 MINGW64 ~/OneDrive/바탕 화면/argos/gg
$ git clone https://github.com/Leebyeongman/2019_Spring_Algorithm.git
Cloning into '2019_Spring_Algorithm'...
remote: Enumerating objects: 7, done.
remote: Counting objects: 100% (7/7), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 7 (delta 0), reused 4 (delta 0), pack-reused 0
Unpacking objects: 100% (7/7), done.

ssey0@DESKTOP-TU5B5I3 MINGW64 ~/OneDrive/바탕 화면/argos/gg
$ cd 2019_Spring_Algorithm/

ssey0@DESKTOP-TU5B5I3 MINGW64 ~/OneDrive/바탕 화면/argos/gg/2019_Spring_Algorithm
$ ls
01_팀_신청서 / README.md
```

// 원격저장소에서 로컬저장소로 가져온다.

git clone [자신의 Github 원격저장소 주소]



remote 설정

```
ssey0@DESKTOP-TU5B5I3 MINGW64 ~/OneDrive/바탕 화면/argos/gg/2019_Spring_Algorithm (master)
$ git remote -v
origin https://github.com/Leebyeongman/2019_Spring_Algorithm.git (fetch)
origin https://github.com/Leebyeongman/2019_Spring_Algorithm.git (push)
```

```
ssey0@DESKTOP-TU5B5I3 MINGW64 ~/OneDrive/바탕 화면/argos/gg/2019_Spring_Algorithm (master)
$ git remote add upstream https://github.com/4RG0S/2019_Spring_Algorithm.git

ssey0@DESKTOP-TU5B5I3 MINGW64 ~/OneDrive/바탕 화면/argos/gg/2019_Spring_Algorithm (master)
$ git remote -v
origin https://github.com/Leebyeongman/2019_Spring_Algorithm.git (fetch)
origin https://github.com/Leebyeongman/2019_Spring_Algorithm.git (push)
upstream https://github.com/4RG0S/2019_Spring_Algorithm.git (fetch)
upstream https://github.com/4RG0S/2019_Spring_Algorithm.git (push)
```

1. Fork 한 repo 를 최신으로 동기화시켜야 할 때가 있어요
2. 이를 위해서는 먼저 원본 repo 를 remote repo 로 추가해줘야 해요
3. Fork 해온 repo 에서 remote repo 를 확인하면 아래와 같이 나올 거예요
4. 여기에 동기화해오고 싶은 원본 repo 를 upstream 이라는 이름으로 추가해줘요
5. upstream repo 가 제대로 추가 되었는지 확인해주세요



branch 생성

```
ssey0@DESKTOP-TU5B5I3 MINGW64 ~/OneDrive/바탕 화면 /argos/gg/2019_Spring_Algorithm (master)
$ git branch
* master

ssey0@DESKTOP-TU5B5I3 MINGW64 ~/OneDrive/바탕 화면 /argos/gg/2019_Spring_Algorithm (master)
$ git checkout -b mani
Switched to a new branch 'mani'

ssey0@DESKTOP-TU5B5I3 MINGW64 ~/OneDrive/바탕 화면 /argos/gg/2019_Spring_Algorithm (mani)
$ git branch
* mani
  master
```

자신의 로컬 컴퓨터에서 코드를 추가하는 작업은 **branch**를 만들어서
진행해줘야 해요.

여러 사람이 작업하려면 나만의 ID 같은 것이 필요하겠죠? 그게 바로

branch예요

먼저 현재 브랜치가 어떤 것이 있는지 확인해주세요 명령어는 **git branch**

이예요

master밖에 없죠? 이제 자신만의 **branch**를 생성해줄 거예요

오른쪽에 저렇게 많은 명령어들이 있는데 저는 보통 저 두 개를 많이 쓰고
있어요

노란 박스를 보면 **master**에서 **mani**로 브랜치가 바뀐 것을 알 수 있죠?

// branch 보기
git branch

// branch 생성
git branch [branch name]

// branch 수정
git branch -m [branch name][바꿀 이름]

// branch 삭제
git branch -d [branch name]

// 특정 브랜치로 워킹 디렉터리 변경
git checkout [branch name]

// 특정 커밋으로 워킹 디렉터리 변경
git checkout [Commit ID]

// 브랜치 생성 및 체크아웃을 같이 할 경우
git checkout -b mani



파일 수정 후 add, commit, push

```
ssey0@DESKTOP-TU5B5I3 MINGW64 ~/OneDrive/바탕 화면 /argos/gg/2019_Spr
$ vi test.txt

ssey0@DESKTOP-TU5B5I3 MINGW64 ~/OneDrive/바탕 화면 /argos/gg/2019_Spring_Algorithm (mani)
$ git add .
warning: LF will be replaced by CRLF in test.txt.
The file will have its original line endings in your working directory

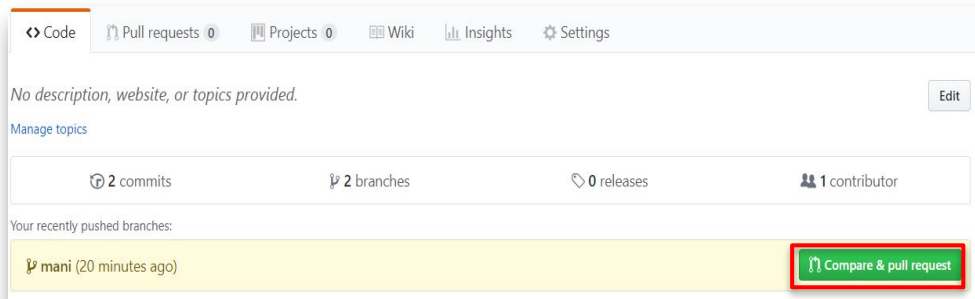
ssey0@DESKTOP-TU5B5I3 MINGW64 ~/OneDrive/바탕 화면 /argos/gg/2019_Spring_Algorithm (mani)
$ git commit -m "test"
[mani bb6038c] test
1 file changed, 1 insertion(+)
create mode 100644 test.txt

ssey0@DESKTOP-TU5B5I3 MINGW64 ~/OneDrive/바탕 화면 /argos/gg/2019_Spring_Algorithm (mani)
$ git push origin mani
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 316 bytes | 158.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
remote:
remote: Create a pull request for 'mani' on GitHub by visiting:
remote:   https://github.com/Leebyeongman/2019_Spring_Algorithm/pull/new/mani
remote:
To https://github.com/Leebyeongman/2019_Spring_Algorithm.git
 * [new branch]      mani -> mani
```

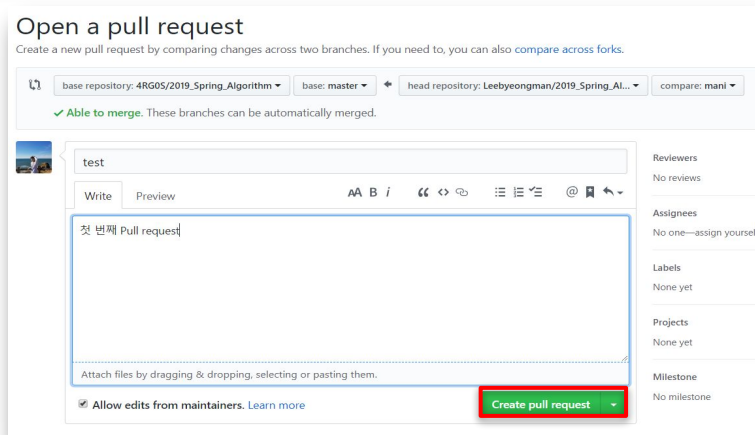
1. 자신이 추가하고 싶은 코드를 수정해줘요
2. 수정이 완료되면, **basic** 단계에서 배웠던 **add, commit, push**를 통해서 자신의 **git repo (origin)**에 수정사항을 반영해줘요
3. **주의사항** push 진행 시에 **branch** 이름을 명시해주어야 해요



Pull request 생성

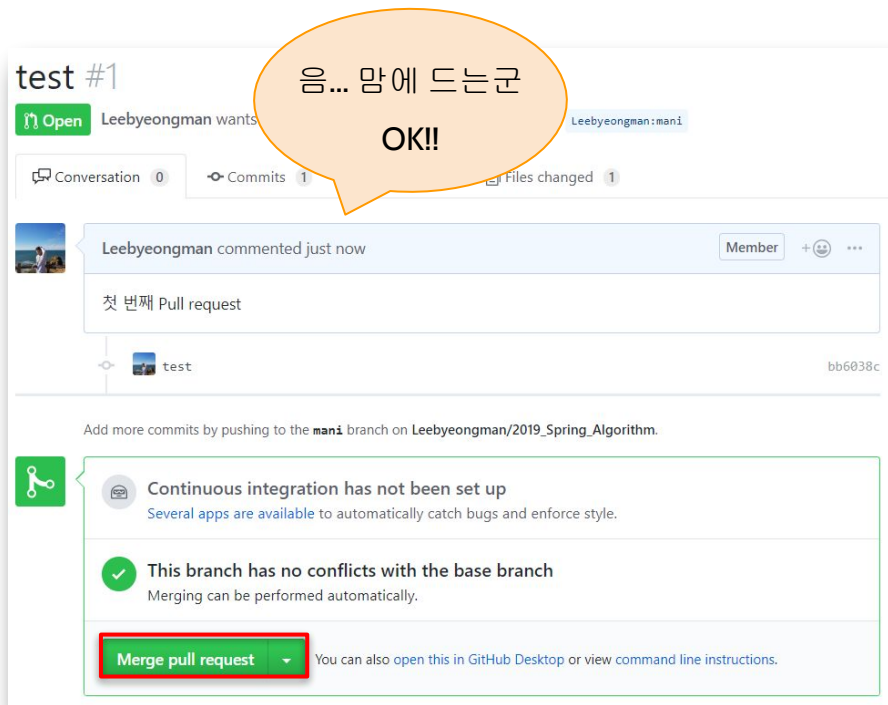


1. push 완료 후 본인 계정의 **github** 저장소에 들어오면 **Compare & pull request** 버튼이 저런식으로 **씩** 활성화 되어 있을거예요.
2. 해당 버튼을 선택하세요. 그러면 아래쪽 사진과 같은 창이 나와요.
3. 메시지를 작성하고 아래 버튼을 눌러주세요 (PR을 생성해줘요)





Merge



1. PR을 받은 원본 저장소 관리자는 코드 변경내역을 확인하고 **Merge** 여부를 결정해요
2. 변경하기 싫으면 안 받아준답니다. (3학년 2학기 공개소프트웨어 실습 수업에서 많이 할 거예요)
3. 여기서는 제가 만든 저장소이기 때문에 가능해요
4. 또는 **Collaborator** 이여도 가능해요



Merge 이후 동기화

```
ssey0@DESKTOP-TU5B5I3 MINGW64 ~/OneDrive/바탕 화면 /argos/2019_Spring_Coding_Test/이 병 만 (byeongman)
$ git remote -v
origin https://github.com/Leebyeongman/2019_Spring_Coding_Test.git (fetch)
origin https://github.com/Leebyeongman/2019_Spring_Coding_Test.git (push)
upstream https://github.com/4RG0S/2019_Spring_Coding_Test.git (fetch)
upstream https://github.com/4RG0S/2019_Spring_Coding_Test.git (push)
```

```
ssey0@DESKTOP-TU5B5I3 MINGW64 ~/OneDrive/바탕 화면 /argos/2019_Spring_Coding_Test/이 병 만 (byeongman)
$ git fetch upstream
remote: Enumerating objects: 14, done.
remote: Counting objects: 100% (13/13), done.
remote: Compressing objects: 100% (9/9), done.
remote: Total 10 (delta 2), reused 7 (delta 1), pack-reused 0
Unpacking objects: 100% (10/10), done.
From https://github.com/4RG0S/2019_Spring_Coding_Test
d459c1f..deffa47 master -> upstream/master
```

```
ssey0@DESKTOP-TU5B5I3 MINGW64 ~/OneDrive/바탕 화면 /argos/2019_Spring_Coding_Test/이 병 만 (byeongman)
$ git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.
```

```
ssey0@DESKTOP-TU5B5I3 MINGW64 ~/OneDrive/바탕 화면 /argos/2019_Spring_Coding_Test/이 병 만 (master)
$ git merge upstream/master
Updating deffa47..9ed30ae
Fast-forward
.../1238. Contact/Solution_v1.py | 5 ---
.../1238. Contact/Solution_v2.py | 42 ++++++
2 files changed, 42 insertions(+), 5 deletions(-)
rename "354\235\264\353\263\221\353\247\214\1238. Contact/Solution.py" -> "354\235\264\353\263\221\353\247\214\1238. Contact/Solution_v1.py" (84%)
create mode 100644 "354\235\264\353\263\221\353\247\214\1238. Contact/Solution_v2.py"
```

```
ssey0@DESKTOP-TU5B5I3 MINGW64 ~/OneDrive/바탕 화면 /argos/2019_Spring_Coding_Test/이 병 만 (master)
$ git push origin master
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 706 bytes | 353.00 KiB/s, done.
Total 2 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/Leebyeongman/2019_Spring_Coding_Test.git
deffa47..9ed30ae master -> master
```

git **fetch** upstream(remote 별명)

git checkout master

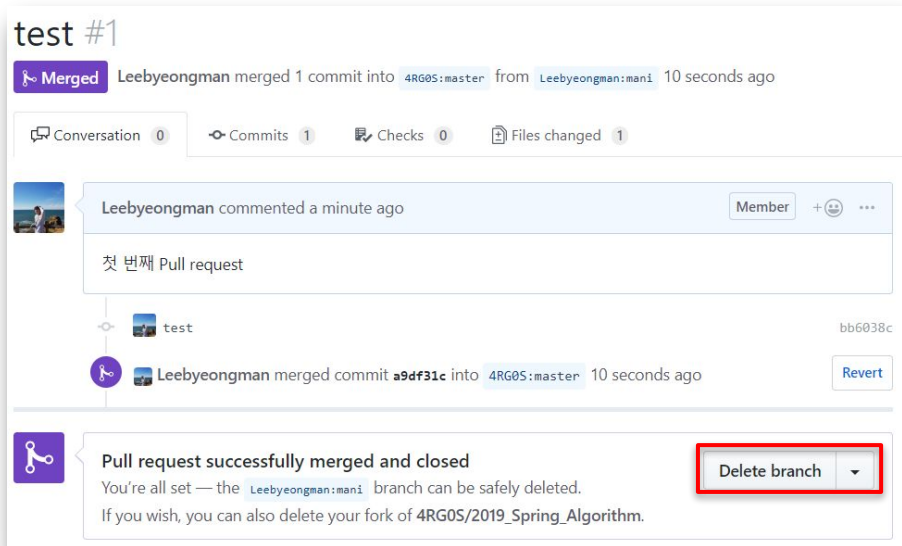
git merge upstream/master

git push origin master

1. Merge 이후 로컬 코드와 원본 저장소의 코드를 동기화 해줘요
2. 이제 이전에 등록한 upstream repo로부터 최신 업데이트를 가져와요
3. Git의 fetch 명령어를 통해 upstream repo의 내용을 불러와요
4. upstream repository 의 master branch (혹은 원하는 branch) 로부터 나의 local master branch 로 merge 해줘요
5. 이 과정까지는 local repository 에서 일어난 것이므로 push 를 통해 remote repository 에도 적용시켜주면 Finish!



branch 삭제



// 브랜치 삭제

git **branch -d** mani(브랜치 별명)

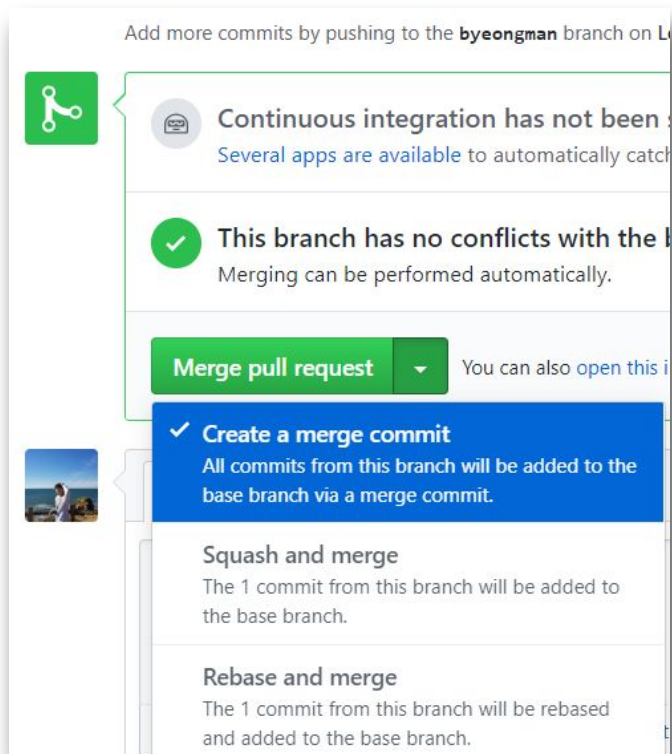
1. 동기화를 완료했다면 **branch** 작업했던 로컬의 **branch**를 삭제해주세요(빨간 박스 버튼으로도 삭제 가능해요)
2. 나중에 추가로 작업할 일이 있으면 이전과 같은 명령어를 통해 원본 저장소와 동기화를 진행하고 **branch** 생성부터 지금까지 반복해주세요



Additional



Merge 종류



이전 자료에 대해서 질문이 들어와서 잠시 **merge**의 종류에 대해서 간단하게 알려드릴게요.

GitHub의 새 버전에서 **merge**, **squash and merge**, **rebase and merge** 세 종류의 **merge**를 모두 지원하기 시작했네요.

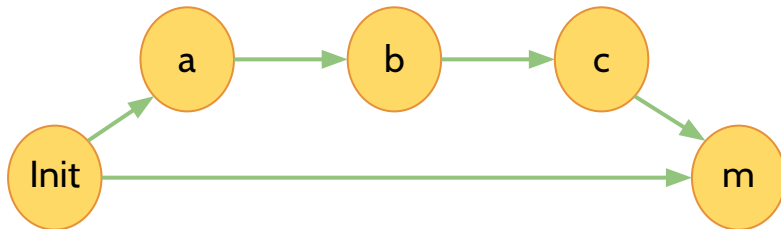
각 머지 방식에 따라 커밋 히스토리가 천차만별로 달라지는데요, 어떤 경우에 어떤 머지를 사용하는 것이 좋은지 공유하고자 각 **merge**에 대해 설명해 드리려고요~



Merge

1. 그래프

- Merge (a, b, c 를 refer 하는 m 커밋 노드 생성, m은 parent로 Init, c 를 가짐



2. 메인 브랜치의 관점

커밋 **m**에서부터 뒤로 되돌아가면서 부모를 모두 찾아 브랜치를 구성.

커밋 **m**은 부모로 **c**, **Init**을 가지고 있으며, **c**는 **b**, **b**는 **a**, **a**는 **Init**을 다시 부모로 가짐.

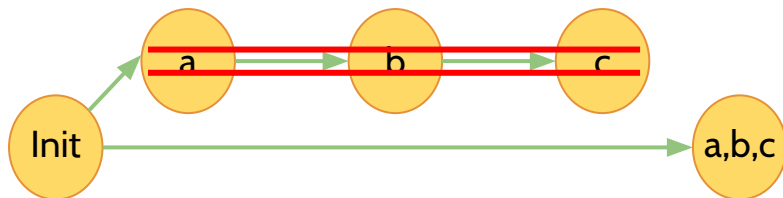
이 형상을 모두 **backtrace** 하여, **Init -> a -> b -> c -> m**이라는 구조를 만들고 이 구조가 모두 히스토리에 남음.



Squash and Merge

1. 그래프

- Squash and Merge (a, b, c 를 합쳐서 새로운 커밋으로 만들고, 머지 대상 브랜치에 추가, 'a,b,c' 커밋은 parent를 Init 하나만 가짐.)



2. 메인 브랜치의 관점

커밋 'a,b,c' 는 Init만을 부모로 가진 단일 커밋.

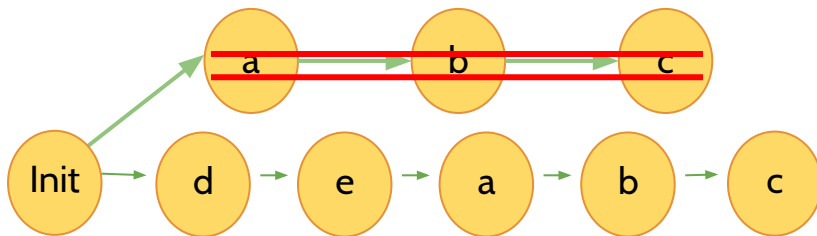
작업했던 브랜치의 a, b, c 커밋들은 머지 후의 메인 브랜치 커밋 Init, 'a,b,c' 와 아무런 연관을 가지지 않음.



Rebase and Merge

1. 그래프

- Rebase and Merge (a, b, c 를 심리스하게 머지 대상 브랜치로 추가, 각 커밋들은 모두 **parent**를 하나씩만 가짐.)



2. 메인 브랜치의 관점

커밋 **a, b, c** 의 관계를 그대로 유지한 채, 메인 브랜치에 그대로 추가.

커밋 **a**는 부모로 커밋 **e**를 가짐.

Rebase and Merge 작업 후에는, 작업했던 브랜치의 **a, b, c** 커밋들은 머지 후의 메인 브랜치의 **Init, d, e, a, b, c** 커밋들과 연관 관계를 가지지 않음.



사용 예시

Git Flow 를 따른다고 했을 때, 아래와 같이 정리할 수 있습니다.

- **develop - feature** 브랜치간 머지 : **Squash and Merge**가 유용합니다. **feature**의 복잡하고 지저분한 커밋 히스토리를 모두 묶어 완전 새로운 커밋으로 **develop** 브랜치에 추가하여, **develop** 브랜치에서 독자적으로 관리할 수 있기 때문입니다. 일반적으로 머지 후에 **feature** 브랜치를 삭제해버리는 점을 떠올려 보면, **feature** 브랜치의 커밋 히스토리를 모두 **develop** 브랜치에 직접 연관 지어 남길 필요가 없습니다.
- **master - develop** 브랜치간 머지 : **Rebase and Merge**가 유용합니다. **develop**의 내용을 **master**에 추가할 때에는 별도의 새로운 커밋을 생성할 이유가 없기 때문입니다.
- **hotfix - develop, hotfix - master** 브랜치간 머지 : **Merge** 또는 **Squash and Merge** 모두 유용합니다. 때에 따라 골라 사용하면 좋을 것 같습니다. **hotfix** 브랜치 작업의 각 커밋 히스토리가 모두 남아야 하는 경우 **Merge**, 필요 없는 경우 **Squash and Merge**를 사용하면 됩니다.

지금까지 설명한 부분은 언젠가는 필요한 부분이지만, 아직은 그렇게 중요하지 않다고 생각해요.

저도 이번에 처음 알게 되었거든요.... ㅎㅎ

처음 **Git**을 사용하시는 분들은 이런 **merge** 종류가 있다~ 이정도만 알아두시는 게 좋을 거 같아요.



git add 취소 (파일 상태를 Unstage로 변경해요)

```
ssey0@DESKTOP-TU5B5I3 MINGW64 ~/OneDrive/바탕 화면/blog/test (master)
$ vi README.md

ssey0@DESKTOP-TU5B5I3 MINGW64 ~/OneDrive/바탕 화면/blog/test (master)
$ git status
On branch master
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:   README.md

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

// add 취소

git reset HEAD 파일 이름

```
ssey0@DESKTOP-TU5B5I3 MINGW64 ~/OneDrive/바탕 화면/blog/test (master)
$ git add .
warning: LF will be replaced by CRLF in README.md.
The file will have its original line endings in your working directory

ssey0@DESKTOP-TU5B5I3 MINGW64 ~/OneDrive/바탕 화면/blog/test (master)
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    modified:   README.md
```

```
ssey0@DESKTOP-TU5B5I3 MINGW64 ~/OneDrive/바탕 화면/blog/test (master)
$ git reset HEAD README.md
Unstaged changes after reset:
M   README.md

ssey0@DESKTOP-TU5B5I3 MINGW64 ~/OneDrive/바탕 화면/blog/test (master)
$ git status
On branch master
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:   README.md

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

1. Staging Area(git add 명령 수행한 후의 상태)에 넣은 파일을 빼고 싶을 때가 있다
2. 지금은 없다고 생각하지만 있을거야 한 번은 있을거야..
3. 그럴 때 위의 명령어를 쓰면 취소할 수 있어요~



git commit 취소

```
ssey0@DESKTOP-TU5B5I3 MINGW64 ~/OneDrive/바탕 화면/blog/test (master)
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   README.md

ssey0@DESKTOP-TU5B5I3 MINGW64 ~/OneDrive/바탕 화면/blog/test (master)
$ git commit -m "test commit"
[master aaa32e9] test commit
1 file changed, 1 insertion(+)

ssey0@DESKTOP-TU5B5I3 MINGW64 ~/OneDrive/바탕 화면/blog/test (master)
$ git log
commit aaa32e916a4c24d88dbf26b5cc8dfe2d68309cd4 (HEAD -> master)
Author: leebyeongman <ssey0921@gmail.com>
Date:   Sat Apr 13 01:07:58 2019 +0900

    test commit

commit 331fe0168b9254e91d4f43c006ad17d2a99ef7e1 (origin/master)
Author: leebyeongman <ssey0921@gmail.com>
Date:   Thu Apr 11 21:23:44 2019 +0900

    first commit

ssey0@DESKTOP-TU5B5I3 MINGW64 ~/OneDrive/바탕 화면/blog/test (master)
$ git reset --soft HEAD^

ssey0@DESKTOP-TU5B5I3 MINGW64 ~/OneDrive/바탕 화면/blog/test (master)
$ git log
commit 331fe0168b9254e91d4f43c006ad17d2a99ef7e1 (HEAD -> master, origin/master)
Author: leebyeongman <ssey0921@gmail.com>
Date:   Thu Apr 11 21:23:44 2019 +0900

    first commit
```

// [방법 1] commit을 취소하고 해당 파일들은 staged 상태로 워킹 directory에 보존

git reset --soft HEAD^

// [방법 2] commit을 취소하고 해당 파일들은 unstaged 상태로 워킹 directory에 보존

git reset --mixed HEAD^ // 기본 옵션

git reset HEAD^ // 위와 동일

git reset HEAD~2 // 마지막 2개의 commit을 취소

// [방법 3] commit을 취소하고 해당 파일들은 unstaged 상태로 워킹 directory에서 삭제

git reset --hard HEAD^

완료한 commit을 취소해야 할 때가 있어요.

1. 너무 일찍 commit한 경우
2. 어떤 파일을 빼먹고 commit한 경우 이때, git reset HEAD^ 명령어를 통해 git commit을 취소할 수 있어요.

왼쪽 사진처럼 commit을 했는데 위와 같은 경우가 생겨서 취소를 하고 싶을 때 방법 1처럼 하면 git log를 확인해봤듯이 commit이 취소가 되었다는 것을 알 수 있죠? 다른 방법들은 설명을 써놓았으니 읽어보시고 나중에 테스트 repo 만들어서 해보세요.



Tip

- reset 옵션
 - `-soft` : index 보존(add한 상태, staged 상태), 워킹 directory의 파일 보존. 즉 모두 보존.
 - `-mixed` : index 취소(add하기 전 상태, unstaged 상태), 워킹 directory의 파일 보존 (기본 옵션)
 - `-hard` : index 취소(add하기 전 상태, unstaged 상태), 워킹 directory의 파일 삭제. 즉 모두 취소.
- 만약 워킹 directory를 원격 저장소의 마지막 commit 상태로 되돌리고 싶으면, 아래의 명령어를 사용하세요. 단, 이 명령을 사용하면 원격 저장소에 있는 마지막 commit 이후의 워킹 디렉터리와 add했던 파일들이 모두 사라지므로 주의해야 해요.

// 워킹 directory를 원격 저장소의 마지막 commit 상태로 되돌린다.

git **reset** --hard HEAD



git commit message 변경

```
ssey0@DESKTOP-TU5B5I3 MINGW64 ~/OneDrive/바탕 화면/blog/test (master)
$ git log
commit d94a0e1697b57d122c979b071969557a9abf0501 (HEAD -> master)
Author: leebyeongman <ssey0921@gmail.com>
Date: Thu Apr 11 21:23:44 2019 +0900

first commit
```

// git commit message를 변경할 수 있다.

git commit --amend

```
first commit
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# Date: Thu Apr 11 21:23:44 2019 +0900
#
# On branch master
#
# Initial commit
```

git push -f

```
commit message change
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# Date: Thu Apr 11 21:23:44 2019 +0900
#
# On branch master
#
# Initial commit
```

Git을 사용하다 보면 **commit message**를 오타가 나거나 조금 더 나은 내용으로 바꿔야 할 때가 있어요. 일반적으로 그대로 두는 것이 제일 좋지만 불가피하게 바꿔야 한다면 어떻게 해야 할까요?.

```
ssey0@DESKTOP-TU5B5I3 MINGW64 ~/OneDrive/바탕 화면/blog/test (master)
$ git commit --amend
[master 4f9a8fe] commit message change
Date: Thu Apr 11 21:23:44 2019 +0900
1 file changed, 1 insertion(+)
create mode 100644 README.md

ssey0@DESKTOP-TU5B5I3 MINGW64 ~/OneDrive/바탕 화면/blog/test (master)
$ git log
commit 4f9a8fef23f152458a3e41ff3cc719dad0777000 (HEAD -> master)
Author: leebyeongman <ssey0921@gmail.com>
Date: Thu Apr 11 21:23:44 2019 +0900

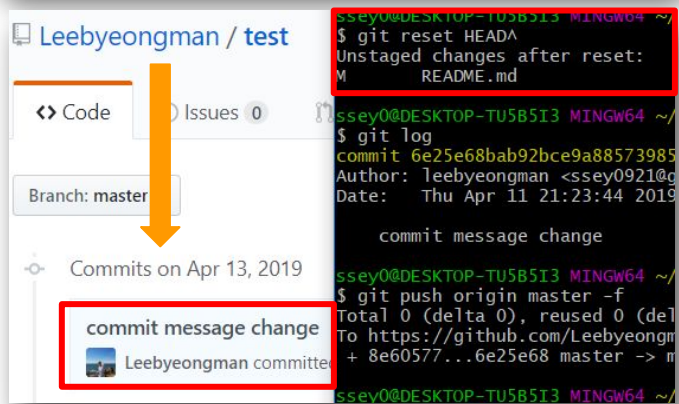
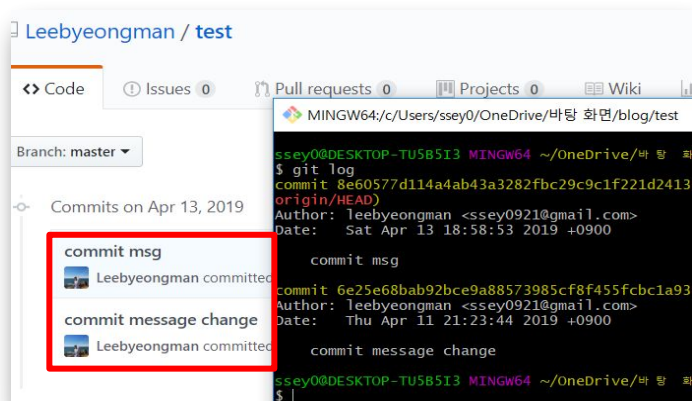
commit message change
```

그럴 때는 **commit** 취소를 해서 다시 **commit**할 수도 있겠지만, 같은 파일에 대해서 **commit message**만 변경한다면 이 명령어를 사용하는 것이 더 좋을 수 있어요.

명령어는 매우 간단합니다. 위와 같이 명령어를 입력하고, 변경하고자 하는 **message**를 작성 후 다시 **push**를 해주면 된답니다.



git push 취소



1. 워킹 directory에서 commit 되돌린다.
 - a. 가장 최근의 commit 을 취소하고 워킹 directory를 되돌린다. (기본옵션 : --mixed)
 - i. `git reset HEAD^`
 - b. 원하는 시점으로 워킹 directory를 되돌린다.
// Reflog(브랜치와 HEAD가 지난 몇 달 동안에 가리켰었던 커밋) 목록 확인
 - i. `git reflog` 또는 `git log -g`
// 원하는 시점으로 워킹 directory를 되돌린다.
 - ii. `git reset HEAD@[number]` 또는 `git reset [commit id]`
2. 원격 저장소에 강제로 push 한다.
 - a. `git push origin [branch name] -f` 또는 `git push origin +[branch name]`
ex) master branch를 원격 저장소(origin)에 강제로 push
`git push origin +master`

저도 이거는 거의 사용하지 않는데 혹시 몰라서 넣어봤어요. 이 명령을 사용하면 자신의 local의 내용을 remote에 **강제로 덮어쓰기**를 하는 것이기 때문에 **주의**해야 해요.

- 되돌리자마자 commit 이후의 모든 commit 정보가 사라지기 때문에 주의해야 해요.
- 특히, 협업 프로젝트에서는 동기화 문제가 발생할 수 있으므로 팀원과 상의 후 진행하는 것이 좋아요.



Q & A

질문이 있으시거나 추가했으면 하는 내용있으시면 카톡주세요... ㅎ

또 **Git**을 사용하고 싶은데 자료를 봐도 모르겠다~ 한다면 직접 알려줄게요 ㅋㅋ



The end