

개발툴(vs code나 intellij)의 터미널을 사용해도되고 아님 sourcetree나 맥의 터미널을 사용해도된다.
개발하기 위한 폴더를 하나 생성하고 그 폴더에 관해 깃허브 권한을 설정해줘야한다.

Git = version control system

→ 파일의 내용이 바뀌어도 파일의 이름을 바꾸지 않아도 그대로 파일의 이름은 그대로 유지되고 내용만 바뀌게 해줌

→백업(만약의 사태를 대비해서 코드가 날라가는 걸 방지해줌), recovery(이전상태로 쉽게 돌아가게 해줌),협업을 제공한다.

Git: 코드를 관리하기 위한 소프트웨어이자 도구(형상관리 툴이라고도 함), git이라는 소프트웨어를 설치하고 사용한다.

GitHub: 코드를 온라인에 저장할수있는 저장소

기본명령어(sourcetree를 사용하면 명령어를 쓰지 않아도됨, 깃을 편하게 쓰기위해 화면으로 보여주는 소프트웨어임)

Git init : 지금의 디렉토리 기준으로 git이 이 디렉토리를 관리(최초한번만 해주면 됨)

Git status : 업데이트된 파일 보여준다.

Git diff : commit 하기전 이전 코드와 현코드가 어떤차이있는지 알고 싶을 때 씬. 그럼 서로 차이점을 비교해줌

Git add : : 지금의 디렉토리를 기준으로 변경사항에 대해서 git이 추적(add 뒤에 경로 마침표로 명시=변경된 사항 모든 파일 등록한다)→다시말해 변경사항에 대해서 추적

Git add [filename] : filename 파일의 변경사항을 등록한다.

Git commit : 커밋은 변경사항에 대한 작업을 확정한다는 뜻

Git commit -m "첫번째 커밋" : 커밋할때 메시지이다. (수정된게 무엇인지 아님 첫번째로 올린건지 마음대로 올리면됨 뒤에는 메시지니깐, 주석이라 생각하면됨)

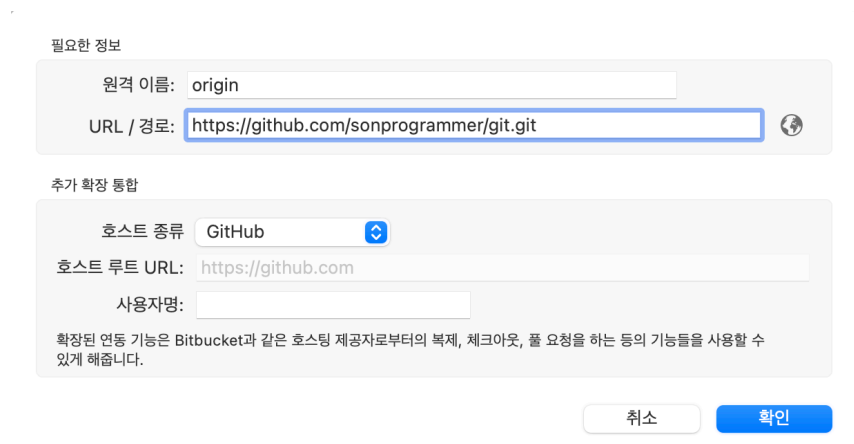
```
youngjinson@youngjinui-MacBookPro git 실습 % git add .
youngjinson@youngjinui-MacBookPro git 실습 % git branch
  creating-hello-file
* master
  test1
  test2
youngjinson@youngjinui-MacBookPro git 실습 % git commit -m "test1, hello"
[master 53336d7] test1, hello
 6 files changed, 40 insertions(+)
  create mode 100644 untitled/.idea/.gitignore
  create mode 100644 untitled/.idea/misc.xml
  create mode 100644 untitled/.idea/modules.xml
  create mode 100644 untitled/.idea/vcs.xml
  create mode 100644 untitled/untitled.iml
youngjinson@youngjinui-MacBookPro git 실습 % git branch
  creating-hello-file
* master
  test1
  test2
youngjinson@youngjinui-MacBookPro git 실습 %
```

github에 코드 올리기

1. Git remote add origin [git 주소] : 깃허브저장소와 연결하는 명령어(여기서 origin은 긴git레파지토리주소를 쓰기 번거로우니 origin이라는 이름으로 부르겠다는 의미)
즉 다시말해 우리의 local과 깃허브의 리파지토리 주소를 연결함

<sourcetree로 쓰면 이렇게 하면됨

(원격에 연결되어 있는지 확인하기 위해서 원격에 origin이 있는지 확인하기>



필요한 정보

원격 이름: origin

URL / 경로: <https://github.com/sonprogrammer/git.git>

추가 확장 통합

호스트 종류: GitHub

호스트 루트 URL: <https://github.com>

사용자명:

확장된 연동 기능은 Bitbucket과 같은 호스팅 제공자로부터의 복제, 체크아웃, 풀 요청을 하는 등의 기능들을 사용할 수 있게 해줍니다.

취소 확인

2. Git remote -v :

그리고 난 후에 개발툴(vscode or IntelliJ)의 터미널에서 git remote -v를 하면 origin이라는 이름에 url이 작성되었는걸 확인 가능—아님 맥 터미널 들어가서 그 폴더에 접근후에 git remote -v 써도 됨

3. Git push -u origin main(브랜치 이름) : origin(github 리파지토리)뒤에는 깃허브 브랜치 이름(여기서는 main으로 해놔서 main으로 한거임) 을 적어두면됨

※git branch : 브랜치확인하는 방법(개발툴 터미널창에서 아님 맥 터미널 들어가서 그 폴더에 접근후에 git branch써도 됨)

4. Git push origin main(브랜치 이름) : 터미널에서 입력해도 됨

<Sourcetree에선 푸시라는 버튼을 누르면됨>

Git push —help : git명령어에 관련한 것

github에 파일 올리기

1. 파일 폴더 들어가서 git init하기
2. Git add <파일명> 하기, git commit -m “주석설명”
3. Git remote -v(현재 깃헙이랑 연결되었는지 확인, 아무것도 안뜨면 연결x)
4. Git remote add origin github리파지토리 주소
5. Git branch -M main
6. Git push -u origin main

브랜치란?

현재 작업을 하고 있는 위치이자 작업의 줄기

- 각각의 브랜치는 작업 영역이 독립적이고 다른 브랜치에 영향을 끼치지 않는다.
- 가장 핵심 작업의 줄기가 중앙 줄기(흔히 master 브랜치라함)

→ 중앙 줄기 다시말해 중앙 브랜치에 영향이 없도록 브랜치를 만들어서 작업을 수행해야한다.

Why? 첫번째 이유 : master 브랜치가 현재 운용중인 코드로 사용되기 때문(현재 사용되는 코드를 다른사람이 커밋해서 바꾸면 안되기 때문)

두번째 이유 : 협업을 하는 경우

브랜치 기본명령어

Git branch : 현재 branch 확인

Git branch creating-hello-file : creating-hello-file라는 브랜치 생성(예를들어 master branch이런거 같은건데 여기서는 creating-hello-file이라는 브랜치가 생성된것임)

Git checkout creating-hello-file : branch 위치 변경(master 브랜치 -> creating-hello-file브랜치로 변경한다는 것임)

```
youngjinson@youngjinui-MacBookPro Desktop % cd git\ 실습
youngjinson@youngjinui-MacBookPro git 실습 % ls
untitled
youngjinson@youngjinui-MacBookPro git 실습 % git branch
* master
youngjinson@youngjinui-MacBookPro git 실습 % git branch creating-hello-file
youngjinson@youngjinui-MacBookPro git 실습 % git branch
creating-hello-file
* master
youngjinson@youngjinui-MacBookPro git 실습 % git checkout creating-hello-file
'creating-hello-file' 브랜치로 전환합니다
youngjinson@youngjinui-MacBookPro git 실습 % git branch
* creating-hello-file
master
youngjinson@youngjinui-MacBookPro git 실습 %
```

Merge 명령어

Git checkout master : master브랜치로 변경

Git merge creating-hello-file : creating-hello-file브랜치 병합

※브랜치 작업시 항상 현재 브랜치가 어디인지 git branch명령어나 sourcetree를 이용해서 자주 확인해야 브랜치가 엉키는 문제를 방지할 수 있다

Pull request(협업을 하기위한 깃허브 플로우임)

1. [로컬=각자의 개인 컴퓨터] : 새로운 작업 브랜치에서 코드 작성 진행
2. [로컬=각자의 개인 컴퓨터] : 작업 완료후 commit 만들고 깃허브 저장소에 작업 브랜치 push
3. [github] : 새로운 브랜치 push 시, 저장소에 pr만들기 버튼 표시됨
4. [github] : PR만들기 버튼 클릭 후, PR생성
5. [github] : PR메뉴에서 새롭게 생성이 된 PR을 확인할 수 있으며, 협업자들이 코드 리뷰 진행
6. [github] : 코드리뷰 완료 후, merge하기 버튼을 클릭함으로써 merge(작업 브랜치 — (merge) —> master브랜치) 진행
7. [github] : master 브랜치에 새로운 기능 적용 완료

pull(저장소에 저장되어 있는 작업 내용을 가져와 병합작업을 실행한다. 쉽게 말해 local repository(내 컴퓨터) 에 remote repository(깃허브 저장소)의 내용을 덮어 씌운다고 할수 있다.)

Git pull origin(깃허브 저장소 이름) master(pull하고 싶은 branch명)

Ex) 협업하는 동료가 jin이라 하면 동료의 작업 내용을 자신의 로컬 리포지토리(내 컴퓨터)에 가져오면서 병합을 하고자 하면

—>git pull jin main(가져오고자하는 브랜치명)이라 하면 된다.

Git log(현재까지 commit된 내역들을 터미널 창에 출력해주는 명령어, 이후 로그 창에서 벗어나고 싶다면 q를 입력하면된다.

명령어 : Git log

```
youngjinson@youngjinui-MacBookPro ~ % cd Desktop/git\ 실행
youngjinson@youngjinui-MacBookPro git 실행 % git log
commit ea9b6266c44fe8f1e02eba8b21d4a367039f8100 (HEAD -> master)
Merge: fe700b9 fdda5a5
Author: 손영진 <ods04138@gmail.com>
Date:   Fri Jan 27 19:05:01 2023 +0900

    Merge commit 'fdda5a54d28bf16a611ee8344e7646b040101d36'

commit fe700b9a8a85ab2c85df49b3e134e6e89abcb57
Merge: 4c2fe06 738f459
Author: 손영진 <ods04138@gmail.com>
Date:   Fri Jan 27 19:04:50 2023 +0900

    Merge branch 'son'

commit 4c2fe06dd0f8fb472dc99c9574f3a6cb1aba2c94
Author: 손영진 <ods04138@gmail.com>
Date:   Fri Jan 27 18:56:54 2023 +0900

    no message

commit fdda5a54d28bf16a611ee8344e7646b040101d36 (origin/master)
Merge: 324cbdf 738f459
Author: sonprogrammer <97154156+sonprogrammer@users.noreply.github.com>
Date:   Fri Jan 27 18:34:06 2023 +0900

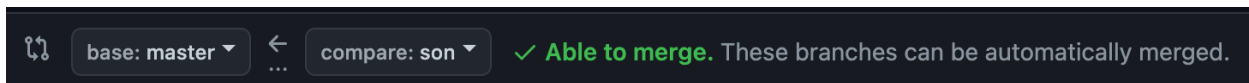
    Merge pull request #1 from sonprogrammer/son
```

협업을 위한 자기만의 로컬 브랜치 만들고 병합하는 법

1. Git branch son(자기가 만든 브랜치이름) : 브랜치를 생성한다
2. Git checkout son : 생성한 브랜치로 브랜치 위치 변경
3. (Git branch : 자신의 브랜치 위치 확인) -> 권장사항이지만 항상 자기의 브랜치 위치를 확인하는 것이 좋다.
4. Git commit -m "hello"(" " 사이에는 자기가 적을 문구를 적는거임 쉽게 말해 주석, 즉 설명같은거임, commit은 필수지만 뒤 메시지는 필수 아닌 권장사항임)
5. Git push origin son : son 브랜치를 깃허브 주소인 origin에 push를 하겠다는 것. 즉 origin에 내가 만든 브랜치를 올려서 만들겠다는 것이다. (그럼 그 주소에서 아래와 같이 요청이 들어온다.)



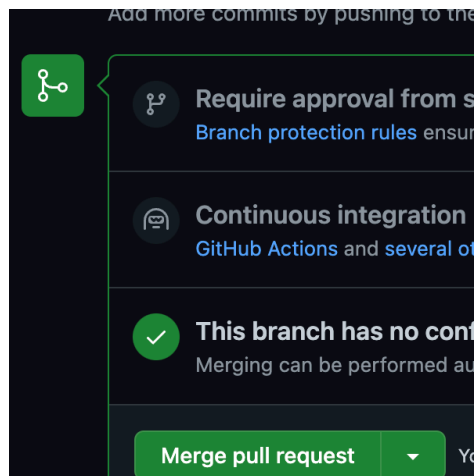
6. 그 이후에 위 사진의 녹색 버튼을 눌러 pr을 생성할 수 있다.



※위 사진에서 내 브랜치에서 어느 브랜치로 합병할건지 잘 확인해야한다.

PR에서 코드리뷰를 하고 댓글도 남길수 있다

그런 후에 아래 사진에 있는 녹색 버튼(merge pull request)를 누르면 master 브랜치와 병합할 수 있다.



Git rebase(pull request 충돌 해결하기위한 기법임)

rebase는 커밋을 재배포(rebase)하기 위해서 사용을 한다. 재배포라 함은 커밋의 위치를 다시 배치하는 작업이다.

Rebase 명령어

#충돌이 없는 경우

Git checkout added-task2-printing

Git rebase master //바로 rebase성공

#충돌이 있는 경우

Git checkout added-task2-printing

Git rebase master //충돌로 인해서 rebase실패→ 병합 작업 필요

#코드 상에서 충돌 코드 직접 정리 수행

Git add .

Git rebase --continue //rebase성공

작업플로우

1. 작업 플로우

1) [github] - 개발 진행할 레파지토리 fork -> <https://github.com/mcocl/awesome-pr-whoami>

2) [로컬] - fork 한 레파지토리를 로컬로 clone

```
git clone https://github.com/[내 github ID]/awesome-pr-whoami.git
```

3) [로컬] - remote 명령어를 통해서 upstream 생성

```
git remote add upstream https://github.com/mcocl/awesome-pr-whoami.git
```

4) [로컬] - remote(원격지)의 origin, upstream 확인

```
git remote -v
```

5) [github] - 개발 프로젝트 내 작업할 issue 생성 (일종의 티켓)

6) [로컬] 작업 브랜치 생성

7) [로컬] 작업 브랜치 내에서 작업 후 커밋 생성

8) [로컬] 내 레파지토리(origin)로 push

9) [github] PR 생성 후 merge 완료

10) [로컬] master 브랜치 upstream pull 진행 후, origin의 master push 진행 (origin 최신 소스 유지)

위에 upstream은 원격 이름이다. (내가 정하는 거임)

깃 stash(commit 하기전 임시저장을 위한 stash)

- commit을 하기전에 다른 branch로 이동이 안된다. 그 때 사용하는 것이 stash이다.(stash list를 사용하면 임시 저장된 작업 내용을 볼 수 있고 필요할 때 꺼낼 수도 있다.)

명령어

Git stash : 현재 작업 내용을 임시 저장(commit이 아님)

Git stash list : 임시 저장된 작업 리스트 확인

Git stash pop : 마지막에 저장한 작업 불러오기

깃 플로우와 브랜치 전략

대표적인 브랜치 명 예로

Develop : 현재 개발중인 브랜치명

Feature : develop에서 만들어진 기능 개발 브랜치

Release : 릴리즈(서비스 공개) 준비 브랜치

Hotfixes : 버그 수정 브랜치

Master : 최신 운영 브랜치

clone하는 법

명령어

- git clone giturl

협업하기!!



정리

기본셋팅



반복작업

팀원 : Pull Request , PR

Able to merge

팀장 : merge

conflict

팀원 : 로컬에 pull 하고
conflict 해결

팀원 : 다시 push 후에 PR

Ep.3 Conflict 해결하기

팀장

팀원

Can't automatically merge

Conflict

로컬에 Pull

Conflict 전부 해결, 수정

Merge 진행

Able to Merge

팀장 레포의 지정 브랜치에

Pull Request 보냄

Ep.4 여러 명과 협업하기

Merge 진행

팀장

가정 : 각자 브랜치에 push 해둔 상황

merge

PR

PR

1

팀원 로컬에서 팀장브랜치를 다시 pull
↓
Conflict를 로컬에서 직접 해결

2

팀원 로컬에서 팀장브랜치를 다시 pull
↓
Conflict를 로컬에서 직접 해결

3

pull
↓
Conflic 해결

