KWEB Study: Week 1

1. Before Study
2. Module in Node.js
3. yarn 이란?
4. Git 기초
5. 실습 및 과제

Module, Yarn, Git

KWEB 2학기 준회원 스터디

Designed by M.G. BAE

Before Study

- 2018 KWEB 2학기 준회원 스터디
  - ✓ 수 7시: 15 배민근 (baemingun@naver.com)
  - ✓ 목 7시: 15 임경섭 (officialbusiness@naver.com)

- 0주차에선 2학기 스터디에 사용할 Node.js를 소개하고 다른 개발환경들과 함께 설치해보았습니다.

- 1주차의 목표는 Node.js를 활용한 프로젝트에 필요한 기본기에 대해 접해보고 실습해보는 것입니다.
  - ✓ Node.js의 Module, yarn (+ npm), Git 기초(, 다음 스터디에선 비동기와 ECMAScript!)
  - ✓ 오늘은 이론보단! 실습의 비중이 높을 예정입니다~

- 2학기에도 역시 OUT COUNT 제도를 사용합니다. 출석 & 과제 열심히 해주세요!
  - ✓ 여러분 모두 정회원에서 뵙고 싶으니 파이팅 하세요~ㅎ

Module이란?

- 아래는 0주차에서 사용했던 Node.js의 예시 코드입니다. 빨간 테두리에 집중해봅시다.

```
const http = require('http');

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello World!\n');
});

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

Module이란?

- Node.js에서의 모듈
  - ✓ 관련된 코드들을 하나의 코드 단위로 캡슐화 하는 것으로 메서드와 속성을 미리 정의해 놓은 것

- 즉, 모듈을 사용하면 프로그램의 여러 부분에서 쓰이는 코드조각들을 한대 모아두고 사용하고자 하는 곳에서 불러와 사용 할 수 있습니다.

- Node.js의 모듈들은 일반적으로 require를 통해 불러와 사용합니다. (2018.09. 기준)
  - ✓ ex) const fs = require('fs');

- (다음 스터디 주제인 ECMAScript에서 다루겠지만 앞으로 변수 선언 시 var의 사용은 최대한 없을 것입니다. 대신! const나 let을 사용해 선언해주세요~ 모듈은 참조형이므로 const를 사용하도록 하겠습니다.)

내장 모듈

- Node.js 이미 개발진들이 만들어 놓은 다양한 내장 모듈들이 있습니다. 아래 URL에 모든 정보가 있습니다.
  - ✓ 최신버전 공식문서 URL: https://nodejs.org/api/
  - ✓ LTS버전 공식문서 URL: https://nodejs.org/dist/latest-v8.x/docs/api/

- 예제 코드에 있던 http도 내장 모듈이라 할 수 있죠! → const http = require('http');
  - ✓ 실습 때 언급하겠지만 require 키워드를 통해 모듈을 불러와 사용합니다. (ES 표준에 따르는 import 키워드도 지원하기 시작했으나 아직은 require가 일반적입니다.)

- 모든 걸 다 가르칠 순 없으니! (원래 다 찾아가면서 하는 거죠. 코드를 적절한 곳에 잘 갖다 써서 기능을 구현하는 개발자도 유능하고 실력 있는 개발자입니다. 실제로 많은 개발이 이 방식으로도 이루어지고요~)

- 우리는 오늘 그 중에 fs모듈과 path모듈을 지금 살펴보고! 좀 있다 실습 때 써먹어보겠습니다~

fs 모듈

- fs 모듈: 파일을 읽고 쓰는 사용하는 모듈, fs는 File System 줄임말입니다.
  - ✓ 공식 문서 URL: https://nodejs.org/dist/latest-v8.x/docs/api/fs.html

- fs 모듈 아래에는 엄청나게 많은 메소드가 있어서 홈페이지에서 더 찾아보시고! 지금은 오늘 실습 때 사용할 2가지만 한번 짚어봅시다.
  - ✓ fs.readFileSync(file, encoding); → 파일을 동기적으로 읽어옵니다.
  - ✓ fs.readFile(file, encoding, callback); → 파일을 비동기적으로 읽어옵니다.

- 비동기적으로 파일을 읽으면 이벤트 리스너를 등록하고 파일을 모두 읽은 뒤 후처리를 할 수 있습니다.
  - ✓ 무슨 말이냐고요? 다음주부터 알아가는 걸로. 굳.

path 모듈

- path 모듈: 파일의 path를 다루는 모듈입니다.
  - ✓ 공식 문서 URL: https://nodejs.org/dist/latest-v8.x/docs/api/path.html

- path 모듈에도 여러가지 메소드가 존재하나 밑의 예제 코드에 있는 4가지만 살펴봅시다.
  - ✓ path.join(): 여러 개의 이름들을 모두 합쳐 하나의 파일 path로 만들어주는 메소드
  - ✓ path.dirname(): 파일 path에서 디렉토리 이름을 반환해주는 메소드
  - ✓ path.basename(): 파일 path에서 파일의 확장자를 제외한 이름을 반환해주는 메소드
  - ✓ path.extname(): 파일 path에서 파일의 확장자를 반환해주는 메소드

```
const path = require('path');

let my_file = "C:\\project\\kweb\\tired.txt";
let dirname = path.dirname(my_file);
let basename = path.basename(my_file);
let extname =  path.extname(my_file);

console.log(`path.dirname = ${dirname}\npath.basename = ${basename}\npath.extname = ${extname}`);
```

외장 모듈

- 내장 모듈가지고만은 모든 서비스를 구현할 수 없으니 당연히! Node.js 개발진이 아닌 전세계 개발자들이 만든 다양한 외장 모듈도 존재합니다.
  - ✓ Node.js는 생태계에 힘입어 엄청나게 많은 외장 모듈을 보유하고 있습니다. 세상은 넓고 능력자는 많..

- 외장 모듈은 npm이나 yarn 같은 패키지 매니저를 활용하여 나의 프로젝트 내부에 가져온 이후, 내장 모듈과 동일하게 require을 이용해 사용합니다.
  - ✓ 패키지 매니저가 뭐냐구요? 다다음 슬라이드부터 살펴보도록 합시다~

- 앞으로 진행될 스터디에서 매주 새로운 외장 모듈들이 등장할 예정입니다~

- 외장 모듈 사용은 불러온 이후로는 내장 모듈과 같은 방법으로 사용하면 되므로 실습 때 내장 모듈과 함께 살펴보도록 합시다!

Module 만들기?

- 그렇다면 나는 Module을 만들 수 없나? → 그랬으면 외장 모듈들이 있을 리가 없다!
- 패키지 매니저 쪽에 올리는 퀄리티는 우리가 못 뽑더라고! 가볍게 어떤 식으로 만드는지에 관해 기초는 알아야 겠죠?
- 좀 이따 진행할 실습 2에서 간단한 계산기 모듈을 제작해봅시다!
- 오른쪽 코드는 정말로 간단한 모듈 예제입니다. 일명! 더하기 함수!
  - ✓ require로 불러와 정의된 add 기능을 사용할 수 있습니다.

```
const calc = {};

calc.add = function(a,b) {
    return a + b;
}

module.exports = calc;
```

Node.js의 생태계

- Node.js의 장점 중 하나는 전 세계 수많은 사용자들이 만든 모듈을 쉽게 사용할 수 있다는 점입니다.
- 그럼 이 모듈들을 어떻게 내가 가져와서 사용하나? → 전전 슬라이드에서 언급했듯 패키지 매니저를 이용!
- Package란? → 관리를 위해 모듈에 몇 가지 정보를 추가한 것으로 결국 모듈이지만 조금 더 큰 단위
- Node.js에는 이미 모듈들을 관리해주는 패키지 매니저들이 존재하며, 패키지들은 공유를 위해 인터넷에 올리기도 하는데 이를 패키지 매니저를 통해 내려 받아 Local에서 이용이 가능합니다.
  - ✓ **정리하자면!** 패키지 매니저를 통해 외부 모듈을 사용할 수 있다는 의미입니다.
- 전세계에서 가장 많이 사용하는 Javascript의 패키지 관리자로는 npm과 yarn이 존재합니다.
  - ✓ 이번 KWEB 준스에서는 yarn을 기본으로 사용할 예정입니다.

잠깐! npm

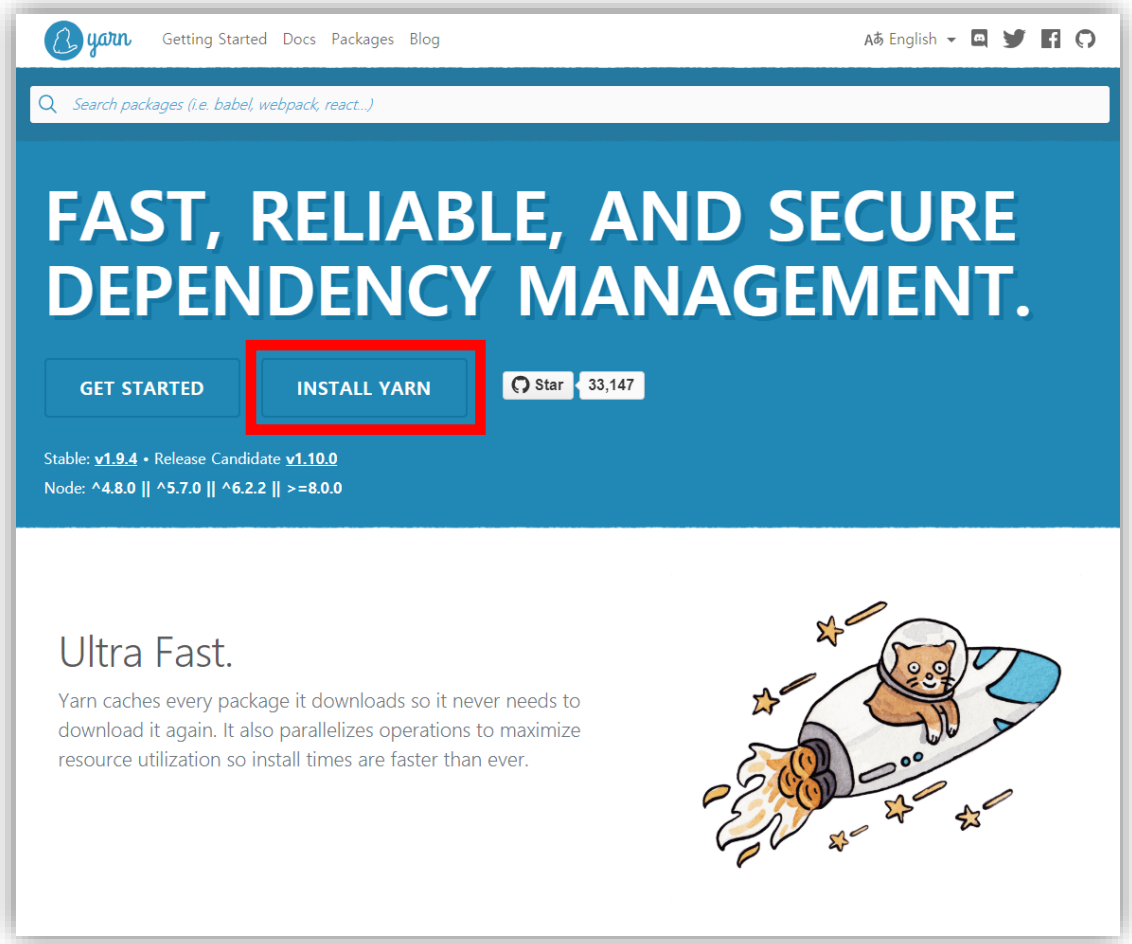
- Node.js 를 위한 패키지 관리 및 배포 시스템입니다. 이름부터 npm (Node.js Package Manager)으로 Node.js를 위해 개발되었습니다.
- 전통적인 자바 스크립트 패키지 매니저로 인만큼 2011년 공개되어 Node.js 에 날개를 달아주었습니다.
- 여전히 널리 사용되며 다양한 외부 모듈들을 npm install을 통해 Local에서 사용할 수 있습니다.
- 하지만 기술이 발전함에 따라 npm에는 한계가 존재했으며 이는 yarn 개발의 배경이 되었습니다.
  - ✓ 성능 이슈 / 비결정적 설치 / 버저닝 이슈 / offline 또는 외부 방화벽이 막혀 있는 서버에서 의존성 모듈들을 받아오는 문제
- **yarn 명령어는 실습에서 배우겠지만 npm 명령어는 다루지 않습니다. 하지만 알아야 하니 각자 찾아보세요!**

yarn 설치

- 뒤에 실습 때 사용할 수 있게 yarn을 설치해봅시다.

- CLI 환경에서 설치하는게 좋지만 아직 CLI와 익숙하지 않으실테니 인스톨러로 설치 ㄱㄱ
  - ✓ URL: https://yarnpkg.com/en/
  - ✓ **MAC은 CLI 설치만 지원합니다.**
  - ✓ CLI 사용하시고 싶은 분들은 INSTALL YARN 페이지에 인스톨러와 함께 안내되어 있으니 참고하세요!

- 설치 진행상황 중 특이한 건 없으니 그냥 쭉 하시면 됩니다~



yarn

- Facebook이 주도해 개발한 Javascript의 새로운 패키지 매니저입니다.
  - ✓ Facebook은 코드베이스가 커지면서 npm을 사용할 때 일관성, 보안, 성능에 문제를 겪게 되었고 Google, Exponent, Tilde의 개발자들과 함께 npm client를 대체할 새로운 패키지 매니저 yarn을 만들었다고 합니다.

- yarn은 이미 널리 쓰이고 있는 npm과 비교했을 때, 더 빠르고 안정적으로 동작합니다.

- **짚고 넘어가기!** yarn.lock 이란? → yarn install 명령 후 생성됩니다.
  - ✓ 실제로 설치된 node_modules 모듈들의 버전을 저장해 어디에서나 같은 버전 같은 구조의 의존성을 갖게끔 해주는 기능입니다.
  - ✓ 자세하게 들어가면 어려우니까 요정도만 알아놓읍시다.

- yarn은 패키지 매니저라 굳이 자세한 설명을 하진 않을 것입니다. (자세한 내부사항은 어렵기도 하구요!)
  - ✓ 더 궁금하신 분들은 각자 찾아보는 걸로!

Git 기초

- 이번엔 Node.js 를 약간 떠나서 지난 주에 Git Bash 라는 것을 설치했던 것 기억하시나요?
  - ✓ 설치 안 하셨으면..... 빠르게 몰래 ㄱㄱ
- 1주차의 마지막 Topic으로는 실무에서 흔히 사용되는 Git 이라는 것에 대해 살펴볼 것입니다.
- 솔직히 뒤에 명령어들도 있지만 그걸 오늘 절대 다 이해할 수 있을 거란 생각은 안 합니다! 그냥 오늘은 따라만 오시고! 과제 1을 통해 좀 더 이해하실 수 있길 바랍니다~
- 이번 2학기 준스는 GitHub를 통해 과제 제출을 받을 예정입니다. 실무에서도 Git은 많이 사용하니 잘 배워두세요~
  - ✓ (일단 바람은 그렇지만 준회원 여러분들이 너무 버거워 보인다고 판단되면 E-mail 제출로 변경할 예정입니다.)

Git

- 컴퓨터 파일의 변경사항을 추적하고 여러 명의 사용자들 간에 해당 파일들의 작업을 조율하기 위한 분산 버전 관리 시스템 → 효과적인 협업도구!
  - ✓ 좀 더 쉽게 말하면! 여러 명의 개발자(분산)가 협업하여 개발하면서 버전을 관리할 수 있는 시스템

- 소스코드의 효율적인 관리를 위한 형상 관리 도구(Configuration Management Tool)이며, 어떠한 집합에서 파일의 변경사항을 지속적으로 추적하기 위해 사용될 수 있습니다.

- GitHub, Bitbucket, GitLab 등 Git을 사용하는 프로젝트를 지원하는 웹 호스팅 서비스들(Git 저장소!)도 존재하며, Source Tree, GitHub Desktop 등의 GUI도 지원하고 있습니다.

- 오늘 배울 내용 중에는 적어도 아래는 기억합시다!
  - ✓ "작업한 내용을 **스테이지**에 올려서 **로컬 저장소에 커밋**하고, 이를 **푸시해서 원격 저장소로** 보낸다."
  - ✓ git add → git commit → git push

Git 명령어 기초

- Git 저장소 초기화 → git init
  - ✓ 저장소나 디렉토리 안에서 이 명령을 실행해야 다양한 Git 명령어들을 사용할 수 있다.
- Git 설정 → git config
  - ✓ Git의 여러가지 기본 설정들을 할 수 있는 명령어입니다.
- Git 주요 명령어 목록 보기 → git help
  - ✓ “git help init”이나 다른 용어를 타이핑하여 특정 깃 명령어를 사용하고 설정하는 법을 이해할 수도 있다.
- Git 저장소 상태 체크 → git status
  - ✓ 어떤 파일이 저장소 안에 있는지, Commit이 필요한 변경사항이 있는지, 어떤 Branch에서 작업하고 있는지 등을 볼 수 있다.
  - ✓ Branch는 Git 입문 단계를 마치고 알아보도록 합시다.

Git 명령어 기초

- Git에 파일의 추가 및 변경사항 반영하기 → git add
  - ✓ Git은 파일의 존재 및 수정사항을 모른다. 파일의 변경사항을 Git이 단순히 추적만 하게 해주는 명령어

- Commit하기(스냅샷을 찍는다고 이해하면 됩니다!) → git commit
  - ✓ **Git에서 가장 중요한 명령어 중 하나!** 변경사항이 있다면 저장소의 스냅샷을 찍기 위해 사용하는 명령어입니다.
  - ✓ 보통 "git commit -m "{메시지}" 형식으로 사용합니다. 여기서 {메시지}가 해당 commit의 설명(제목?요약?)이 됩니다. -m은 명령어의 그 다음 부분을 메시지로 읽어야 한다는 것을 의미합니다.

- Push하기(온라인 저장소에 작업들을 밀어 넣기!) → git push
  - ✓ 이 명령어로 Local에서의 작업의 변경사항(Commit)을 GitHub와 같은 온라인 저장소에 반영합니다.

- Pull하기(온라인 저장소에서 작업들을 받아 오기!) → git pull
  - ✓ 이 명령어로 GitHub와 같은 온라인 저장소의 최신 버전을 Local에서 작업환경으로 다운로드합니다.

실습

Module in Node.js

실습

- 오늘의 실습은! 그냥 배운 거 써먹기 입니다. (ㅋㅋㅋㅋㅋㅋ)

- Node.js의 내부 모듈 & yarn으로 설치한 외부 모듈을 사용해보고, 우리끼리도 모듈 하나 만들어보고~ 마지막으론 실습한 거 GitHub에 올리기! 가 이번주의 목표입니다.

- 오늘은 크게 실습 1,2,3으로 나뉘어져 있습니다. (독립적인 실습이 아닌 순차적으로 덧붙이는 실습입니다!)
  - ✓ **실습 1 – Module 사용해보기**: Node.js의 모듈을 사용해보기
  - ✓ **실습 2 – Custom Module 만들기**: 간단한 계산기 모듈 만들어 보기
  - ✓ **실습 3 – GitHub 맛보기**: 실습 1,2에서 만든 프로젝트를 GitHub에 업로드해보기

- (오늘도 0주차와 마찬가지로 실습이 거의 과제와 같습니다. 실습 2에 기능 추가해서 실습 3의 Repository에 추가적으로 업로드하는게 과제입니다~)

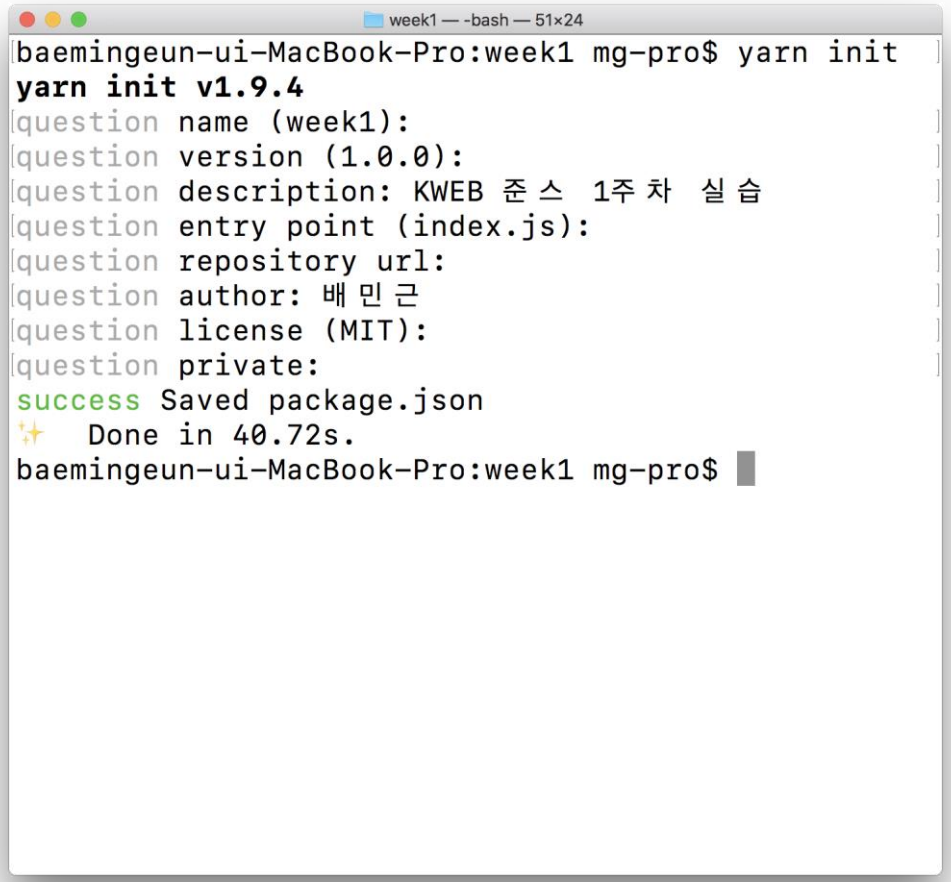
실습 1 – Module 사용해보기

- Module 사용에 앞서 Node.js를 사용하는 프로젝트를 yarn을 이용하여 생성해봅시다.

- 프로젝트를 생성할 위치에서 디렉토리(오른쪽 예제에선 week1)를 하나 생성한 후 Git Bash (혹은 Terminal)을 열어 아래 명령어를 입력해봅시다.
  - ✓ yarn init

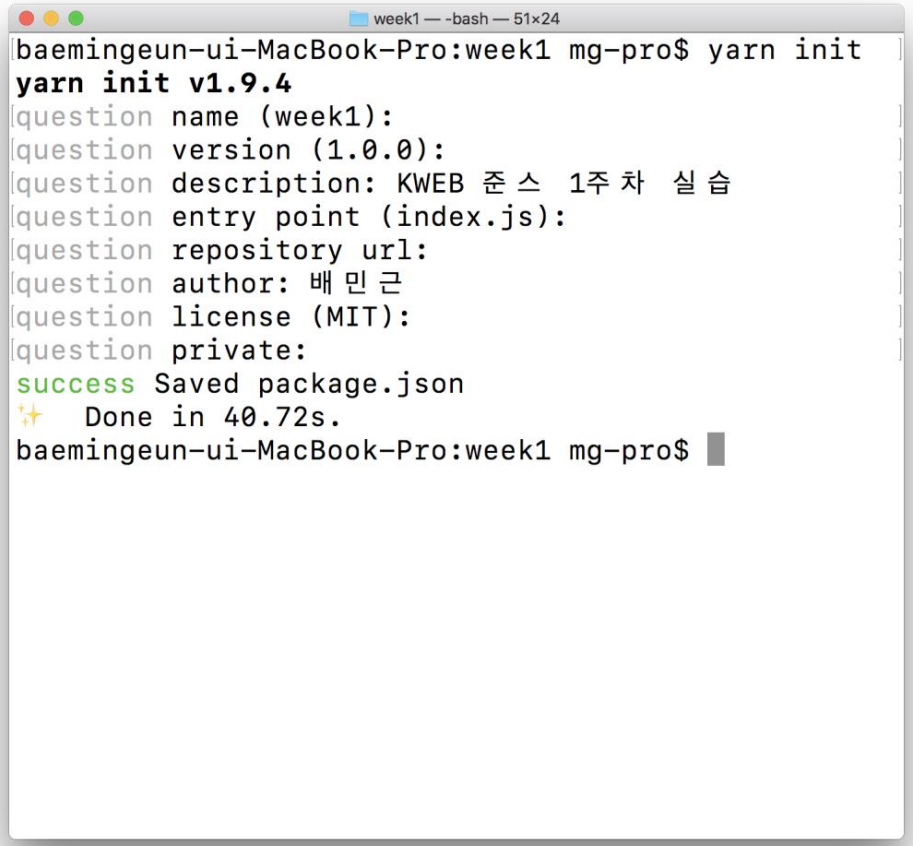
- 영어할줄 아시죠? 원하시는 대로 맞게 설정하시면 됩니다~ (name, description 등등)

- 그러고 나면! package.json이 생성됩니다.



실습 1 – Module 사용해보기

- 한번 생성된 package.json과 우리가 yarn init로 입력했던 값을 비교해봅시다.



```
{
  "name": "week1",
  "version": "1.0.0",
  "description": "KWEB 준스 1주차 실습",
  "main": "index.js",
  "author": "배민근",
  "license": "MIT"
}
```

참고! yarn 명령어 기초

- 초기화 → yarn init
  - ✓ package.json을 생성하는 대화형 프로세스가 진행된다. 이미 pacakge.json이 있을 경우 재사용한다.

- 모듈 설치 → yarn install
  - ✓ 참고! yarn install –flat → 한 모듈당 한 버전만 설치하는 install (prompt를 통해 의존성분석에서 발견된 버전 중 설치할 버전을 선택)

- 의존성 추가 → 제품에 추가할 의존성과 개발에만 쓸 의존성을 선택할 수 있다.
  - ✓ production → yarn add {모듈명}
  - ✓ development → yarn add –dev {모듈명}

- 의존성 삭제 → yarn remove {모듈명}

- 의존성 업그레이드 → yarn upgrade

실습 1 – Module 사용해보기

- 자! 이제 목표는 내장 모듈을 사용해보는 것입니다.
  - ✓ fs 모듈과 path 모듈을 사용해봅시다~
  - ✓ Node.js의 예제 코드를 수정해서 브라우저에 표시되는게 Hello, world! 가 아닌 우리가 제작한 HTML 파일을 띄우도록 해봅시다.

- yarn init 한 폴더에 index.js와 index.html 2개의 파일을 생성해봅시다. 이후 아래와 같이 수정!
  - ✓ index.js: 오른쪽 파일과 같이 수정
  - ✓ index.html: 여러분이 원하시는 대로~

- 그리고! Git Bash (또는 Terminal)을 열어 아래 명령어를 입력해봅시다.
  - ✓ node index.js

```
const http = require('http');
const fs = require('fs');
const path = require('path');

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  //res.end('Hello World!\n');

  const indexContent = fs.readFileSync(path.join(__dirname, 'index.html'));
  res.end(indexContent);
});

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

실습 1 – Module 사용해보기

- 어? 그럼 아래와 같이 이상하게 실행될 겁니다.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>KWEB 1二쇱감 - 쓰뤷 1</title>
</head>
<body>
    <h1>Hello, KWEB!</h1>
</body>
</html>
```

- 지난주 과제와 같이 header 부분을 오른쪽과 같이 고쳐봅시다.
  - ✓ header 부분의 자세한 사항은 3주차에서!

```
const fs = require('fs');
const path = require('path');

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/html');
  //res.end('Hello World!\n');

  const indexContent = fs.readFileSync(path.join(__dirname,'index.html'));
  res.end(indexContent);
});

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

- 다시 node index.js를 입력해봅시다.
  - ✓ 올바르게 html 파일이 출력되면 성공~

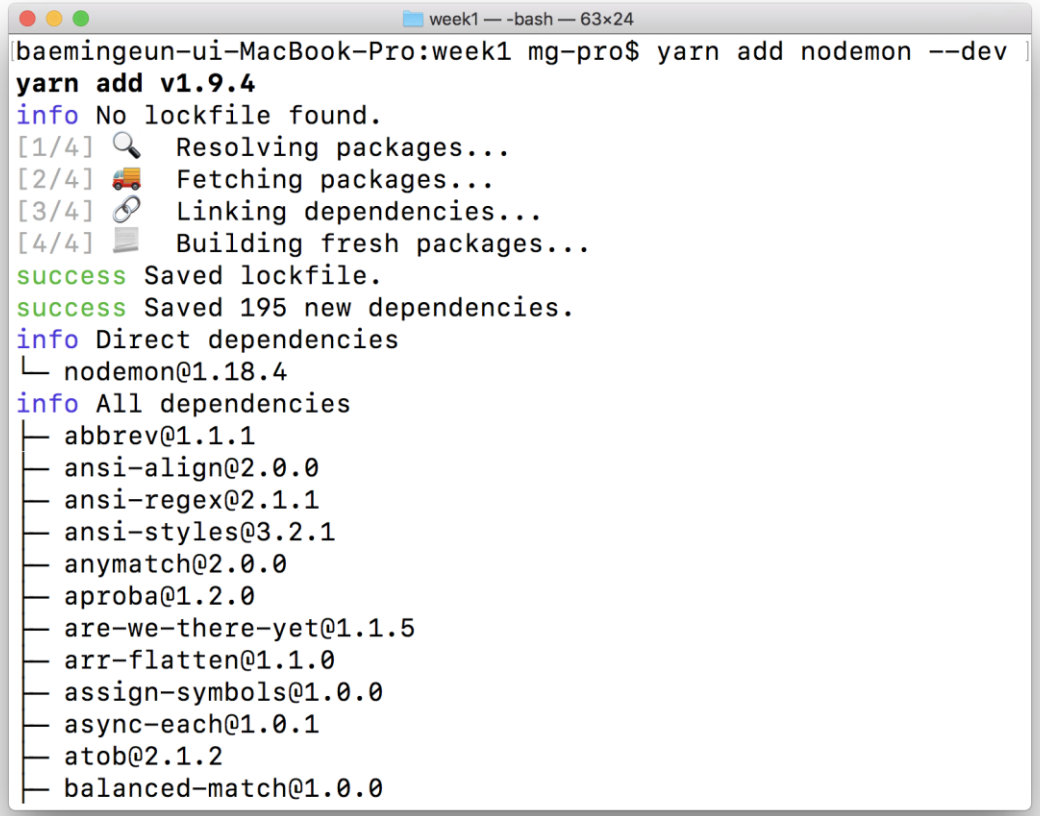
실습 1 – Module 사용해보기

- 얼렁뚱땅 내장 모듈을 사용해보았습니다!

- 이번엔 개발 시에 좀 더 편리하게 해주는 도구인 nodemon 모듈을 설치해보겠습니다.

- yarn 통해 nodemon 설치해봅시다. Git Bash (혹은 Terminal)을 열어 아래 명령어를 입력해봅시다.
  - ✓ yarn add nodemon --dev
  - ✓ nodemon은 개발 시에만 사용할 모듈이므로 –dev 를 붙여서 설치합시다~

- 다 설치되면, nodemon index.js 를 입력해 실행시켜봅시다.



실습 1 – Module 사용해보기

- 안 되죠? ㅋㅋㅋㅋㅋ 지금은 단순히 개발 의존성에만 추가했기 때문에 nodemon 명령어를 못 찾겠다고 나옵니다. (bash는 모르니까요!)
  - ✓ global 옵션을 붙여 설치했으면 실행이 될 겁니다!

- 이 참에 script에 대해 맛볼겸, package.json에 오른쪽과 같은 코드를 추가해보겠습니다.

- 그리고 아래 명령어를 각각 실행해 비교해보세요!
  - ✓ yarn start
  - ✓ yarn start:dev

```
{
  "name": "week1",
  "version": "1.0.0",
  "description": "KWEB 준스 1주차 실습",
  "main": "index.js",
  "author": "배민근",
  "license": "MIT",
  "devDependencies": {
    "nodemon": "^1.18.4"
  },
  "scripts": {
    "start" : "node index.js",
    "start:dev": "nodemon index.js"
  }
}
```

실습 1 – Module 사용해보기

- 기존의 node 명령어는 파일이 바뀌면 매번 현재 실행되는 프로세스를 종료하고 다시 실행해야 하는 번거로움이 있습니다.

- 개발할 때 위와 같으면 귀찮죠? 따라서 변경사항이 있으면 알아서 재실행해주는 nodemon이라는 모듈을 설치했습니다.
  - ✓ production으로 나갈 모듈은 아니니 개발 시에만 적용되게 yarn add --dev 로 설치했습니다!
  - ✓ 물론! yarn add global nodemon 이나 npm install -g nodemon 이라는 명령어로 Local에서 yarn 이나 npm으로 init한 모든 프로젝트에서 사용할 수 있게 할 수도 있습니다.

- 또한, yarn에서 script를 사용하는 방법도 약간 알아보았습니다. 스크립트의 사용을 통해 개발 생산성을 좀 더 높이고 앞의 nodemon과 같은 사례도 해결할 수 있습니다!

실습 2 – Custom Module 만들기

- 지금까지 실습 1에서 yarn과 Module을 사용해보았으니 앞에서 예고한대로! 우리만의 Custom Module을 제작해봅시다.
- 막 그렇게 생각만큼 어려운 사항은 아니니 걱정하시지 마시고~ 뒤에 실습만 따라오시면 Module 하나 뚝딱 만드실 수 있을 겁니다.
- 실습 2에서 우리가 제작할 Module은 바로 계산기 모듈입니다. 사칙연산에 관한 Module을 한 번 제작해봅시다!
- 3가지 형식으로 계산기 모듈을 만들어보겠습니다~

실습 2 – Custom Module 만들기

- 계산기 모듈을 3가지 형태로 만들어봅시다! (더 많은 형태가 있을 수도 있겠죠?)

```
const calc = {};

calc.add = function(a,b) {
    return a + b;
}

calc.sub = function(a,b) {
    return a - b;
}

calc.mul = function(a,b) {
    return a * b;
}

calc.div = function(a,b) {
    if(b == 0) {
        return new Error('zero divider error.');
    }
    return a / b;
}

calc.getE = function() {
    return Math.E;
}

module.exports = calc;
```

```
function add(a,b) {
    return a + b;
}

function sub(a,b) {
    return a - b;
}

function mul(a,b) {
    return a * b;
}

function div(a,b) {
    if(b == 0) {
        return new Error('zero divider error.');
    }
    return a / b;
}

function getE() {
    return Math.E;
}

module.exports = {
    add: add,
    sub: sub,
    mul: mul,
    div: div,
    getE: getE
}
```

```
exports.add = function(a,b) {
    return a + b;
}

exports.sub = function(a,b) {
    return a - b;
}

exports.mul = function(a,b) {
    return a * b;
}

exports.div = function(a,b) {
    if(b == 0) {
        return new Error('zero divider error.');
    }
    return a / b;
}

exports.getE = function() {
    return Math.E;
}
```

실습 2 – Custom Module 만들기

- 만들었으니 사용해봐야죠? 앞에서 만든 3가지 형태의 코드를 모두 실행해봅시다!
- 오른쪽 예제 코드를 참고해서 여러분의 코드에 맞게 고쳐 사용해보세요~
- 중요하게 볼 점 중 하나는! 앞에서 다른 형태로 구현한 모듈들이 모두 올바르게 기능하는가 입니다~
- 앞에서 설치한 nodemon을 이용해서 실행시키면 코드를 바꿔도 보기 간편하겠죠?

```
const http = require('http');
const cal1 = require('./cal1');
const cal2 = require('./cal2');
const cal3 = require('./cal3');

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end(`1 + 1 = ${cal1.add(1,1)}\n` +
          `2 - 10 = ${cal2.sub(2,10)}\n` +
          `3 * 7 = ${cal3.mul(3,7)}`);
});

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

실습 2 – Custom Module 만들기

- 3개의 파일이 모두 같은 동작을 수행한다는 것을 알 수 있었을 것입니다. 그런데 코드를 살펴보면 어떤 파일은 exports로 어떤 파일은 module.exports인데 이것의 차이는 있을까요?

- Node.js에서 Module이 return 되는 형식은 module.exports이고 exports는 이를 Call by Reference로 가리키고 있다. 따라서 모두 같은 하나의 객체를 바라보고 있으므로 편하신 대로 사용하세요~

- 실제 Node.js의 문서에도 exports 설명은 아래와 같습니다.
  - ✓ A reference to the module.exports that is shorter to type. See the section about the exports shortcut for details on when to use exports and when to use module.exports.

- 이렇게 또 얼렁뚱땅 Custom Module을 하나 만들어보았습니다. Javscript 생태계에 yarn이나 npm을 통해 업로드할 수 있지만 이는 준스에선 다루지 않을 예정입니다. (궁금하시면 각자 찾아보세요!)

실습 3 – GitHub 맛보기

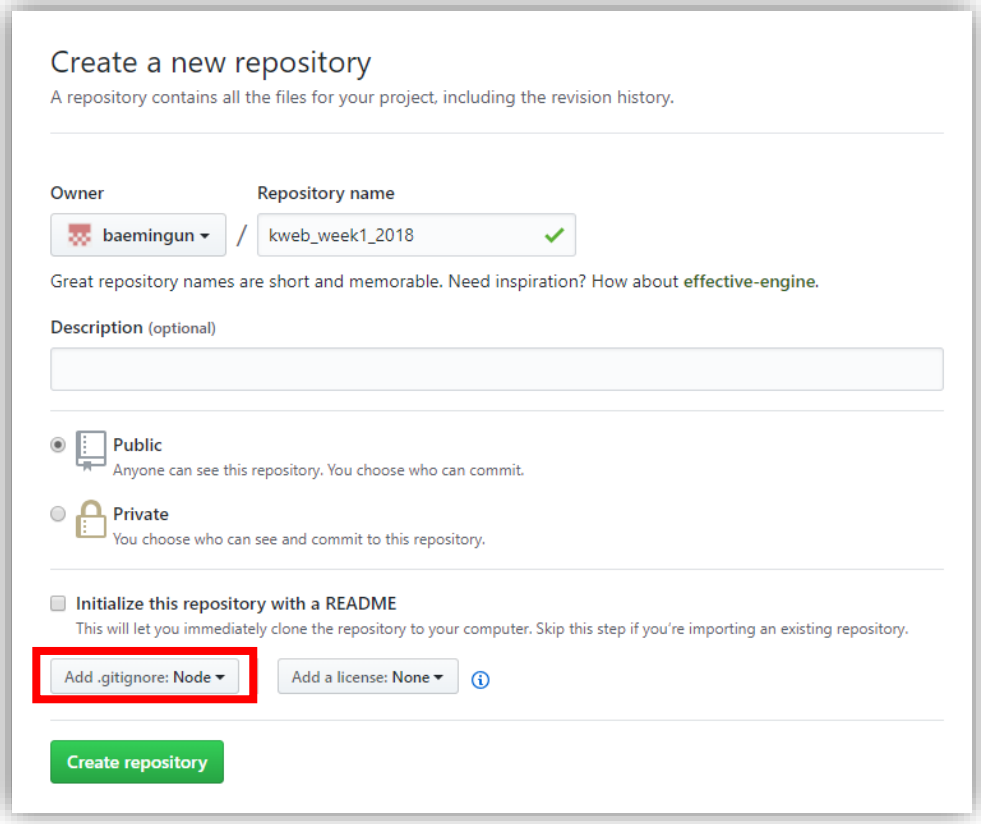
- 오늘 주제인 Module, Yarn, Git 중 Module과 yarn에 대해 실습해보았습니다! 마지막 실습은 Git을 이용하여 Git 저장소 중 가장 유명한 GitHub에 지금까지 실습한 코드를 업로드하는 것입니다.

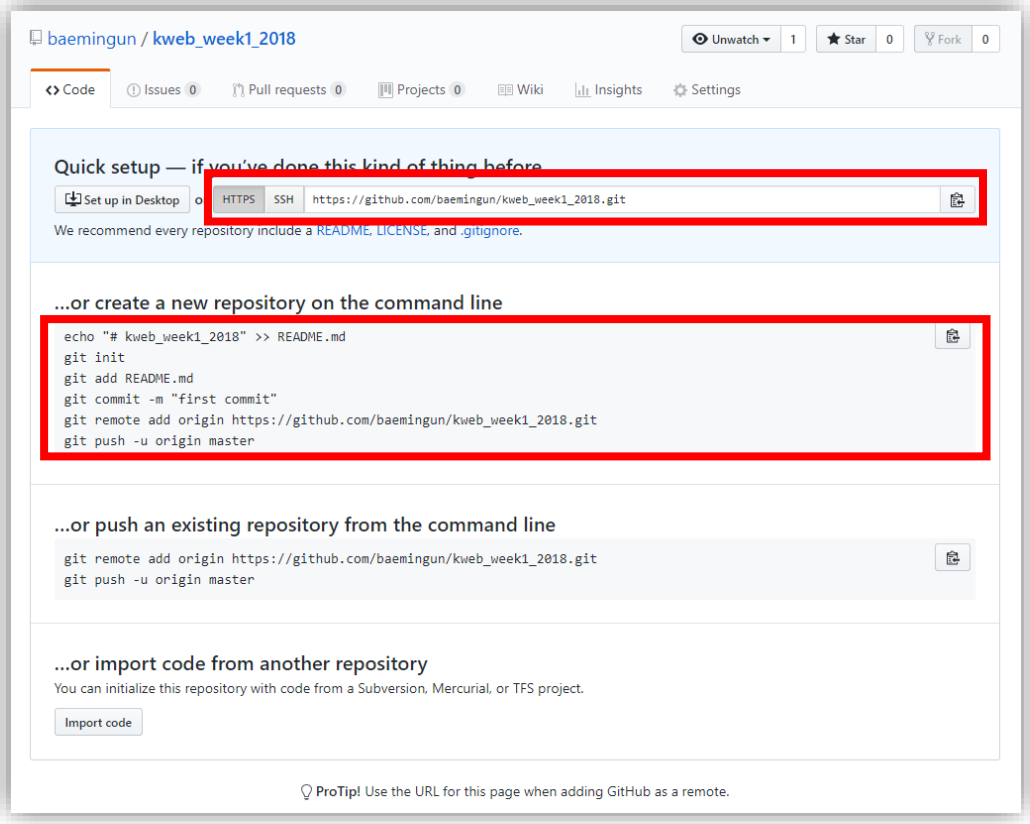
- 먼저, GitHub에 계정이 없으신 분들은 가입해주세요! Facebook 보다 가입 방법이 간단하니 각자 ㄱㄱ

- 아래의 과정을 따라 실습이 진행될 예정입니다.
  - ✓ GitHub에 원격 저장소 생성
  - ✓ 실습 폴더를 Git을 이용해 Local 저장소로 지정
  - ✓ 실습 폴더의 파일들을 Git을 이용해 원격 저장소(GitHub)로 업로드

실습 3 – GitHub 맛보기

- Repository를 생성해봅시다. New를 눌러 생성하고 완료하세요! (오른쪽은 .gitignore을 추가하지 않으시면 나타나는 화면입니다.)

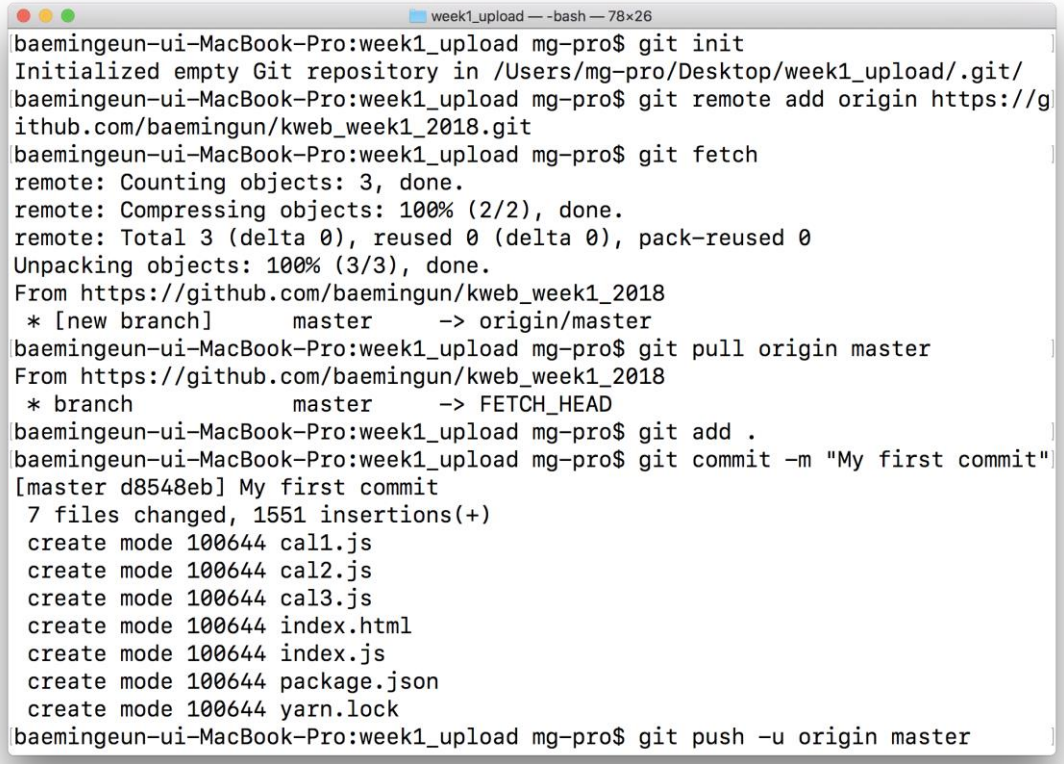




실습 3 – GitHub 맛보기

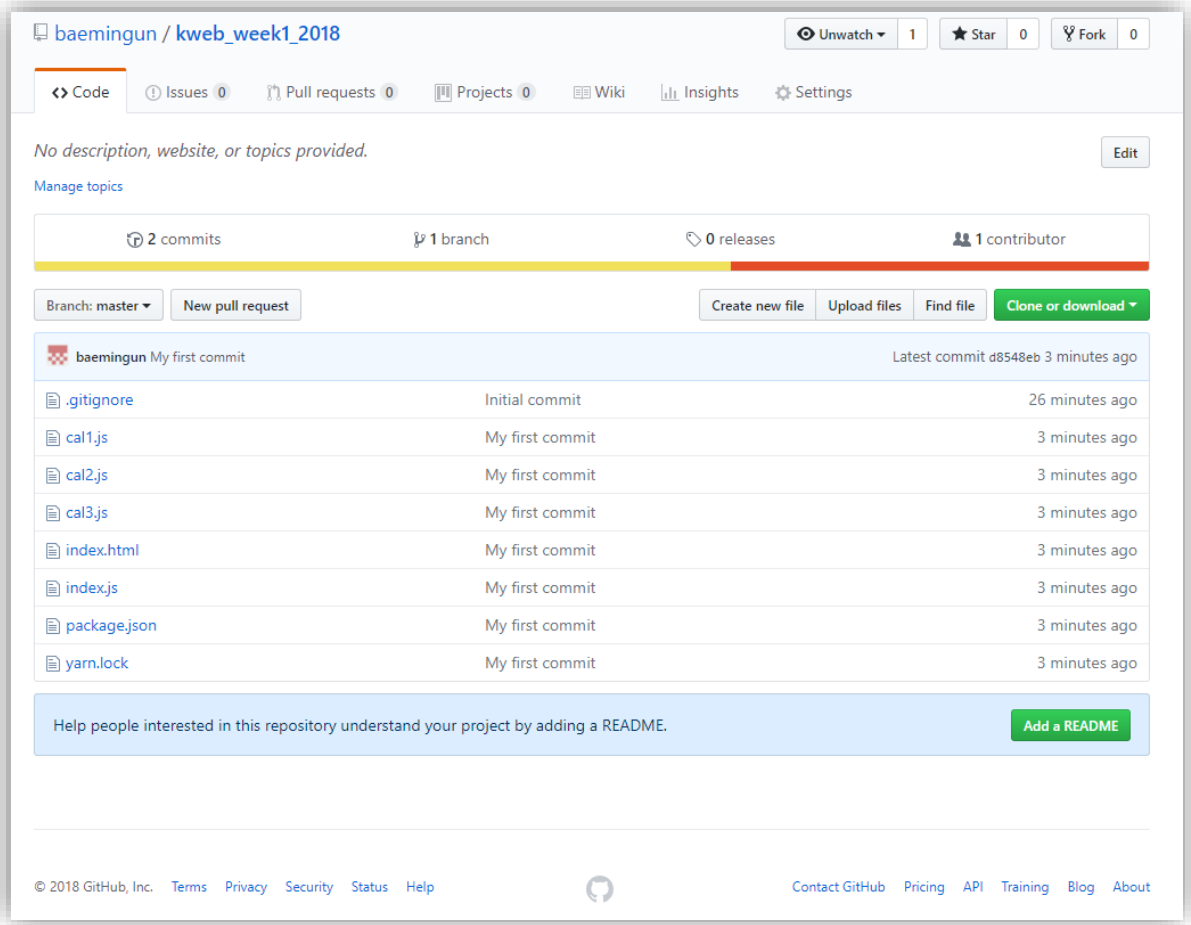
- GitHub의 안내에 따라 다음 명령어들을 실습 1,2 파일들이 위치한 곳에서 실행시켜봅시다.
  - ✓ git init
  - ✓ git remote add origin {복사한 git 저장소 주소}
  - ✓ git fetch
  - ✓ git pull origin master
  - ✓ git add . → git add 다음에 . 쓰면 아래 모든 파일을 포함하겠다는 의미입니다.
  - ✓ git commit –m “My first commit”
  - ✓ git push –u origin master

- 참고! git add . 을 생략하고 git commit –a –m “commit 메시지” 도 가능합니다.



실습 3 – GitHub 맛보기

- 여러분의 GitHub에서 결과를 확인해주세요!
- 솔직히 아직 뭐가 뭔지 모르실겁니다. Git을 처음 사용해보는 거니까요! 차차 익숙해지시면 됩니다.
- 과제 1이 Git에 관한거니 그걸 하시면서 좀 더 알아가시길 바래요! 실습 3은 그냥 한 번 해보는게 목적이었습니다 ㅎ
- (어느 정도 뒤에는 GUI로 간편하게 써봅시다!)



과제 (~ 9/30 23:59:59)

- 과제기한: 2018년 9월 30일 일요일 자정까지

- 추석 연휴도 있고 기한이 제법 길죠? 그러니까 이번주 과제는 2개입니다. (파이팅!) 자세한 과제 명세는 이어지는 슬라이드를 참고해주세요!
  - ✓ **과제 1 – Git 입문 공부하기**: 명세에 있는 동영상 강좌를 완료하기
  - ✓ **과제 2 – Module 기능 추가하기**: 실습 1,2에서 한 Module에 기능을 추가해 GitHub에 업로드하기

- 제출방법: 과제 1을 완료했다는 캡쳐와 과제 2에서 추가 구현한 파일을 여러분의 GitHub Repository에 반영하고 그 Repository 주소를 준회원 톡방에 올리시면 됩니다.

- 참고로, 과제 2는 무슨 말인지 이해가 안가시면 과제 1을 똑바로 하시고 나면 이해될 것입니다! (ㅋㅋㅋ)

과제 명세

- **과제 1** - 아래 URL 코스를 완료해오시면 됩니다. 완료하면 그 화면을 캡처해서 파일로 저장해주세요!
  - ✓ https://www.inflearn.com/course/github-기본기-10분/

- **과제 2** - 실습 1,2에서 다룬 계산기 모듈에 % 연산을 추가하시면 됩니다.
  - ✓ 실습에서 계산기 모듈을 3가지 형태로 만든 만큼 3개 파일 모두에 % 연산을 각 파일의 코딩 스타일에 맞춰 추가하시면 됩니다.

- 최종적인 제출은 과제 1에서 찍은 캡처와 과제 2에서 만든 Module의 추가 기능을 실습 3에서 사용한 Repository에 추가하고 준회원 톡방에 여러분의 Repository 주소를 올리시면 됩니다.
  - ✓ **과제 2는 Repository에 새로운 파일이 추가되면 안되고 꼭 기존에 실습 3에서 업로드된 파일이 수정되는 형태여야만 합니다.**

That’s all for today!