Open-Source Software Practice

7. Testing and Publishing

Instructor: Jaemin Jo (조재민, jmjo@skku.edu)
Interactive Data Computing Lab (*IDCLab*),
College of Computing and Informatics,
Sungkyunkwan University

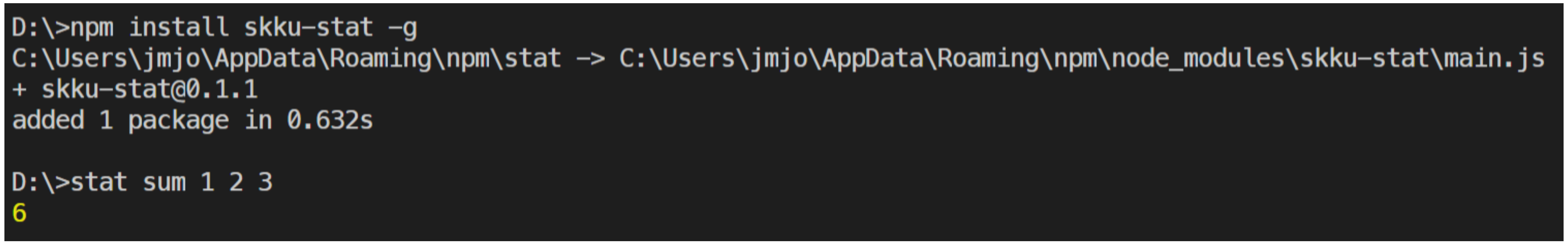
Review: Javascript Advanced

- You can reuse Javascript packages to speed up your development.
- Using external packages introduces a lot of versioning and dependency issues, but a package manager will resolve them.
    - Node Package Manager or NPM
- *package.json* contains the metadata of your project. You must include this file in your project.

- `npm install <name>`
- `npm install --save-dev <name>`
- `npm install <name> -g`

What do we do?

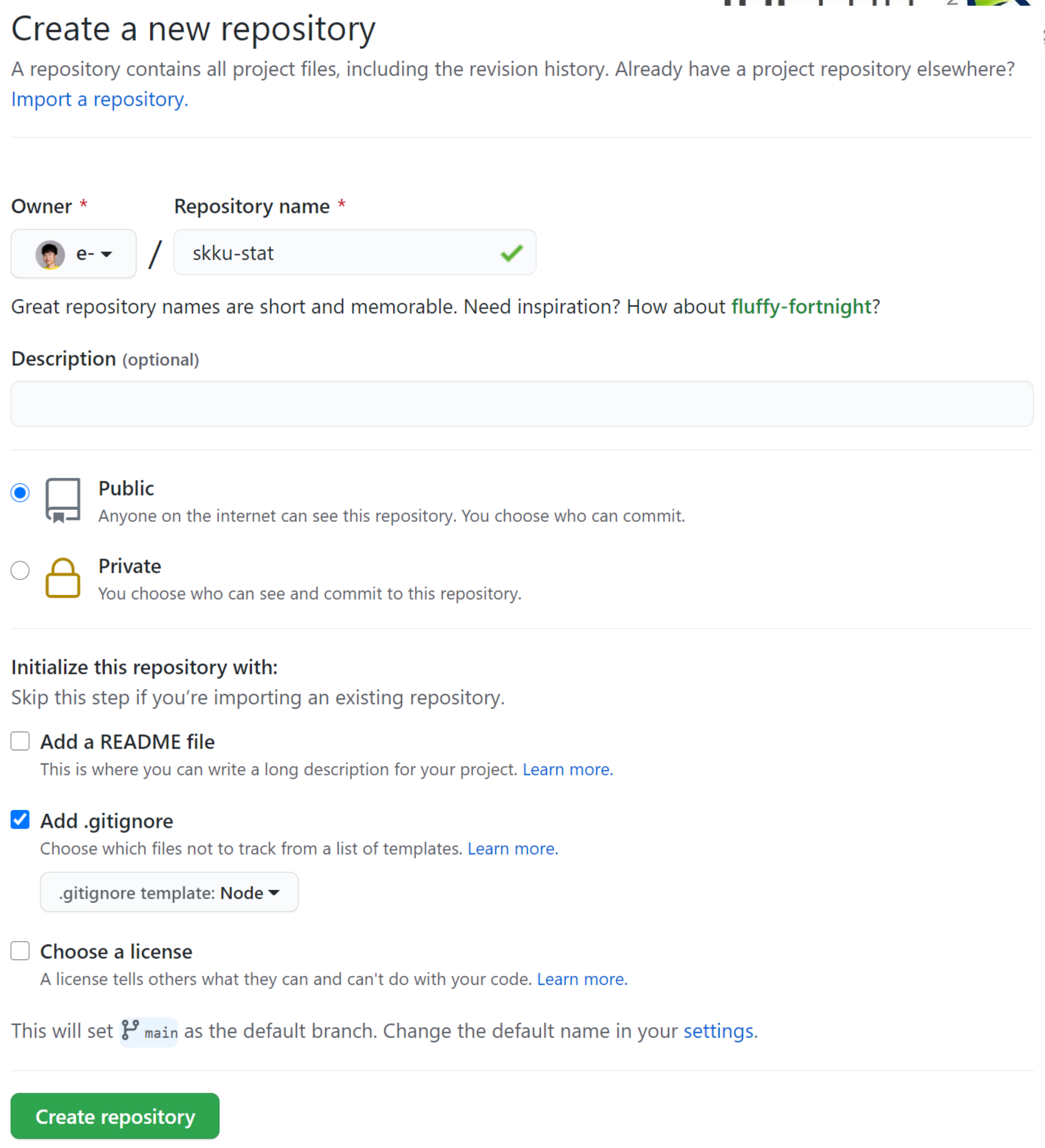
- Let’s write a command-line tool, *stat*, and publish it to NPM.
- Note that below we are installing it from NPM not a local package!
  - `stat sum 1 2 3`
  - `stat avg 1 2 3`
  - `stat max 1 2 3`



```
D:\>npm install skku-stat -g
C:\Users\jmjo\AppData\Roaming\npm\stat -> C:\Users\jmjo\AppData\Roaming\npm\node_modules\skku-stat\main.js
+ skku-stat@0.1.1
added 1 package in 0.632s

D:\>stat sum 1 2 3
6
```

Setup

- Create a GitHub repository
- Don't forget to add the default .gitignore file for Node
- Use a different name for your package (*skku-stat*).
  - Or, there will be a conflict when you publish it since I am using the same name.



Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.

Owner * e- / Repository name * skku-stat ✓

Great repository names are short and memorable. Need inspiration? How about fluffy-fortnight?

Description (optional)

● Public
Anyone on the internet can see this repository. You choose who can commit.

○ Private
You choose who can see and commit to this repository.

Initialize this repository with:
Skip this step if you're importing an existing repository.

☐ Add a README file
This is where you can write a long description for your project. Learn more.

☑ Add .gitignore
Choose which files not to track from a list of templates. Learn more.

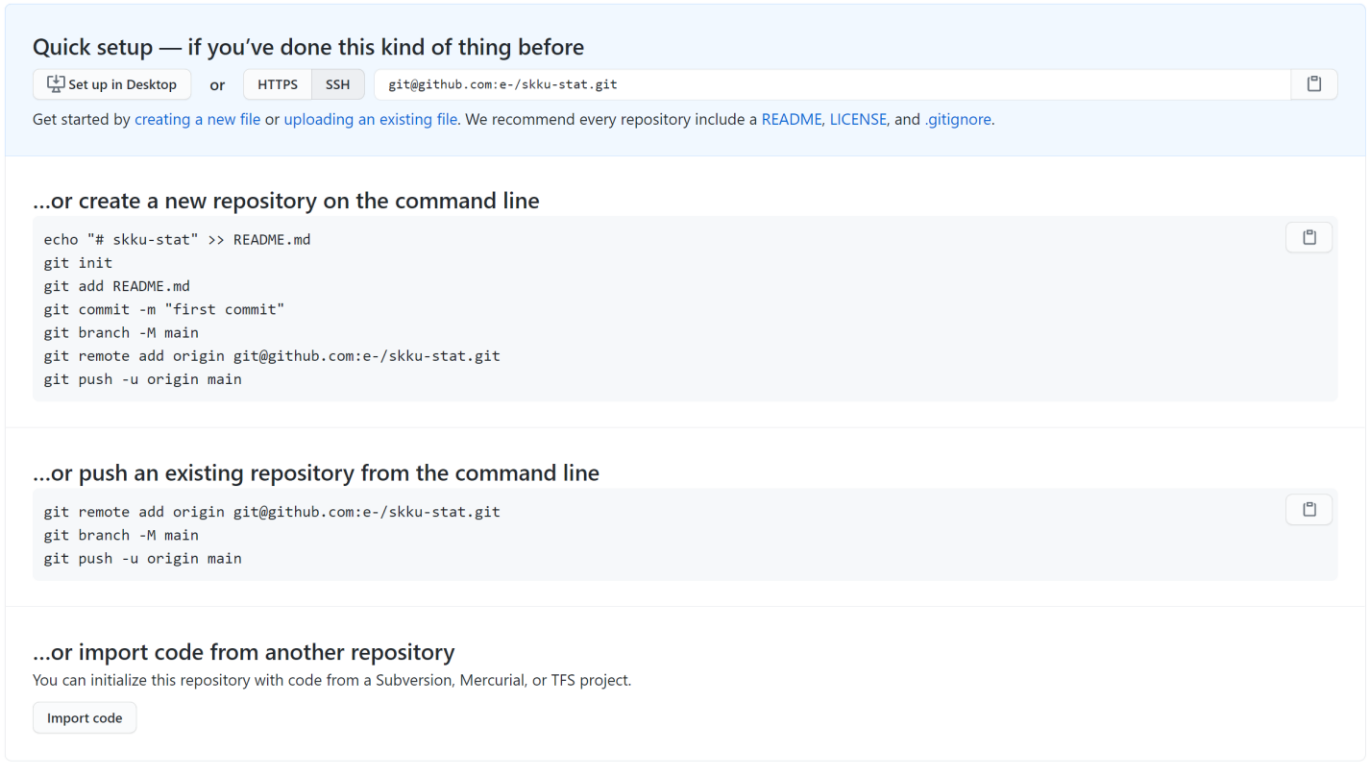
.gitignore template: Node

☐ Choose a license
A license tells others what they can and can't do with your code. Learn more.

This will set `main` as the default branch. Change the default name in your settings.

Setup

- Clone the repository.
  - There will be a default .gitignore file and this file may cause merge conflicts.
  - Consider `git pull origin main --allow-unrelated-histories`



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

git@github.com:e-/skku-stat.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 "# skku-stat" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin git@github.com:e-/skku-stat.git
git push -u origin main
```

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

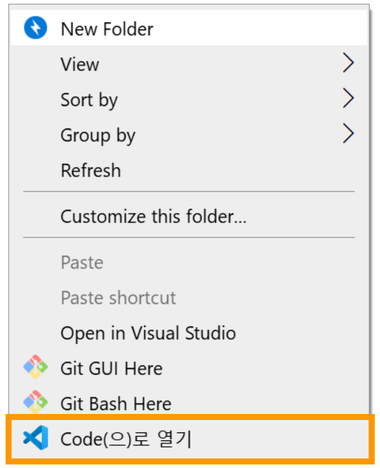
```
git remote add origin git@github.com:e-/skku-stat.git
git branch -M main
git push -u origin main
```

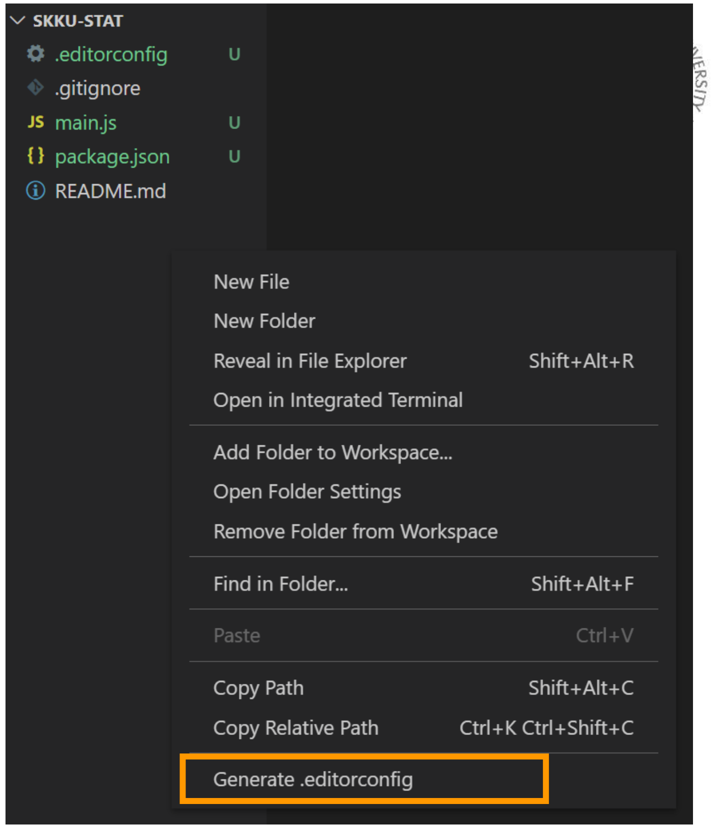
...or import code from another repository

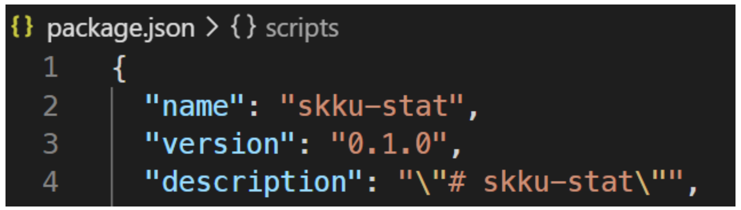
You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

Setup

- Open the cloned project on VSCode.
- Ctrl + Shift + ` to open the terminal
- On the terminal, `npm init`
  - Use the default setting
- Generate .editorconfig
  - Install the plugin if you haven’t installed it.
  - Use the default setting.
- Modify the version to 0.1.0 from 1.0.0 in *package.json*




SKKU-STAT
.editorconfig U
.gitignore
main.js U
package.json U
README.md
New File
New Folder
Reveal in File Explorer Shift+Alt+R
Open in Integrated Terminal
Add Folder to Workspace...
Open Folder Settings
Remove Folder from Workspace
Find in Folder... Shift+Alt+F
Paste Ctrl+V
Copy Path Shift+Alt+C
Copy Relative Path Ctrl+K Ctrl+Shift+C
Generate .editorconfig



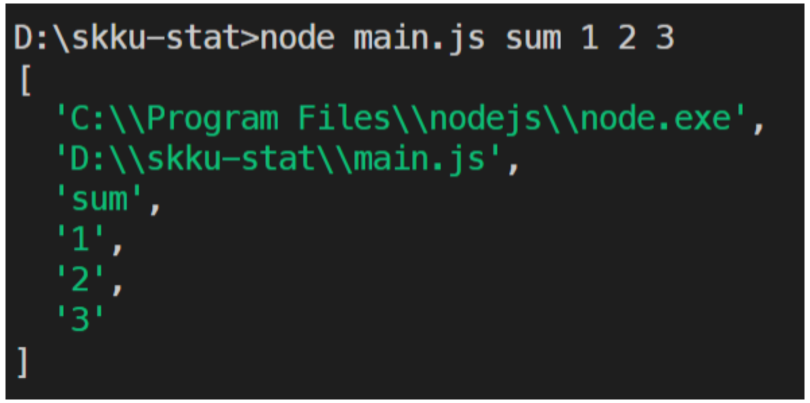
```
{
  "name": "skku-stat",
  "version": "0.1.0",
  "description": "\"# skku-stat\"",
```

Command-Line Arguments

- Command you will run: `stat sum 1 2 3`
- Result: 6

- We need to read the command-line arguments after the command name.
- When the program starts, they are stored in an array *process.argv*.

```
console.log(process.argv);
```

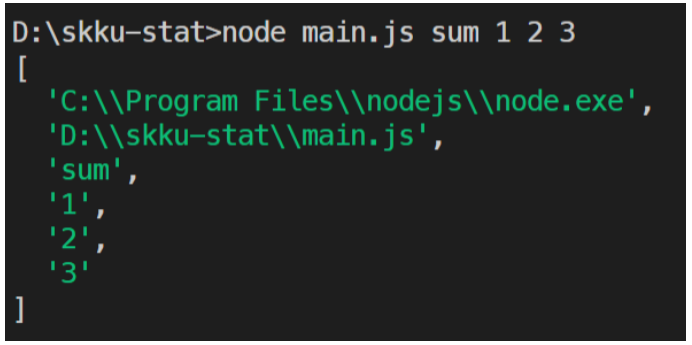


```
D:\skku-stat>node main.js sum 1 2 3
[
  'C:\\Program Files\\nodejs\\node.exe',
  'D:\\skku-stat\\main.js',
  'sum',
  '1',
  '2',
  '3'
]
```

Command-Line Arguments

- *process.argv[0]* is the path to the Node runtime.
- *process.argv[1]* is the path to the current file.
- *process.argv[2]* is the operation type.
- From *process.argv[3]*, there are numbers, but they are given as strings.

- There are (*process.argv.length* – 3) numbers in total.



```
D:\skku-stat>node main.js sum 1 2 3
[
  'C:\\Program Files\\nodejs\\node.exe',
  'D:\\skku-stat\\main.js',
  'sum',
  '1',
  '2',
  '3'
]
```

Reading Command-Line Arguments

- Let's read the arguments and store them in variables.
- *array.slice(a, b)* creates a new array using the elements selected from *a* to *b*.
  - *array.slice(3, array.length)* will select the elements from index 3 to the end.

- *array.map(f)* collects the return values of a function *f* after applying it to each element.
- *parseFloat(n)* converts a string to a number.

```
let command = process.argv[2];

let numbers = process.argv
    .slice(3, process.argv.length)
    .map((n) => parseFloat(n));
```

Switch-Case

- We will use a switch-case statement to process the command type.
  - Store the result in the *result* variable, which will be logged at the end.
- If *command* is not “sum”, “avg”, nor “max”, we will print out an error message.
- *process.exit(1)* exits the program with a return value 1 (error code).

```
let command = process.argv[2];

let numbers = process.argv
    .slice(3, process.argv.length)
    .map((n) => parseFloat(n));

let result;
switch (command) {
    case "sum":
        break;
    case "avg":
        break;
    case "max":
        break;
    default:
        console.log("Wrong command!");
        process.exit(1);
}

console.log(result);
```

Separating the Logic

- We need functions that compute the sum, average, and max of numbers.
- Let’s define the functions in a different file, *lib.js*.
  - lib for library

```
function sum(numbers) {
  let s = 0;
  for (let i = 0; i < numbers.length; i++) s += numbers[i];
  return s;
}

function avg(numbers) {
  return sum(numbers) / numbers.length;
}

function max(numbers) {
  let m = numbers[0];
  for (let i = 1; i < numbers.length; i++) if (m < numbers[i]) m = numbers[i];
  return m;
}

exports.sum = sum;
exports.avg = avg;
exports.max = max;
```

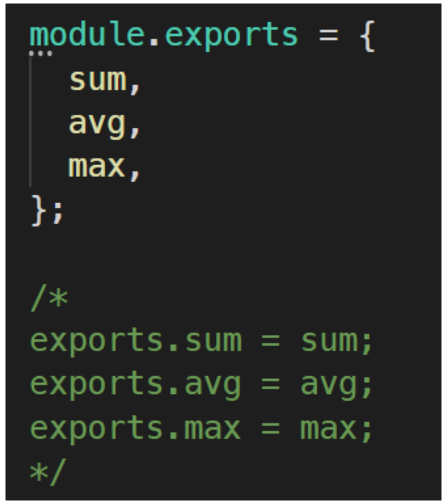
Separating the Logic

- You can make them more “functional”.

```
function sum(numbers) {
  return numbers.reduce((prev, curr) => prev + curr, 0);
}

function avg(numbers) {
  return sum(numbers) / numbers.length;
}

function max(numbers) {
  return numbers.reduce((max, curr) => (max > curr ? max : curr), numbers[0]);
}

exports.sum = sum;
exports.avg = avg;
exports.max = max;
```

Separating the Logic

- Exporting multiple functions at the same time.



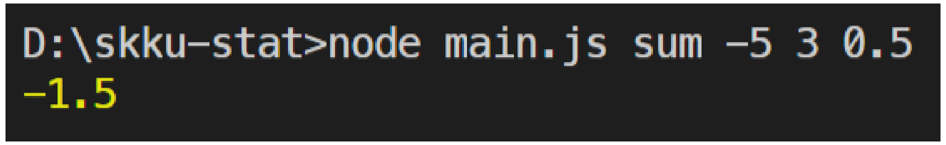
```
module.exports = {
  sum,
  avg,
  max,
};

/*
exports.sum = sum;
exports.avg = avg;
exports.max = max;
*/
```

lib.js

```javascript
function sum(numbers) {
  return numbers.reduce((prev, curr) => prev + curr, 0);
}

function avg(numbers) {
  return sum(numbers) / numbers.length;
}

function max(numbers) {
  return numbers.reduce((max, curr) => (max > curr ? max : curr), numbers[0]);
}

module.exports = {
  sum,
  avg,
  max,
};
```

main.js

- Let’s load lib.js and use the functions!



```
D:\skku-stat>node main.js sum -5 3 0.5
-1.5
```

- Is this all?

```
const lib = require("./lib");

let command = process.argv[2];

let numbers = process.argv
    .slice(3, process.argv.length)
    .map((n) => parseFloat(n));

let result;
switch (command) {
    case "sum":
        result = lib.sum(numbers);
        break;
    case "avg":
        result = lib.avg(numbers);
        break;
    case "max":
        result = lib.max(numbers);
        break;
    default:
        console.log("Wrong command!");
        process.exit(1);
}

console.log(result);
```

Error Handling

- Let's handle possible errors.

1) If *command* is not one of sum, avg, and max, print "Wrong command!"
   - This case is already handled in the previous code.

2) If an insufficient number of parameters is given, print "Insufficient parameter!"
   - There must be at least one command and one number.

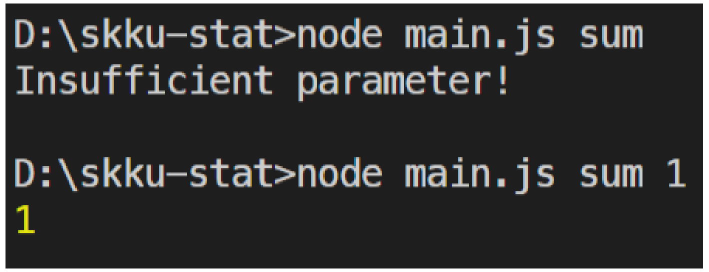
3) If there is a parameter that is not a number, print "Some arguments are not numbers!"

- In either case, exit the problem with an error code 1.

Error Handling

- Insufficient parameters

```javascript
const lib = require("./lib");

if (process.argv.length <= 3) {
    console.log("Insufficient parameter!");
    process.exit(1);
}

let command = process.argv[2];
```



```
D:\skku-stat>node main.js sum
Insufficient parameter!

D:\skku-stat>node main.js sum 1
1
```

Error Handling

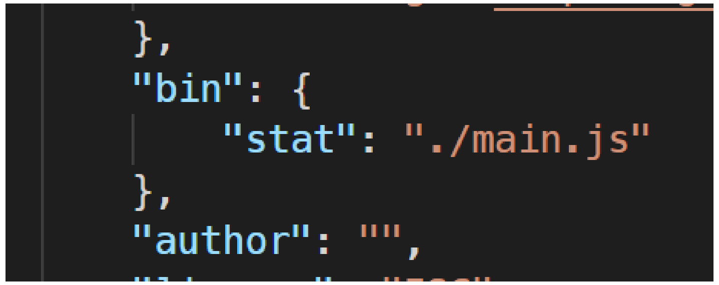
- If a non-number argument is given, *parseFloat* returns *NaN* (Not a Number).
- Function *isNaN(x)* checks whether *x* is *NaN* or not.
- *array.some(f)* returns *true* if *f* returns *true* for at least one element in *array.*
    - *cf.) array.every*

```
let numbers = process.argv
    .slice(3, process.argv.length)
    .map((n) => parseFloat(n));

if (numbers.some((n) => isNaN(n))) {
    console.log("Some arguments are not numbers!");
    process.exit(1);
}

let result;
```

Linking a Command

- Let’s make our script as a command.

1. Add *#!/usr/bin/env node* to the first line of *main.js*.
2. Link *main.js* to a command in *package.json*.



```
},
"bin": {
    "stat": "./main.js"
},
"author": "",
```

3. `npm link`
4. Then, you should be able to run the “stat” command on a terminal.

main.js

- Let’s test the command!

- `stat sum 1 2 -0.2`
- `stat avg -3 0 3`
- `stat max 2 5 3 0`

- Don’t forget to run `npm unlink` <name> after you test the command.

```javascript
const lib = require("./lib");

if (process.argv.length <= 3) {
    console.log("Insufficient parameter!");
    process.exit(1);
}

let command = process.argv[2];

let numbers = process.argv
    .slice(3, process.argv.length)
    .map((n) => parseFloat(n));

if (numbers.some((n) => isNaN(n))) {
    console.log("Some arguments are not numbers!");
    process.exit(1);
}

let result;
switch (command) {
    case "sum":
        result = lib.sum(numbers);
        break;
    case "avg":
        result = lib.avg(numbers);
        break;
    case "max":
        result = lib.max(numbers);
        break;
    default:
        console.log("Wrong command!");
        process.exit(1);
}

console.log(result);
```

Testing

- We manually tested the program by giving some small inputs.

- But what if the program gets bigger?
  - More components
  - Complex test scenarios
  - Heterogenous environments (e.g., Node.js versions)
  - Different developers

- We need to automate the test.

Unit Testing

- **Unit testing** is a type of software testing where individual units or components of a software are tested.

- We have two files: lib.js and main.js.

- We will write tests for each file.

Unit Testing in Node

- There are a lot of frameworks that you can use to do unit testing in Node.
  - Jest, mocha, chai, jasmine, …
  - We will use Jest.
  - https://jestjs.io/

- `npm install jest --save-dev`
  - Note that we set the --save-dev flag for installation.
  - Jest is **not** required to run our program (ours does not have external dependencies!) but is needed to test our program as part of development.

--save-dev

	npm install <name>	npm install <name> --save-dev
Meaning	Required package	Only required for development
Where the dependency is listed	"dependencies" in *package.json*	"devDependencies" in *package.json*
Who concerns?	All users (end-user + developers)	Only developers want to extend your package

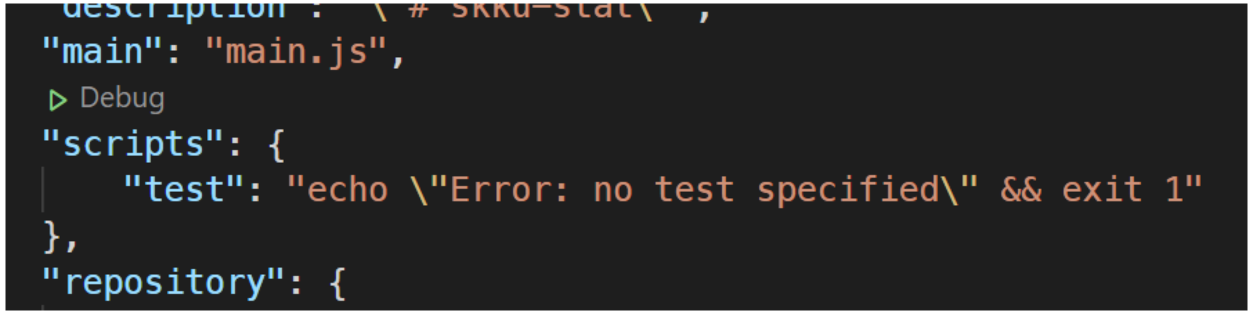
Writing Tests

- Let’s test *lib.js* first.
- Conventionally, tests for *<file_name>.js* go to *<file_name>.test.js*.
  - *lib.test.js*
- In *lib.test.js*,
  - Load the file that is tested (*lib.js*).
  - Run some functions with fixed input.
  - Check if the result is the same as the expected output.

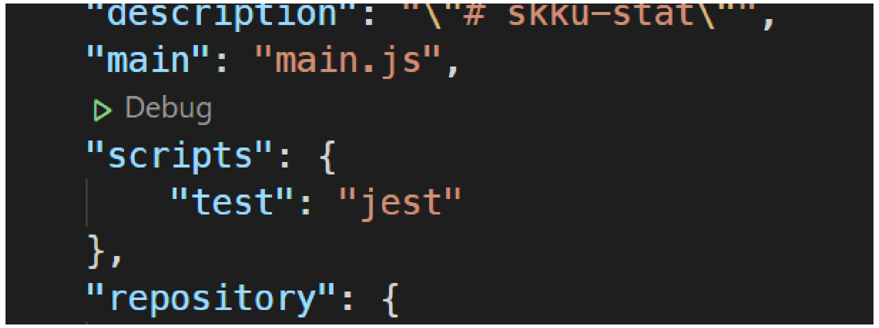
```
const { test, expect } = require("@jest/gl
obals");
const lib = require("./lib");

test("sum([1, 2]) should be 3", () => {
    expect(lib.sum([1, 2])).toBe(3);
});

test("avg([-5, 5]) should be 0", () => {
    expect(lib.avg([-5, 5])).toBe(0);
});

test("max([0, 3, 2]) should be 3", () => {
    expect(lib.max([0, 3, 2])).toBe(3);
});
```

Running Tests

- After you added a test, run it via `npm test`.
- But you will see an error since Jest is not linked to the command.
- Open *package.json* and set “test” under “scripts” to “jest”.



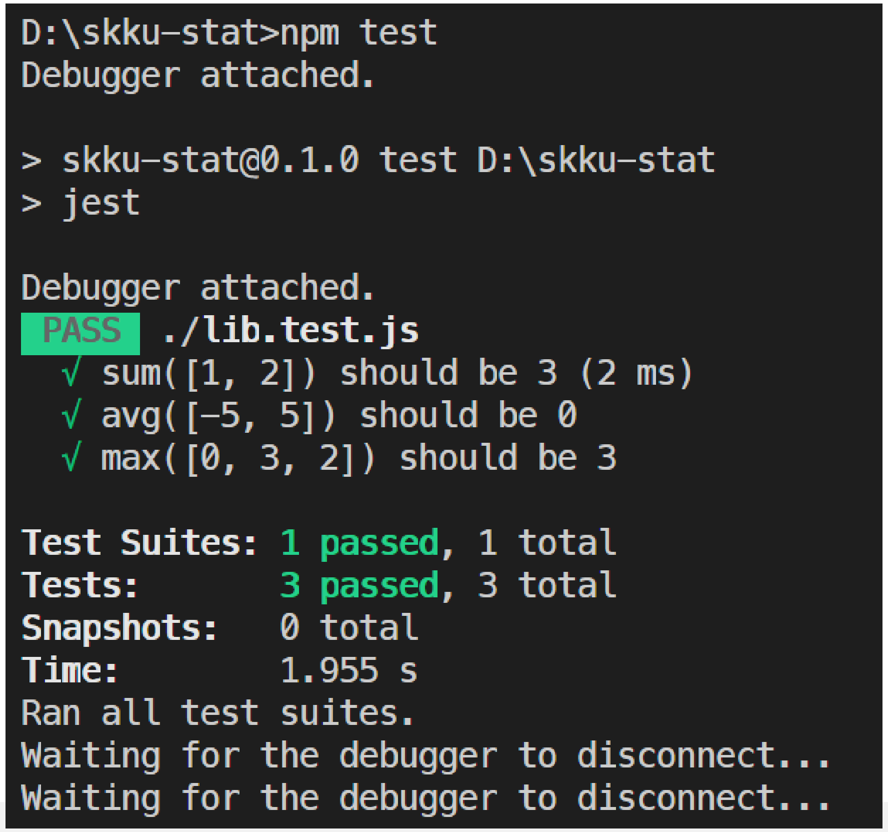
```
"description": "\"# skku-stat\"",
"main": "main.js",
▷ Debug
"scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
},
"repository": {
```



```
"description": "\"# skku-stat\"",
"main": "main.js",
▷ Debug
"scripts": {
    "test": "jest"
},
"repository": {
```

Running Tests

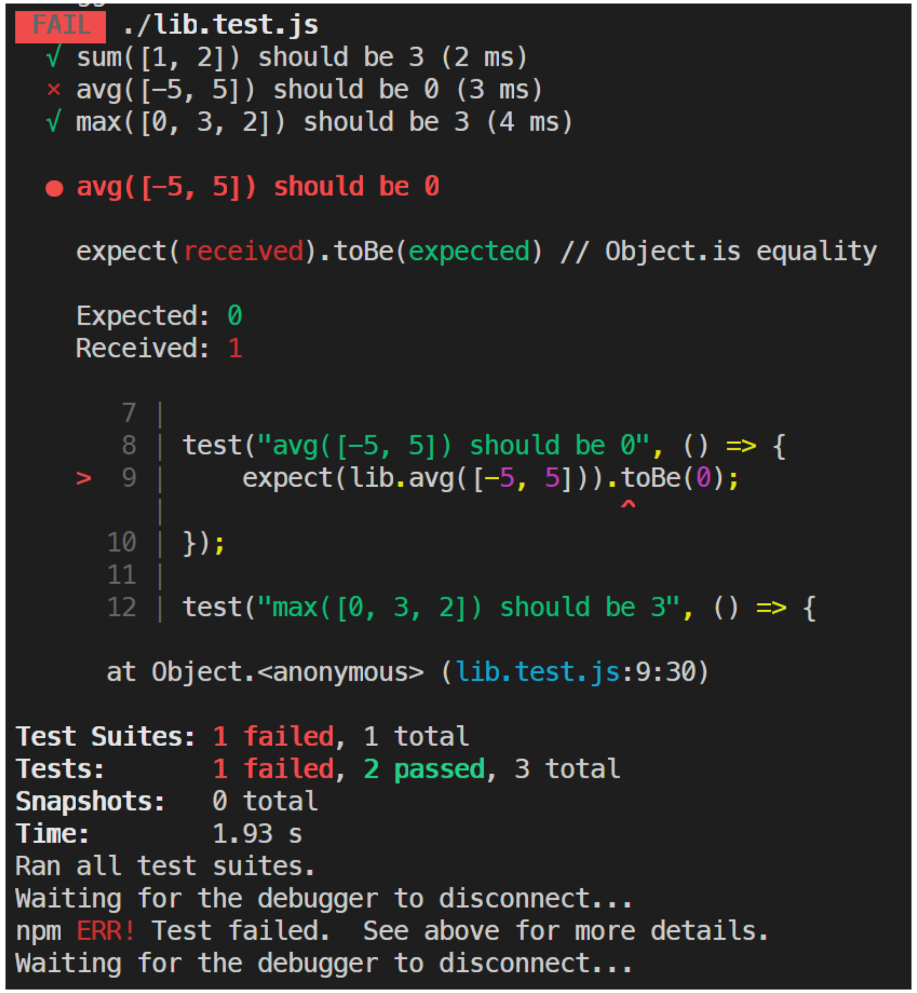
- `npm test`
- Congrats! You just finished your first unit testing.



```
D:\skku-stat>npm test
Debugger attached.

> skku-stat@0.1.0 test D:\skku-stat
> jest

Debugger attached.
 PASS  ./lib.test.js
  √ sum([1, 2]) should be 3 (2 ms)
  √ avg([-5, 5]) should be 0
  √ max([0, 3, 2]) should be 3

Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
Snapshots:   0 total
Time:        1.955 s
Ran all test suites.
Waiting for the debugger to disconnect...
Waiting for the debugger to disconnect...
```

Running Tests

- If your test fails, you will see:



```
 FAIL  ./lib.test.js
  √ sum([1, 2]) should be 3 (2 ms)
  × avg([-5, 5]) should be 0 (3 ms)
  √ max([0, 3, 2]) should be 3 (4 ms)

  ● avg([-5, 5]) should be 0

    expect(received).toBe(expected) // Object.is equality

    Expected: 0
    Received: 1

       7 |
       8 | test("avg([-5, 5]) should be 0", () => {
    >  9 |     expect(lib.avg([-5, 5])).toBe(0);
         |                              ^
      10 | });
      11 |
      12 | test("max([0, 3, 2]) should be 3", () => {

      at Object.<anonymous> (lib.test.js:9:30)

Test Suites: 1 failed, 1 total
Tests:       1 failed, 2 passed, 3 total
Snapshots:   0 total
Time:        1.93 s
Ran all test suites.
Waiting for the debugger to disconnect...
npm ERR! Test failed.  See above for more details.
Waiting for the debugger to disconnect...
```

Writing Tests

- Let’s test *main.js*.
- I want to test whether the error messages are correctly printed if I gave wrong input.

- This is different from the previous case.
  - In *lib.test.js*, we could run a function and check the returned value is correct.
  - But now, we should **run the script** and **check the actual output**.

Writing Tests

- It's fine if you don't fully understand the code.

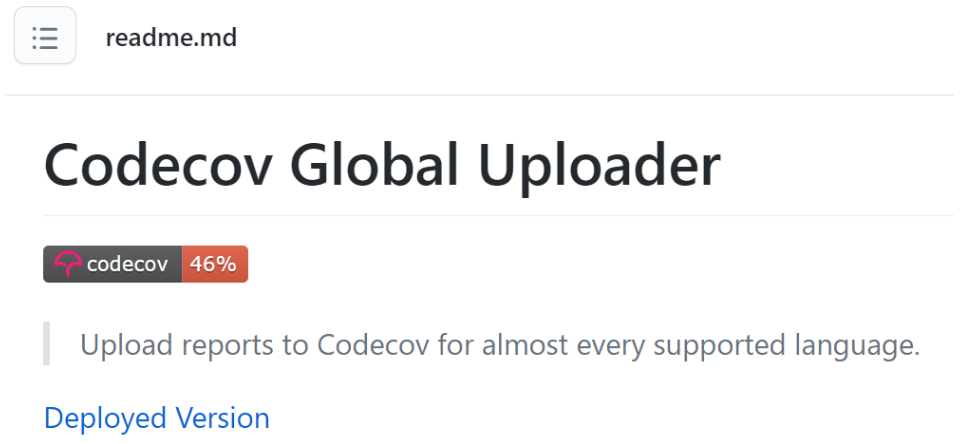
- Just understand the big picture.
    1. *spawn* to run a subprocess with the given command.
    2. store the printed strings to an array.
    3. concatenate the strings and compare it with the expected output.

```
const { test, expect } = require("@jest/globals");
const { spawn } = require("child_process");

test("Insufficient params", () => {
    const main = spawn("node", ["main.js", "avg"]);
    const outputs = [];
    main.stdout.on("data", (output) => {
        outputs.push(output);
    });

    main.stdout.on("end", () => {
        const output = outputs.join("").trim();
        expect(output).toBe("Insufficient parameter!");
    });
});

test("Wrong command", () => {
    const main = spawn("node", ["main.js", "count", "0"]);
    const outputs = [];
    main.stdout.on("data", (output) => {
        outputs.push(output);
    });

    main.stdout.on("end", () => {
        const output = outputs.join("").trim();
        expect(output).toBe("Wrong command!");
    });
});
```

Code Coverage

- Ideally, each line of code should be executed at least once during testing.

- **Code coverage**: # of lines of code executed by tests / total # of lines

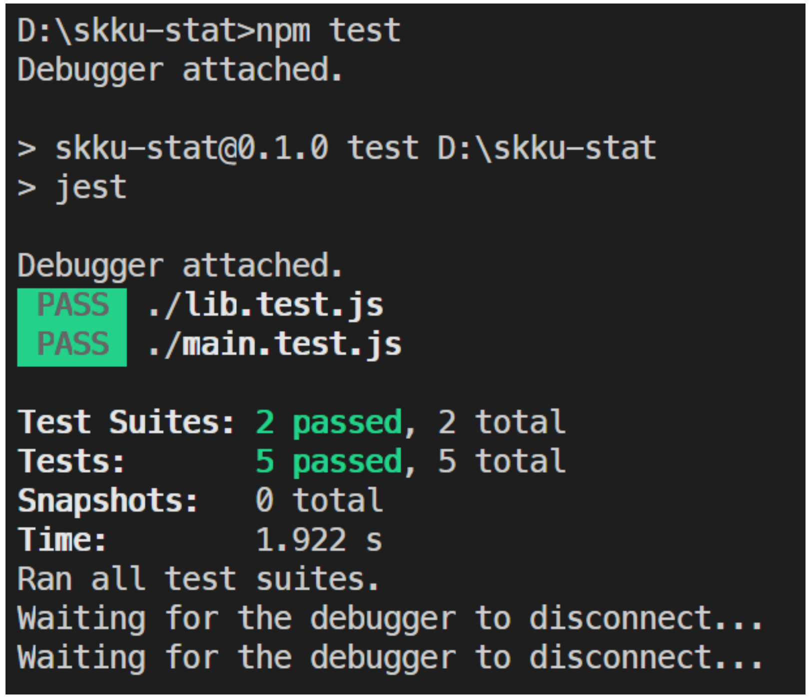


readme.md

Codecov Global Uploader

Upload reports to Codecov for almost every supported language.

Deployed Version



```
D:\skku-stat>npm test
Debugger attached.

> skku-stat@0.1.0 test D:\skku-stat
> jest

Debugger attached.
 PASS  ./lib.test.js
 PASS  ./main.test.js

Test Suites: 2 passed, 2 total
Tests:       5 passed, 5 total
Snapshots:   0 total
Time:        1.922 s
Ran all test suites.
Waiting for the debugger to disconnect...
Waiting for the debugger to disconnect...
```

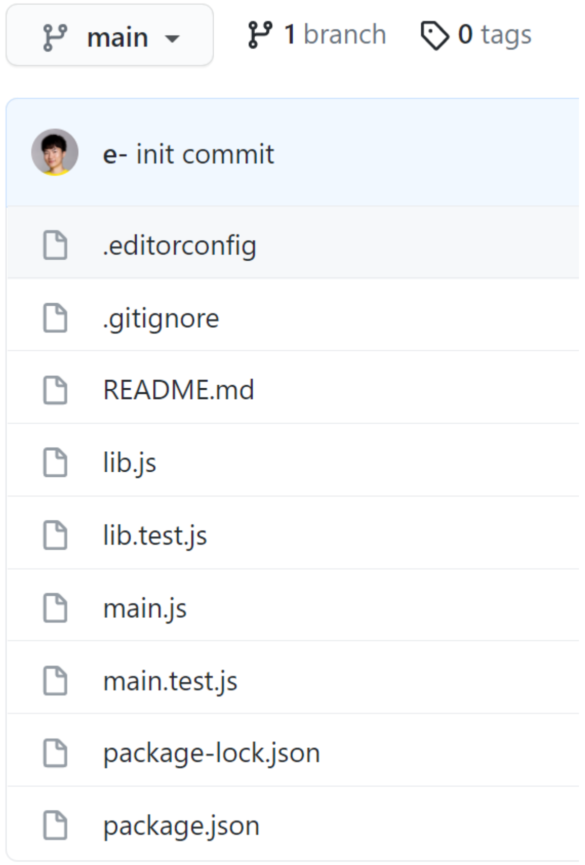
Git Actions

- Let's push the code to Github.

- It would be awesome if unit tests are automatically run every time we update the code.

- This can be automated by GitHub Actions.

- Go to the Action tab in your repo.



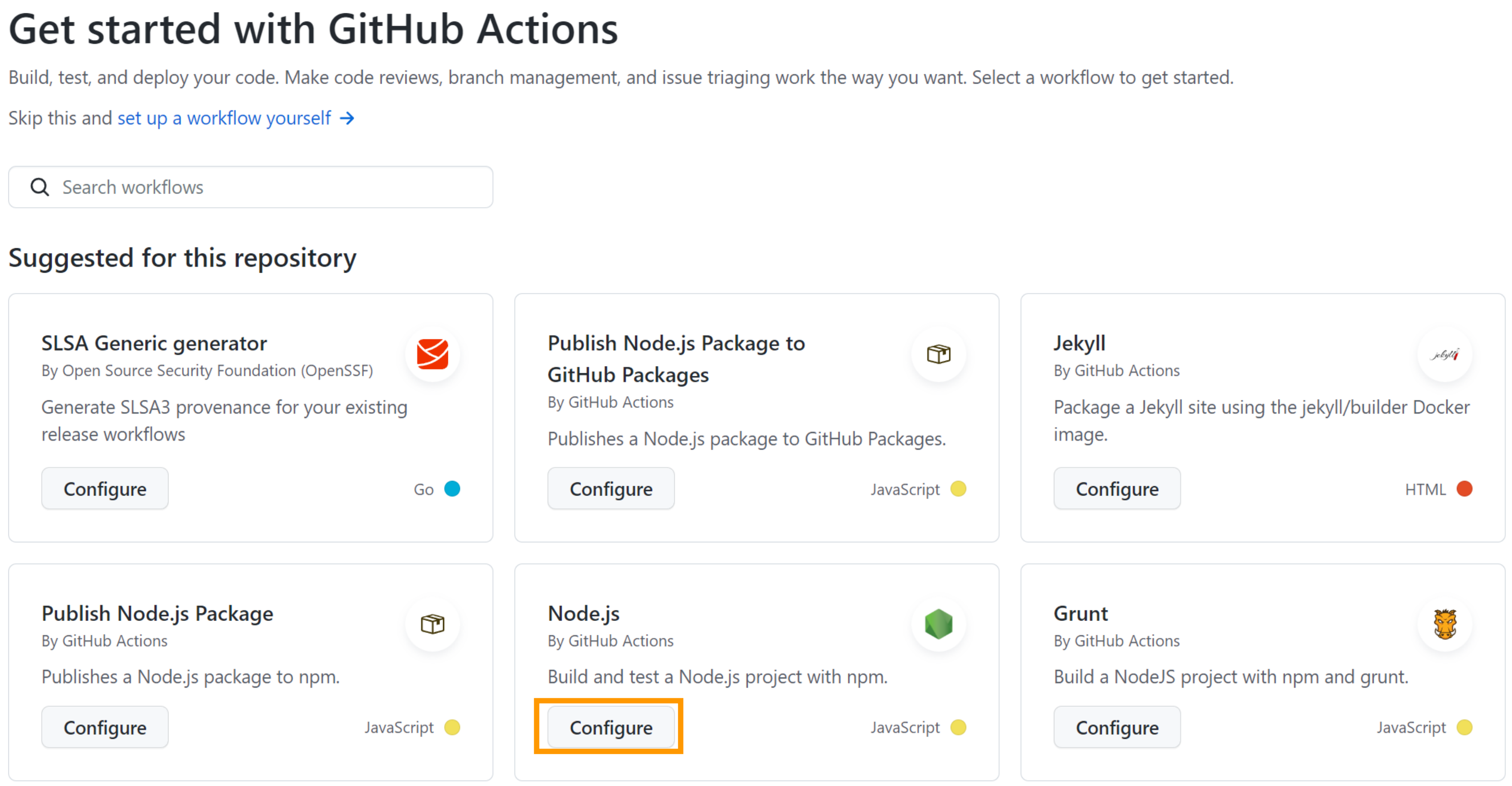
main · 1 branch · 0 tags

e- init commit

.editorconfig
.gitignore
README.md
lib.js
lib.test.js
main.js
main.test.js
package-lock.json
package.json

Git Actions

- Click on “Set up this workflow” under Node.js



Get started with GitHub Actions

Build, test, and deploy your code. Make code reviews, branch management, and issue triaging work the way you want. Select a workflow to get started.

Skip this and set up a workflow yourself →

Search workflows

Suggested for this repository

SLSA Generic generator
By Open Source Security Foundation (OpenSSF)
Generate SLSA3 provenance for your existing release workflows
Configure — Go

Publish Node.js Package to GitHub Packages
By GitHub Actions
Publishes a Node.js package to GitHub Packages.
Configure — JavaScript

Jekyll
By GitHub Actions
Package a Jekyll site using the jekyll/builder Docker image.
Configure — HTML

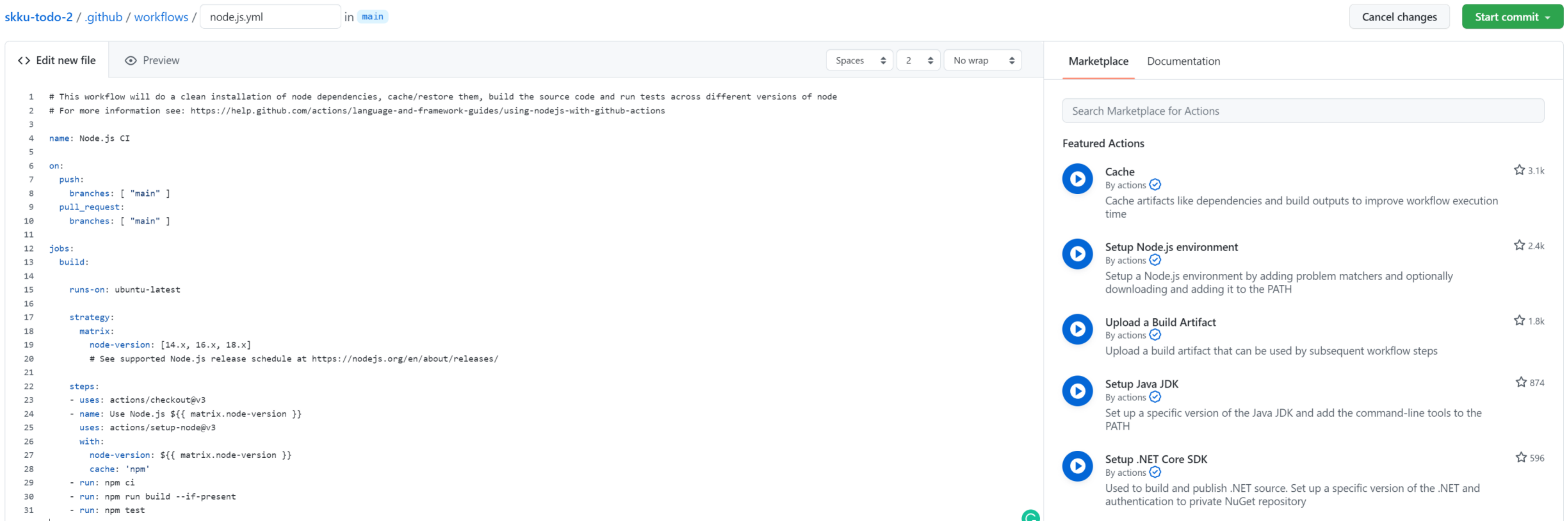
Publish Node.js Package
By GitHub Actions
Publishes a Node.js package to npm.
Configure — JavaScript

Node.js
By GitHub Actions
Build and test a Node.js project with npm.
Configure — JavaScript

Grunt
By GitHub Actions
Build a NodeJS project with npm and grunt.
Configure — JavaScript

Git Actions

- You can configure the workflow, but let’s use the default.
- Press “Start commit” and make a commit.



skku-todo-2 / .github / workflows / node.js.yml in main

Cancel changes | Start commit

Edit new file | Preview | Spaces | 2 | No wrap

```yaml
# This workflow will do a clean installation of node dependencies, cache/restore them, build the source code and run tests across different versions of node
# For more information see: https://help.github.com/actions/language-and-framework-guides/using-nodejs-with-github-actions

name: Node.js CI

on:
  push:
    branches: [ "main" ]
  pull_request:
    branches: [ "main" ]

jobs:
  build:

    runs-on: ubuntu-latest

    strategy:
      matrix:
        node-version: [14.x, 16.x, 18.x]
        # See supported Node.js release schedule at https://nodejs.org/en/about/releases/

    steps:
    - uses: actions/checkout@v3
    - name: Use Node.js ${{ matrix.node-version }}
      uses: actions/setup-node@v3
      with:
        node-version: ${{ matrix.node-version }}
        cache: 'npm'
    - run: npm ci
    - run: npm run build --if-present
    - run: npm test
```

Marketplace | Documentation

Search Marketplace for Actions

Featured Actions

Cache — By actions — ☆ 3.1k
Cache artifacts like dependencies and build outputs to improve workflow execution time

Setup Node.js environment — By actions — ☆ 2.4k
Setup a Node.js environment by adding problem matchers and optionally downloading and adding it to the PATH

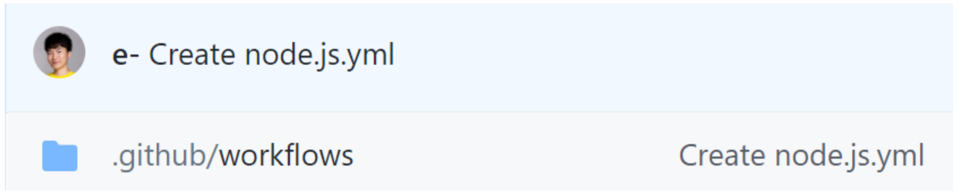
Upload a Build Artifact — By actions — ☆ 1.8k
Upload a build artifact that can be used by subsequent workflow steps

Setup Java JDK — By actions — ☆ 874
Set up a specific version of the Java JDK and add the command-line tools to the PATH

Setup .NET Core SDK — By actions — ☆ 596
Used to build and publish .NET source. Set up a specific version of the .NET and authentication to private NuGet repository

Git Actions

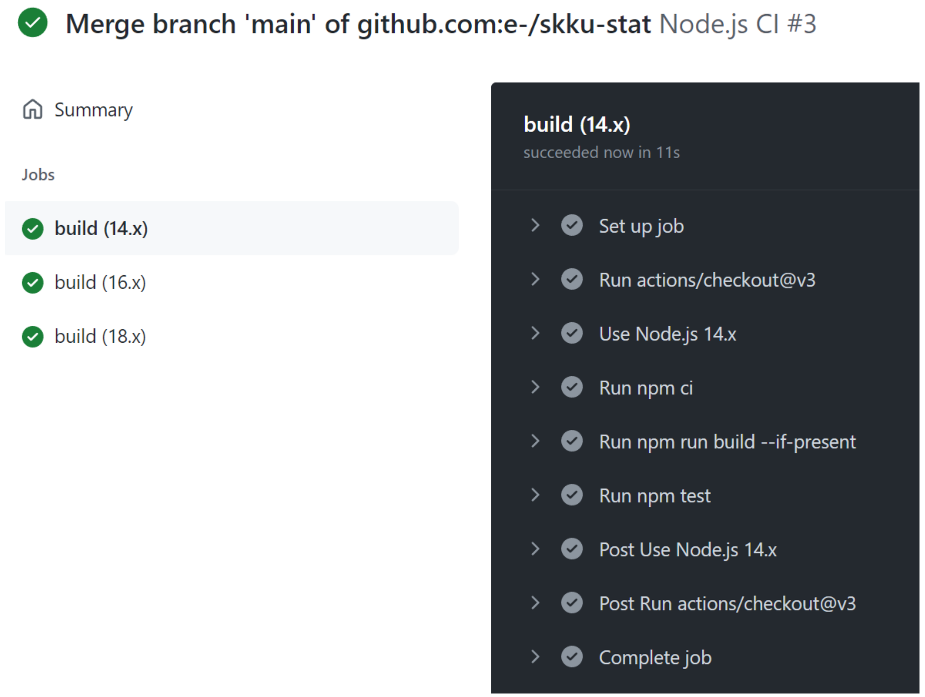
- A configuration file is added to your repo.



e- Create node.js.yml

.github/workflows — Create node.js.yml

- Each time you make a new commit, Git Actions will run the tests on different versions of Node.js.
  - You can view the status in the Action tab.



Merge branch 'main' of github.com:e-/skku-stat Node.js CI #3

Summary

Jobs

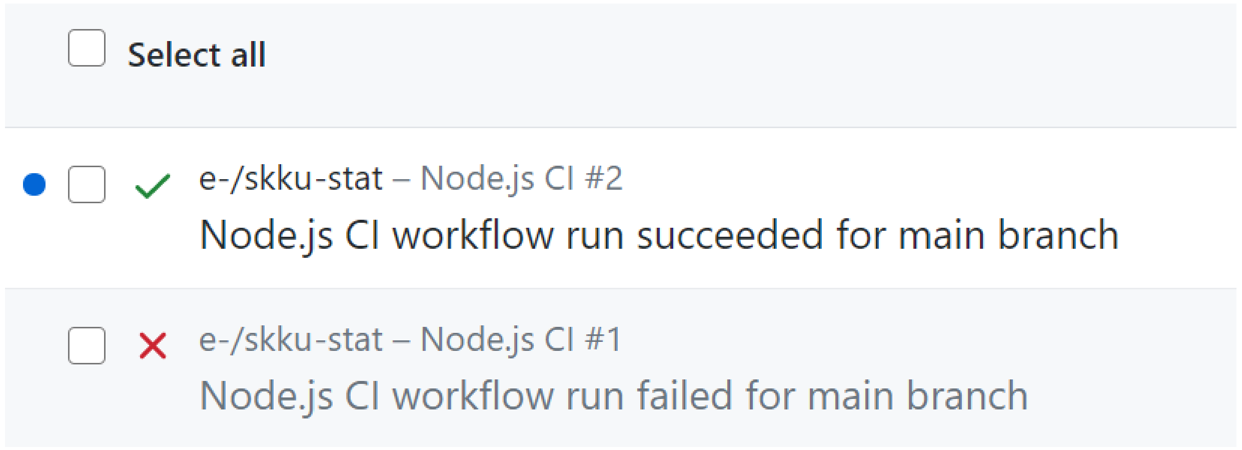
- build (14.x)
- build (16.x)
- build (18.x)

build (14.x)
succeeded now in 11s

- Set up job
- Run actions/checkout@v3
- Use Node.js 14.x
- Run npm ci
- Run npm run build --if-present
- Run npm test
- Post Use Node.js 14.x
- Post Run actions/checkout@v3
- Complete job

Git Actions

- You can also receive notifications if your build fails.



Select all

e-/skku-stat – Node.js CI #2
Node.js CI workflow run succeeded for main branch

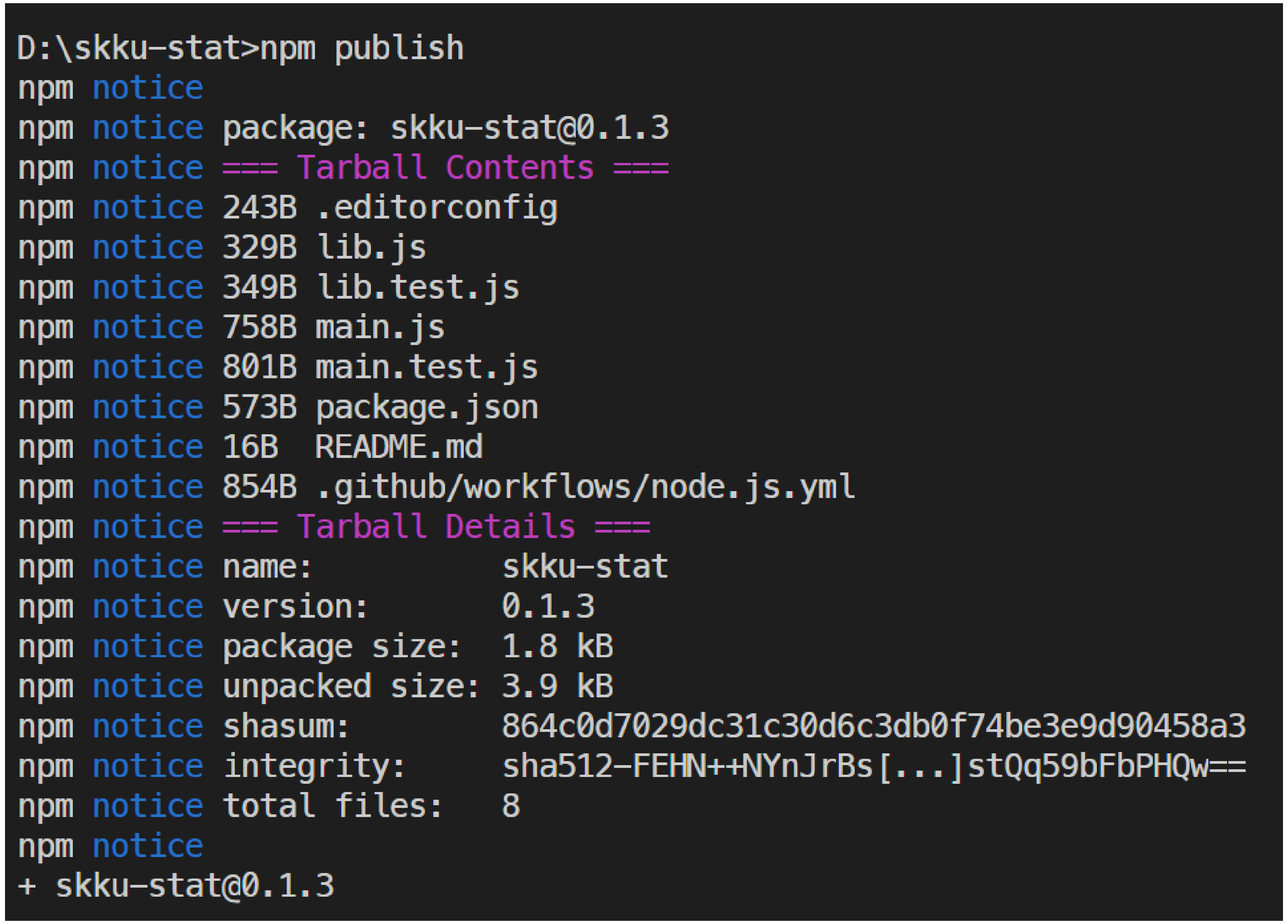
e-/skku-stat – Node.js CI #1
Node.js CI workflow run failed for main branch

Publishing the Project

- Let's publish our project to NPM: `npm publish`
- Change the project name since *"skku-stat"* is already taken.

- You will see errors since you are not logged in.
  - Sign up NPMJS (https://www.npmjs.com/).
  - Finish email verification. If not, your project will not be published.
  - `npm login`
  - `npm publish`
  - Don't forget that you need to increment the version of your project (in *package.json*) at least by 0.0.1 each time you publish.
  - We call this "bump version".

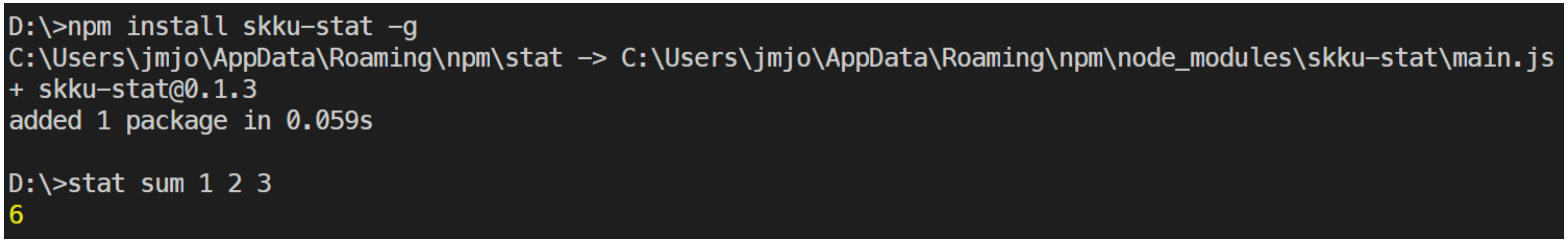
Publishing the Project



```
D:\skku-stat>npm publish
npm notice
npm notice package: skku-stat@0.1.3
npm notice === Tarball Contents ===
npm notice 243B .editorconfig
npm notice 329B lib.js
npm notice 349B lib.test.js
npm notice 758B main.js
npm notice 801B main.test.js
npm notice 573B package.json
npm notice 16B  README.md
npm notice 854B .github/workflows/node.js.yml
npm notice === Tarball Details ===
npm notice name:          skku-stat
npm notice version:       0.1.3
npm notice package size:  1.8 kB
npm notice unpacked size: 3.9 kB
npm notice shasum:        864c0d7029dc31c30d6c3db0f74be3e9d90458a3
npm notice integrity:     sha512-FEHN++NYnJrBs[...]stQq59bFbPHQw==
npm notice total files:   8
npm notice
+ skku-stat@0.1.3
```

Publishing the Project

- `npm install <your_project_name> -g`
- `<your_project_name>` is the project name in *package.json*.



```
D:\>npm install skku-stat -g
C:\Users\jmjo\AppData\Roaming\npm\stat -> C:\Users\jmjo\AppData\Roaming\npm\node_modules\skku-stat\main.js
+ skku-stat@0.1.3
added 1 package in 0.059s

D:\>stat sum 1 2 3
6
```

Summary: Testing and Publishing

- Today, you developed and ***published*** your first open-source project.

- What we used:
  - Git commands, .gitignore, and .editorconfig
  - Command-line arguments, switch-case, array methods, and modules
  - Unit testing + Git actions
  - Publishing to NPM

- HW1: Improve the `stat` command!
  - Details will be announced soon on iCampus.