

Question

Do you have one of the following...

`utils/`

`extras/`

`utils.js`

`utils.ts`

And are they copied from different projects?

Then you might have a
problem

From Zero To Hero

Building and Shipping Your
First JavaScript Library



Hello!

Mike Hartington

Director of DevRel @ ionic

@mhartington
@mhartington.io

What Is A JS Library?

Spoiler, it's complicated

According to npm:

A package is a file or directory that is described by a `package.json` file. A package must contain a **package.json** file in order to be published to the npm registry.

Very Basic

As bare bones as it can be

```
1 ./my-lib
2 |-- package.json
3 |-- index.js
```

Package.json

```
1 {
2   "name": "my-lib",
3   "version": "1.0.0",
4   "main": "index.js",
5 }
```


If it's that simple

Let's make our own package!

Why Bother?

Code isolation

Sharing across projects

Versioned separate from everything else

To the Terminal!

main

Summary

File Structure

```
1  .  
2  ├── index.js  
3  ├── node_modules/  
4  ├── package-lock.json  
5  ├── package.json  
6  └── safe.js
```

Requiring Dependencies

```
1  const os = require('os');
2  const path = require('path');
3  const stream = require('stream');
4
5  const fs = require('fs-extra');
6
7  const statSafe = require('./safe').stat;
8  const readdirSafe = require('./safe').readdir;
```

Exporting Functions

```
1  const statSafe = require('./safe').stat;  
2  const readdirSafe = require('./safe').readdir;  
3  async function fileToString(filePath) {...}  
4  
5  module.exports = {  
6    statSafe,  
7    readdirSafe,  
8    fileToString,  
9  };
```

Exposing to Node

```
1  {  
2    "name": "mh-file-utils",  
3    "version": "0.0.1",  
4    "main": "index.js",  
5    "dependencies": {  
6      "fs-extra": "^11.1.1"  
7    }  
8  }
```


Congrats

You just published an npm package 🎉

But it's 2023

The CJS-ESM Problem

What is CJS?

CommonJS: Package format from 2009

Uses `requires` and `module.exports`

Mainly only used in Node

What is ESM?

ES Modules: Package format from 2015

Uses `import` and `export`

Browser, Node, and beyond!

Both **ESM** and **CJS** are supported in Node*

Up to package authors to configure

It can be...confusing

To the Terminal!

esm-port

Summary

File Structure

```
1  .  
2  |  
3  |— index.mjs  
4  |— node_modules  
5  |— package-lock.json  
6  |— package.json  
   |— safe.mjs
```

Requiring Dependencies

```
1  import { readFile } from 'fs/promises'
2  import 'fs-extra/esm';
3
4  import { stat as statSafe, readdir as readdirSafe } from './safe.mjs';
```

Exporting Functions

```
1  export default {  
2    statSafe,  
3    readdirSafe,  
4    fileToString,  
5  };
```

Exposing to Node

```
1 {  
2   "name": "mh-file-utils",  
3   "version": "0.0.1",  
4   "type": "module",  
5   "main": "index.js",  
6   "dependencies": {  
7     "fs-extra": "^11.1.1"  
8   }  
9 }
```

Congrats

It's about to get more complicated 🙄

Main Vs Module Vs Exports

main

```
"main": "./index.cjs"
```

CJS was exposing the main entry

Only supports one entry point

Not deprecated, but specialized

module

```
"module": "./index.mjs"
```

Older ESM-older entry point

Only one ESM entry point

Not recommended (expect for TypeScript)

exports

Object for defining multiple entry points

Mostly replaces older **module**

Supports both ESM & CJS

To the Terminal!

esm-cjs

Summary

Use **exports** when possible

ESM should be the default (it's 2023)

Dual Packages should be avoided

Adding A Build Step

Build Process

Only if you need it

Simplifies what you ship

Can make dev more complex

tsup

esbuild based build system

JavaScript, TypeScript, and more

It's not webpack 🎉

To the Terminal!

esm-better

Alternatives

unbuild (unified not "un")

Robust ecosystem

Used by vite (internally), and others



@u1F99D · [Follow](#)



Replying to @mhartington

tsup because it's easy, it works, and you can extend it with esbuild plugins

8:30 PM · May 26, 2023



Reply



Copy link

[Read 1 reply](#)

Automation

Making releasing this easy

Conventional Commits

Meaningful commits to drive versioning

`chore()`, `feat(scope)`, **BREAKING**

Makes your team write good messages

Semantic Release

Fully automated release process

Commit-drive versioning

Connect with CI to handle release process

`semantic-release-cli`

To the Terminal!

automation

Summary

Init semantic-release

```
1 npx semantic-release-cli setup
2 ? What is your npm registry? 'https://registry.npmjs.org/'
3 ? What is your npm username? 'mhartington'
4 ? What is your npm password? '[hidden]'
5 ? Provide a GitHub Personal Access Token? ''
6 ? What CI are you using? 'Github Actions'
```


Add a workflow

```
1  name: CI
2  on:
3    push:
4      branches:
5        - stable
6  jobs:
7    test:
8      runs-on: ubuntu-latest
9      steps:
10       - uses: actions/checkout@v3
11         with:
12           fetch-depth: 0
13       - uses: actions/setup-node@v3
14         with:
15           node-version: 18
16       - run: npm ci
17       - run: npm build
18       - name: Release
19         env:
20           NPM_TOKEN: ${ secrets.NPM_TOKEN }
21       run: npm run semantic-release
```

Have meaningful commits

```
1  git commit -m 'feat(scope): add a new feature'  
2  git commit -m 'chore(): update formatting'  
3  git commit -m 'fix(scope): return correct error code'
```

Congrats, you've shipped your first Library!

Wrapping up

You do not need to do all of this

A simple `package.json` + `index.js` is also valid

JS Ecosystem is confusing 🙄

But you got this

mhartington/first-js-lib-demo

esm-port

esm-cjs

esm-better

automation

Questions?

</html>