

# Zero Dependency CLIs

with Node.js



# Who am I?

## Ian Sutherland

Architect, Head of DX and OSS at Neo Financial

Node.js Contributor, Tooling Working Group

Maintainer of Create React App



@iansu





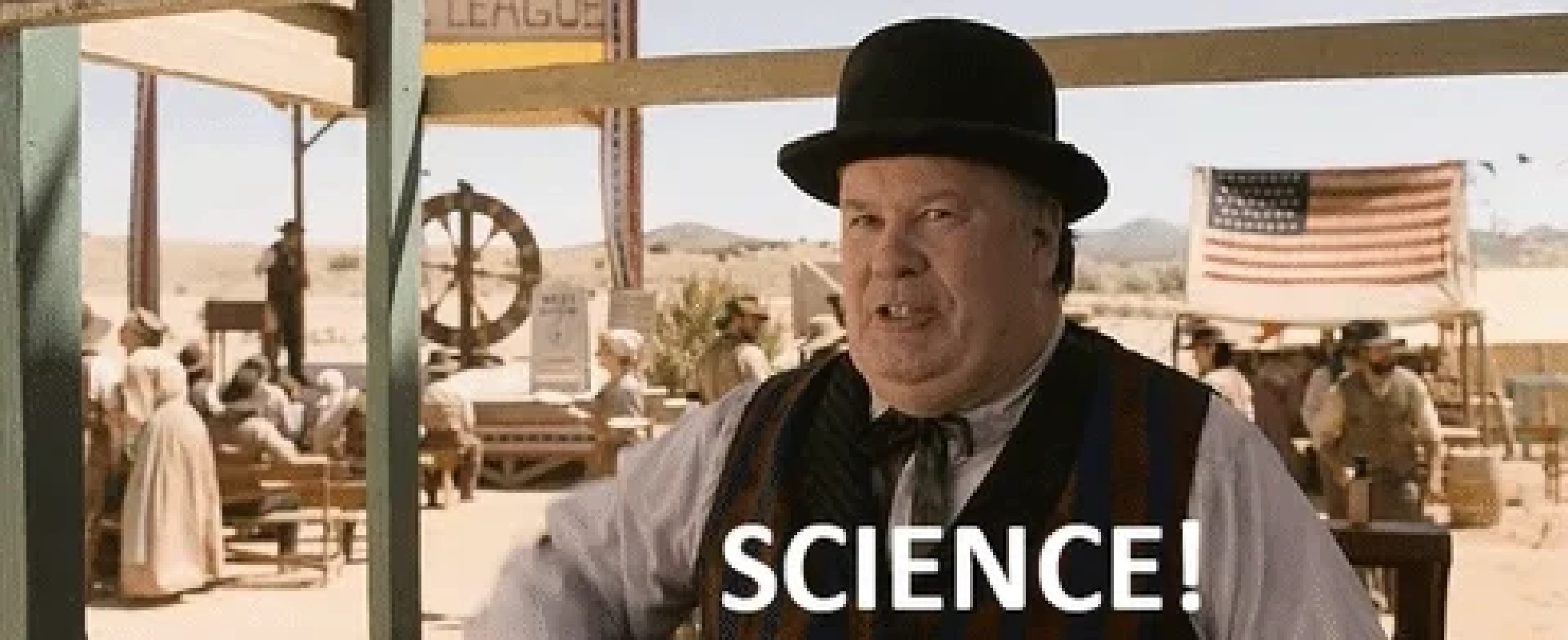
**I SHOULD BUILD A PACKAGE MANAGER**



imgflip.com

IS





# Zero Dependency CLIs

- What is a CLI app and why are we talking about them?
- What are dependencies and why are we trying to avoid them?
- What new Node.js features can help us write CLI apps?
- What new Node.js features might be coming next?
- About that package manager...

# CLI Apps

What are they?

- CLI stands for Command Line Interface
  - Basically a fancy way to describe a script that takes some input and generates output
- These kinds of apps run in a terminal
- Commonly used for dev tools, build scripts and process automation
- Range from simple like `ls` to complex like `git`



# CLI Apps



```
> ls -l1
```

```
README.md
```

```
bin
```

```
node_modules
```

```
package-lock.json
```

```
package.json
```

```
src
```

# CLI Apps



```
> git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   src/index.ts

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

# CLI Apps

Why am I talking about them?

- I work on a number of CLI apps as a part of my job and use them daily
- I just really like CLI apps
  - They're fun to build and the terminal is a challenging, somewhat constrained, environment
  - Serious 80s vibes
- The things I'm talking about here don't just apply to CLI apps
- There are other types of apps that have similar constraints and I'll talk about some of them a bit more later
  - Serverless functions
  - Shell scripts
  - GitHub Actions

# Dependencies

What are they and why do we need them?

- Some other piece of code, like a library, that your program needs to run
- In Node.js dependencies commonly come from, and are installed with, npm
- Other users and contributors will also need to install the same dependencies
- Dependencies need to be installed at build time and deployed with your code
- Node.js does not include a big "standard library" and relies on npm packages instead

# Dependencies

Are dependencies bad?

- Not necessarily
- Dependencies do introduce some overhead in the development and deploy process
  - Users need to install the dependencies to run and develop on a project
  - Dependencies need to be installed or bundled at build time
- If you're building a complex app it probably already has a build and deploy pipeline and many dependencies so adding more dependencies probably doesn't make much of a difference
- For smaller apps and libraries, like CLIs, not having an install or build step makes setup, contributing and distribution easier

# What's New in Node.js?

As of v18.3.0



# New Syntax

I'm going to be using some new syntax in these examples

- `import` instead of `require`
  - ES Modules instead of CommonJS
  - Add `type: module` to `package.json` to use
  - Or name all your files with the `.mjs` extension
- Import Node.js internals with `node:` prefix
  - For example: `import fs from 'node:fs'`
  - Disambiguates internal libraries from npm packages

# fs Package on npm

fs

0.0.1-security • Public • Published 6 years ago



Readme



Explore

BETA



0 Dependencies

## Security holding package

This package name is not currently in use, but was formerly occupied by another package. To avoid malicious use, npm is hanging on to the package name, but loosely, and we'll probably give it to you if you want it.

You may adopt this package by contacting [support@npmjs.com](mailto:support@npmjs.com) and requesting the name.



# Argument Parsing

- Simple scripts often don't take any inputs
- As they get more complex it becomes helpful to accept some user input and arguments
- This also helps make scripts more general and useful in more cases
- Most languages provide at least a basic argument parser

# Argument Parsing

Parsing arguments is surprisingly complicated and can be tedious and error prone

```
mycli --silent
mycli -s
mycli --silent=true
mycli --slient true
mycli --no-silent
```

These all do the same thing, set the ``silent`` argument, but in several different ways

# Argument Parsing

How did we used to handle this?

```
const args = process.argv;  
  
// ['node', 'mycli', '--silent']  
// ['node', 'mycli', '-s']  
// ['node', 'mycli', '--silent=true']  
// ['node', 'mycli', '--slient', 'false']  
// ['node', 'mycli', '--no-silent']
```

More likely you would use a third-party library like `yargs` or `commander`



# Argument Parsing

Now there's a new way to do this in Node.js! Introducing `util.parseArgs` 🎉

```
import { parseArgs } from 'node:util';

const input = process.argv.slice(2);
const options = {
  silent: {
    type: 'boolean',
    short: 's',
  },
};

const { values } = parseArgs({ input, options });
```

# Argument Parsing

Using our previous examples this is what ``parseArgs`` will give us

```
const { values } = parseArgs({ input, options });  
  
// { silent: true }  
// { silent: true }  
// Option '-s, --silent' does not take an argument  
// Option '-s, --silent' does not take an argument  
// Unknown option '--no-silent'
```

# pkgjs/parseargs

Polyfill of proposal for `util.parseArgs()`



 10

Contributors

 5

Used by

 86

Stars

 9

Forks



# Fetch

- Web API for making HTTP requests
- People have been attempting to "make fetch happen" in Node.js for several years



# Fetch





# Fetch

- Web API for making HTTP requests
- People have been attempting to "make fetch happen" in Node.js for several years
- Fortunately we didn't listen to that advice and Fetch is now available in Node.js 18! 🎉
- Built on top of Undici (an HTTP/1.1 client, written from scratch for Node.js)
- Built with the Web Streams API which is also now available in Node.js 18

# Fetch

An example of using `fetch` to query the GitHub API for my profile

```
const response = await fetch('https://api.github.com/users/iansu');
const body = await response.json();

// {
//   "login": "iansu",
//   "name": "Ian Sutherland",
//   "company": "@neofinancial",
//   "blog": "https://iansutherland.ca/",
//   "location": "Calgary, AB"
// }
```

# Life Before Fetch



# Life Before Fetch





**sigfig**

@sigfig · [Follow](#)



git blame should return twitter handles

10:44 PM · Feb 17, 2022



[Read the full conversation on Twitter](#)



297



Reply



Copy link

[Read 11 replies](#)



**Jared Forsyth**

@jaredforsyth · [Follow](#)



git blast



**sigfig** @sigfig

git blame should return twitter handles

1:39 PM · Feb 18, 2022



8



Reply



Copy link

[Read 1 reply](#)

```
#!/usr/bin/env node
```

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

const githubRequest = async (path) => {
  return new Promise((resolve, reject) => {
    const options = {
      hostname: 'api.github.com',
      port: 443,
      path: path,
      method: 'GET',
      headers: { 'User-Agent': 'git-blast' },
    };

    https
      .get(options, (resp) => {
        let data = '';
        // A chunk of data has been received.
        resp.on('data', (chunk) => {
          data += chunk;
        });
        // The whole response has been received. Print out the result.
        resp.on('end', () => {
          // console.log(data);

          resolve(JSON.parse(data));
        });
      })
      .on('error', (err) => {
        console.log('Error: ' + err.message);
      });
  });
}
```

```
const githubRequest = async (path) => {  
  try {  
    const response = await fetch(`https://api.github.com/${path}`, { headers: { 'User-Agent': 'git-blast' } });  
    const body = await response.json();  
  
    return body;  
  } catch (error) {  
    console.log('Error: ' + error.message);  
  }  
}
```





# iansu/git-blast

A custom git command that works just like git blame but shows the committer's Twitter username instead of their name



1

Contributor



0

Issues



2

Stars



0

Forks



# Test Runner

- Basic API to create tests, both synchronous and asynchronous
- Supports subtests, ``skip`` tests, ``todo`` tests and ``only`` tests
- Can be used with Node's existing built in ``assert`` library
- Simple runner to run tests
  - You can provide a list of files
  - It will automatically run files in a ``test`` directory
  - It will automatically run files that end in ``.test.js``, ``.test.cjs`` and ``.test.mjs``

# Test Runner

Let's write some basic tests

```
import test from 'node:test';
import assert from 'node:assert';

test('synchronous test', (t) => {
  assert.strictEqual(1, 1);
});

test('asynchronous test', async (t) => {
  return Promise.resolve(1).then(value => {
    assert.strictEqual(value, 1);
  });
});
```

# Test Runner

Now let's run those tests

```
node --test
```

```
TAP version 13
```

```
ok 1 - index.test.js
```

```
---
```

```
  duration_ms: 0.038415792
```

```
...
```

```
1..1
```

```
# tests 1
```

```
# pass 1
```

```
# fail 0
```

```
# skipped 0
```

```
# todo 0
```

```
# duration_ms 0.095865209
```



# Recursive Filesystem Operations

- Built in methods like `fs.readFile`, `fs.unlink`, etc. are great for working with a small number of files
- If you want to work with a large number of files and/or directories recursive operations are helpful
- In the past doing any recursive filesystem operations required adding one or more dependencies to your project
- Not anymore!

# Recursive Filesystem Operations

`fs.mkdir` - recursively create a directory and any non-existent parent directories

```
import { mkdir } from 'node:fs/promises';  
  
await mkdir('/tmp/a/b/c', { recursive: true });
```

# Recursive Filesystem Operations

`fs.rm` - recursively delete a directory and all child directories and files

```
import { rm } from 'node:fs/promises';  
  
await rm('/tmp/a', { recursive: true, force: true });
```

# Recursive Filesystem Operations

`fs.cp` - recursively copy a directory and all child directories and files

```
import { cp } from 'node:fs/promises';  
  
await cp('/tmp/a', '/tmp/z', { recursive: true });
```

 This API is currently marked as experimental



# Recursive Filesystem Operations

`fs.readdir` - recursively read the contents of a directory and all child directories

```
import { readdir } from 'node:fs/promises';

await readdir('/tmp/a', { recursive: true });
```

ⓘ There is currently an open PR for this feature: [nodejs/node#41439](https://github.com/nodejs/node/pull/41439)

# What's Next?



# What's Next?

Glob?



```
"packages/**/package.json"
```

# What's Next?

Self-contained Executables?



```
> node compile index.js
```

# What's Next?

TypeScript?!



```
> node index.ts  
Hello, World!
```

# What's Next?

Want to help figure out, and build, what's next?

Join the Node Tooling Group!

[github.com/nodejs/tooling](https://github.com/nodejs/tooling)




# About That Package Manager

- I wanted a way to try out all these new features
- What's a CLI app that makes API requests, downloads files and manipulates the filesystem? 🤔
- A package manager! 📦
- Let's build a package manager...?! 🤔


# Bad Package Manager


Introducing Bad Package Manager (or ``bad``). It's a package manager that is... bad. For science!


iansu/**bad-**  
**package-manager**





A bad package manager. For science!

 1  
Contributor

 0  
Issues

 0  
Stars

 0  
Forks





# Bad Package Manager

Introducing Bad Package Manager (or `bad`). It's a package manager that is... bad. For science!

- Might one day be a good CLI app
- Will probably never be a good package manager
- Ongoing project to test new features in a Node.js CLI app
- Currently uses `parseArgs`, `fetch`, `test`, `rm`, `cp`, `mkdir`
- Contributions welcome!
  - [iansu/bad-package-manager](https://github.com/iansu/bad-package-manager)



# Bad Package Manager

Currently supported features

- `bad install` - install all `dependencies` and `devDependencies` from `package.json`
- `bad clean` - delete `node_modules`



# Bad Package Manager

## Future ideas

- ``bad install <package>`` - install the named package and add it to ``dependencies`` in ``package.json``
- ``bad install --dev <package>`` - install the named package and add it to ``devDependencies`` in ``package.json``
- ``bad uninstall <package>`` - uninstall previously installed ``dependencies`` and ``devDependencies``
- ``bad run <script>`` - run scripts specified in ``package.json``
- ``bad ls <package>`` - list part of the package tree (hopefully with ``fs.readdir``)
- ?



# Thanks for Listening!

## Ian Sutherland

Architect, Head of DX and OSS at Neo Financial

Node.js Contributor, Tooling Working Group

Maintainer of Create React App



@iansu



# Thanks for Listening!

## Ian Sutherland

Architect, Head of DX and OSS at Neo Financial

Node.js Contributor, Tooling Working Group

Maintainer of Create React App



@iansu

