YOU DON'T NEED A DEPENDENCY

BRIAN MUENZENMEYER

JS JSConf, October 2025

👋 ABOUT ME

PRINCIPAL ENGINEER

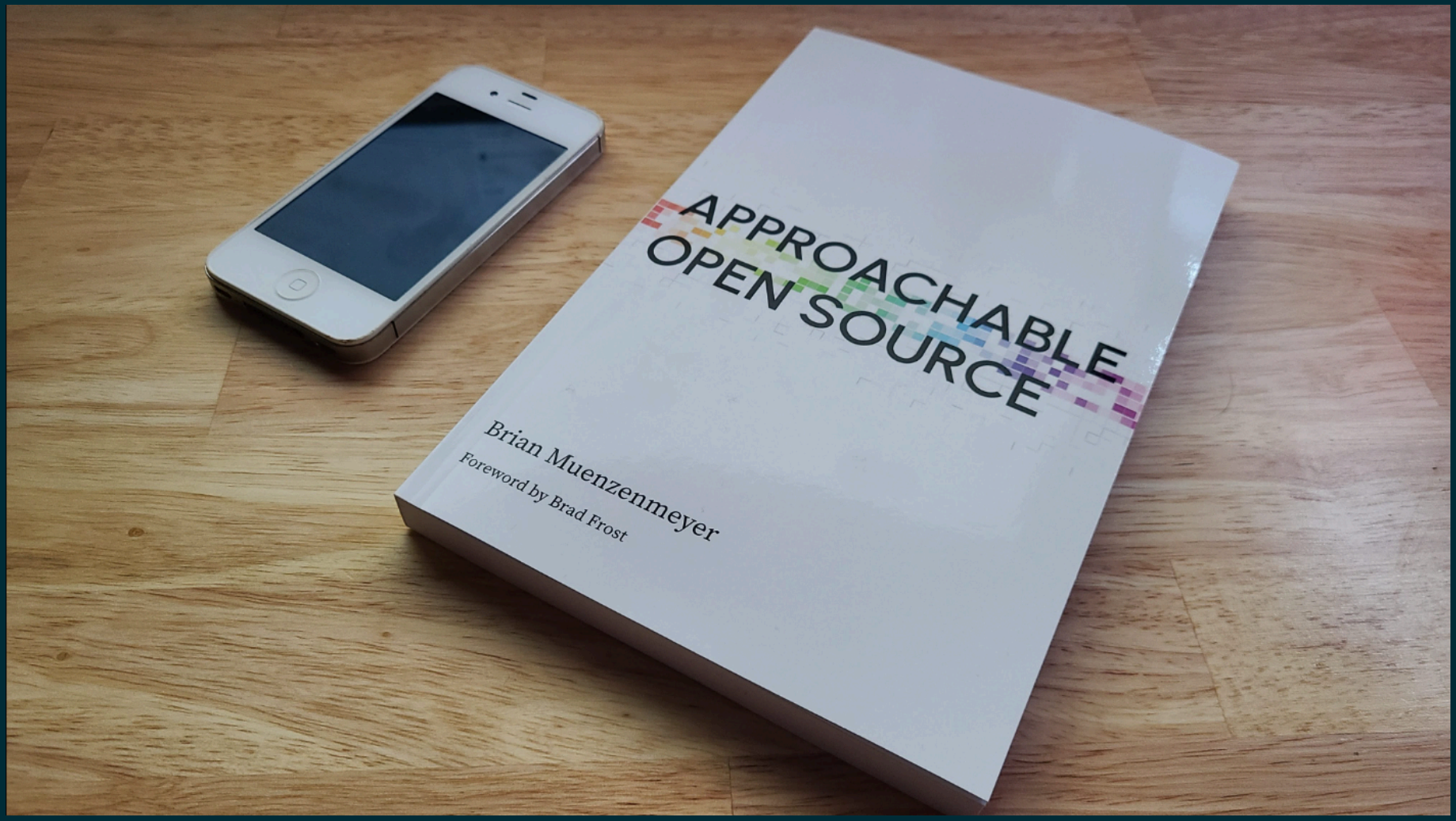
LEADER,
OPEN SOURCE PROGRAM OFFICE

@BRIANMUENZENMEYER.COM

@BMUENZENMEYER

@ME

AUTHOR, APPROACHABLE OPEN SOURCE



MAINTAINER, NODEJS.ORG

Run JavaScript Everywhere

Node.js® is a free, open-source, cross-platform JavaScript runtime environment that lets developers create servers, web apps, command line tools and scripts.

Get Node.js®

Get security support
for EOL Node.js versions

Create an HTTP Server | Write Tests | Read and Hash a File | Streams Pipeline | Work with Threads

```javascript
// server.mjs
import { createServer } from 'node:http';

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

// starts a simple http server locally on port 3000
server.listen(3000, '127.0.0.1', () => {
  console.log('Listening on 127.0.0.1:3000');
});

// run with `node server.mjs`
```

JavaScript — Copy to clipboard

Learn more what Node.js is able to offer with our Learning materials.

DEPENDENCIES ARE GREAT!

The Rule of Least Power

TAG Finding 23 February 2006

This version:
http://www.w3.org/2001/tag/doc/leastPower-2006-02-23
Latest version:
http://www.w3.org/2001/tag/doc/leastPower
Previous versions:
http://www.w3.org/2001/tag/doc/leastPower-2006-2-13.html http://www.w3.org/2001/tag/doc/leastPower-2006-01-23.html http://www.w3.org/2001/tag/doc/leastPower-2005-12-20.html http://www.w3.org/2001/tag/doc/leastPower-2005-12-19.html
Editors:
Tim Berners-Lee, World Wide Web Consortium <timbl@w3.org>
Noah Mendelsohn, IBM Corporation <noah_mendelsohn@us.ibm.com>

This document is also available in these non-normative formats: XML.

Abstract

When designing computer systems, one is often faced with a choice between using a more or less powerful language for publishing information, for expressing constraints, or for solving some problem. This finding explores tradeoffs relating the choice of language to reusability of information. The "Rule of Least Power" suggests choosing the least powerful language suitable for a given purpose.

Status of this Document

This document has been produced by the W3C Technical Architecture Group (TAG). The TAG approved this finding at its 14 February 2006 teleconference. The text of this finding was adapted from Principle of Least Power in [Axioms]. Earlier drafts of this finding had the working title: The Principle of Least Power. Additional TAG findings, both accepted and in draft state, may also be available. The TAG may incorporate this and other findings into future versions of the [AWWW].

Please send comments on this finding to the publicly archived TAG mailing list www-tag@w3.org (archive).

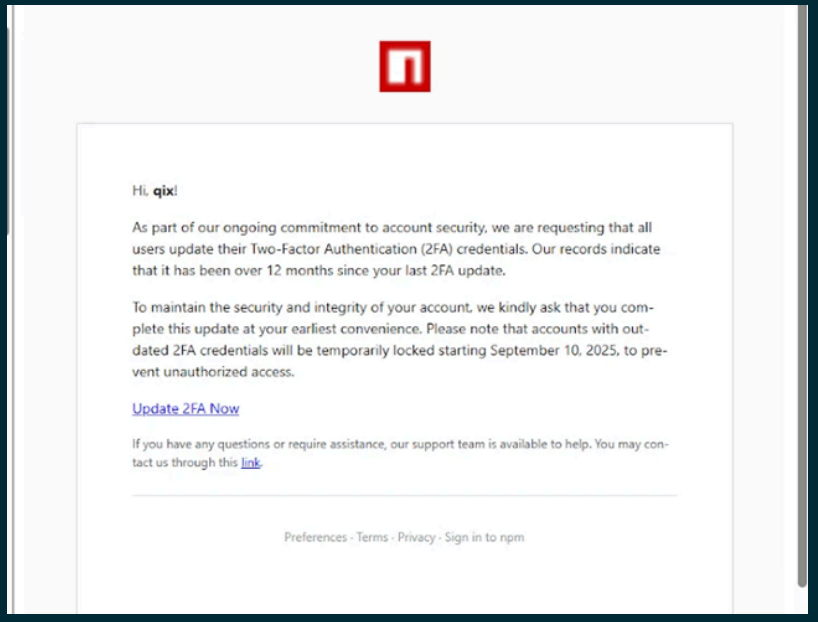
Table of Contents

---

1 Language Power and Information Reuse

The World Wide Web is unique in its ability to promote information reuse on a global scale. Information published on the Web can be flexibly combined with other information, read by a broad range of software tools, and browsed by human users of the Web. For such reuse to succeed, the broadest possible range of tools must be capable of understanding the data on the Web, and the relationships among that data. Thus, when publishing information or programs on the Web, the choice of language is important. This finding explores the connection between the choice of computer language and the reusability of information represented in that language.

CLICK ME



Hi, **qix**!

As part of our ongoing commitment to account security, we are requesting that all users update their Two-Factor Authentication (2FA) credentials. Our records indicate that it has been over 12 months since your last 2FA update.

To maintain the security and integrity of your account, we kindly ask that you complete this update at your earliest convenience. Please note that accounts with outdated 2FA credentials will be temporarily locked starting September 10, 2025, to prevent unauthorized access.

Update 2FA Now

If you have any questions or require assistance, our support team is available to help. You may contact us through this link.

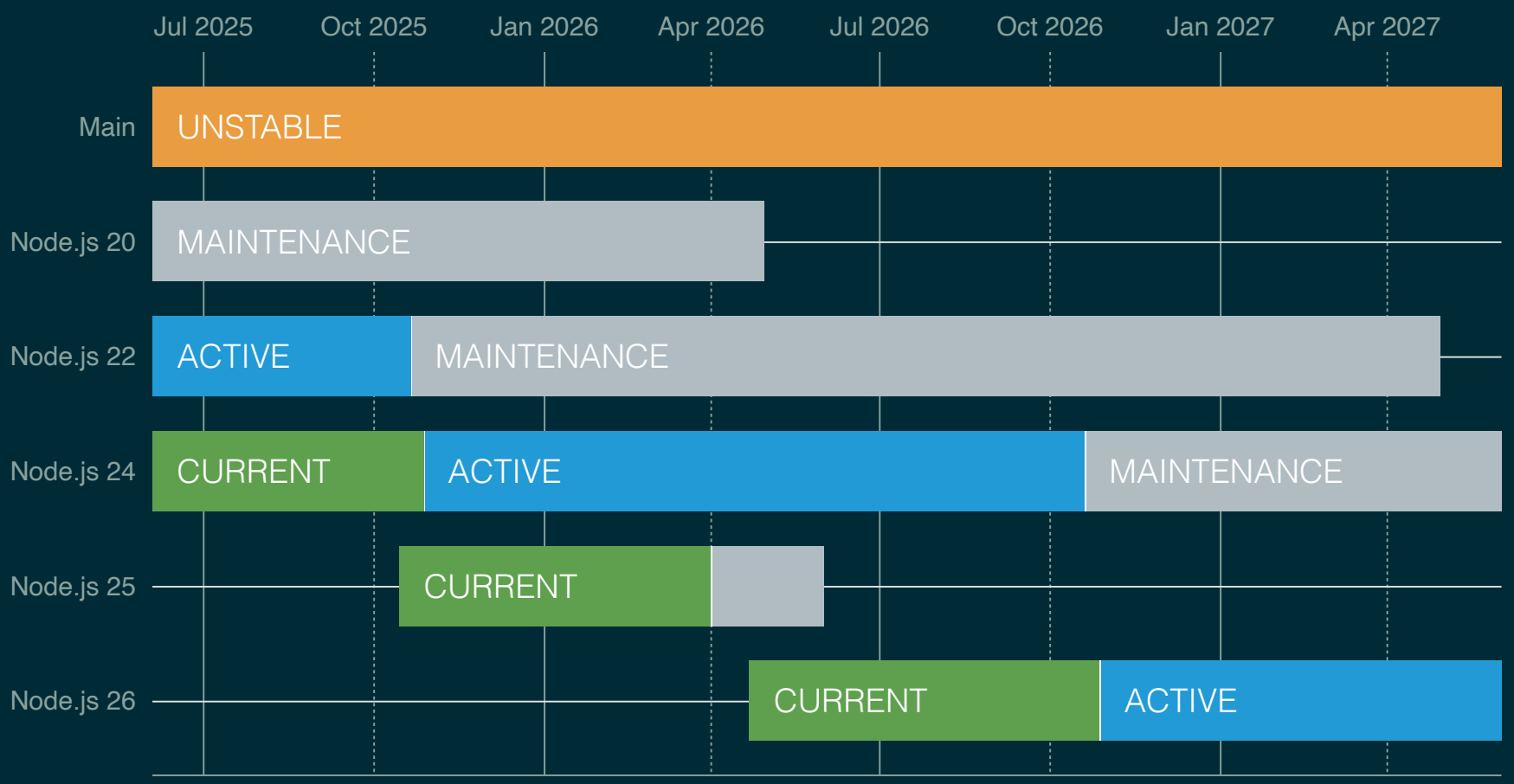
Preferences · Terms · Privacy · Sign in to npm

NEW(ISH) FEATURES

Capability	Introduced
testing source code	16.17.0
watching source code	16.19.0
parsing arguments	18.3.0
reading environment	20.6.0
styling output	20.12.0
glob files	22.0.0
run typescript	22.6.0

NEW(ISH) FEATURES

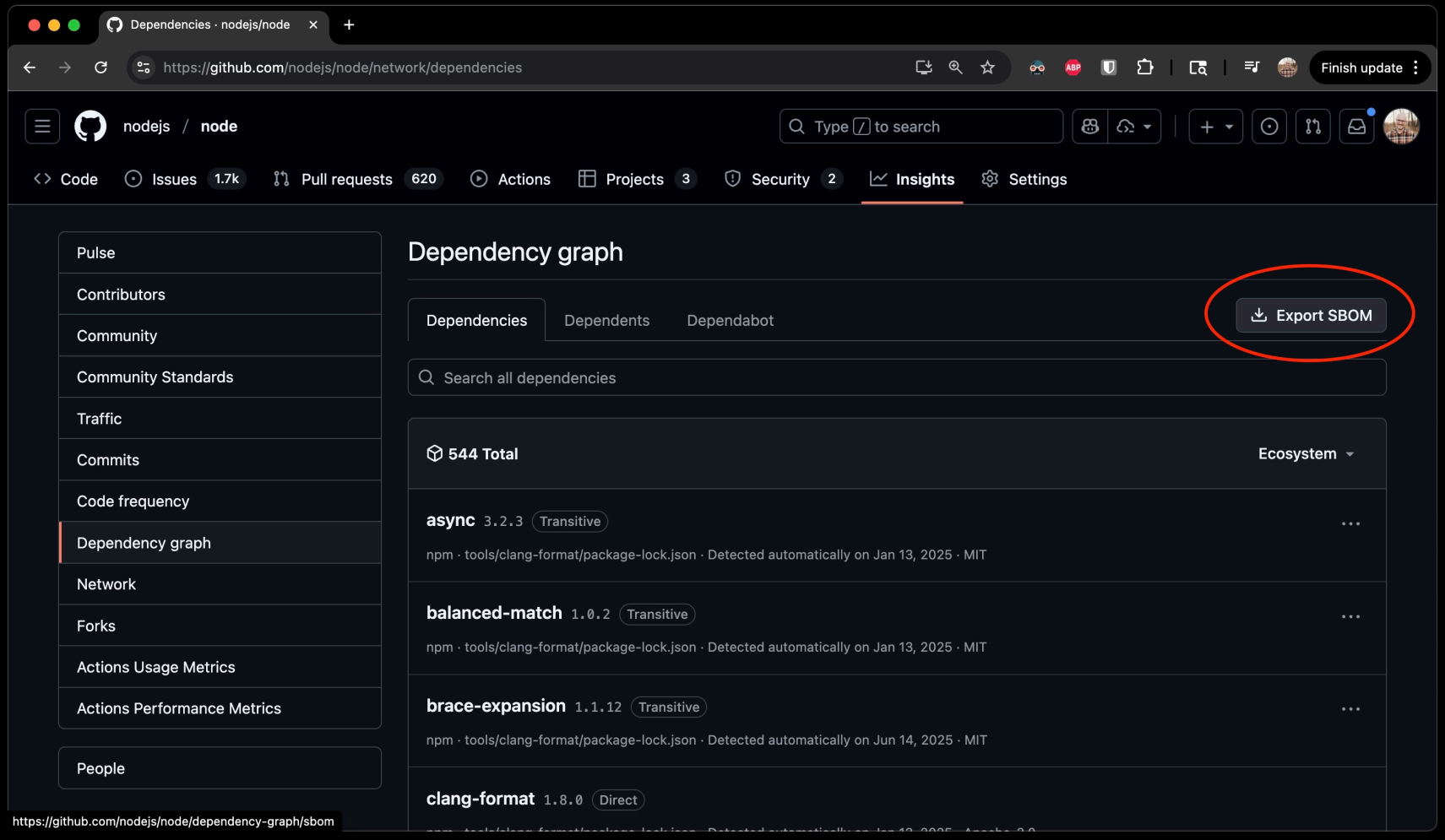
Capability	Introduced
testing source code	16.17.0
watching source code	16.19.0
parsing arguments	18.3.0
reading environment	20.6.0
styling output	20.12.0
glob files	22.0.0
run typescript	22.6.0


Jul 2025
Oct 2025
Jan 2026
Apr 2026
Jul 2026
Oct 2026
Jan 2027
Apr 2027
Main
UNSTABLE
Node.js 20
MAINTENANCE
Node.js 22
ACTIVE
MAINTENANCE
Node.js 24
CURRENT
ACTIVE
MAINTENANCE
Node.js 25
CURRENT
Node.js 26
CURRENT
ACTIVE

These releases potentially replace an external dependency in your project.

Capability	Dependency Replaced
testing source code	jest, ava, ts-jest
watching source code	nodemon
parsing arguments	commander, yargs
reading environment	dotenv
styling output	colors, chalk
glob files	glob, globby, fast-glob
run typescript	ts-node, tsx

SAMPLE PROJECT


Dependencies · nodejs/node
https://github.com/nodejs/node/network/dependencies
Finish update
nodejs / node
Type / to search
Code
Issues 1.7k
Pull requests 620
Actions
Projects 3
Security 2
Insights
Settings
Pulse
Contributors
Community
Community Standards
Traffic
Commits
Code frequency
Dependency graph
Network
Forks
Actions Usage Metrics
Actions Performance Metrics
People
Dependency graph
Dependencies
Dependents
Dependabot
Export SBOM
Search all dependencies
544 Total
Ecosystem
async 3.2.3 Transitive
npm · tools/clang-format/package-lock.json · Detected automatically on Jan 13, 2025 · MIT
balanced-match 1.0.2 Transitive
npm · tools/clang-format/package-lock.json · Detected automatically on Jan 13, 2025 · MIT
brace-expansion 1.1.12 Transitive
npm · tools/clang-format/package-lock.json · Detected automatically on Jun 14, 2025 · MIT
clang-format 1.8.0 Direct
https://github.com/nodejs/node/dependency-graph/sbom

```
$ sbom-parser demos/nodejs_node_480562.json
📋 SBOM Summary
═══════════════════════════════════════════════════

Total packages: 482

📄 License Distribution:
  MIT: 398 (82.6%)
  unknown: 26 (5.4%)
  ISC: 21 (4.4%)
  Apache-2.0: 16 (3.3%)
  BSD-2-Clause: 9 (1.9%)
  BSD-3-Clause: 3 (0.6%)
  Python-2.0: 3 (0.6%)
  MS-RL: 2 (0.4%)
  CC0-1.0: 1 (0.2%)
  CC-BY-3.0: 1 (0.2%)
  CC-BY-4.0: 1 (0.2%)
  BSD-2-Clause AND BSD-2-Clause-Views: 1 (0.2%)
```

BEFORE

```
{
  "dependencies": {
    "chalk": "5.6.0",
    "commander": "14.0.0",
    "dotenv": "17.2.2",
    "glob": "11.0.3"
  },
  "devDependencies": {
    "@types/jest": "30.0.0",
    "jest": "30.1.3",
    "nodemon": "3.1.10",
    "ts-jest": "29.4.1",
    "typescript": "5.9.2"
  },
  "scripts": {
    "build": "tsc src/parser.ts --outDir src --target ES2022 --
```

Core business logic by GitHub + Claude:

TESTING SOURCE CODE

16.17.0 (August 2022)

Tests of all kinds build confidence in release.

For many projects, I'd turn to jest to test my code.

For many projects, I'd turn to jest to test my code.

It's been the default for so long, is part of the OpenJS Foundation, and enjoys a large ecosystem of tools and attention, making it hard to argue against.

We can test our `SBOMParser` class with this test:

```
import { SBOMParser } from "./parser.js"

describe("SBOMParser", () => {
  test("should throw error for invalid JSON", () => {
    expect(() => {
      SBOMParser.parseSBOM("invalid json")
    }).toThrow("Failed to parse SBOM JSON")
  })
})
```

And then run it with:

```
"test": "node --experimental-vm-modules node_modules/jest/bin/j
"test:watch": "pnpm test -- --watch"
```

And then run it with:

```
"test": "node --experimental-vm-modules node_modules/jest/bin/j
"test:watch": "pnpm test -- --watch"
```

Already there's trouble brewing...

Node.js now includes a built-in test runner,

`node --test`

```
-"test": "node --experimental-vm-modules node_modules/jest/bin/
-"test": "pnpm test:jest --watch",
+"test": "node --test",
+"test:watch": "node --test --watch",
```

Here's a test diff:

```
+import assert from "node:assert"
+import { test, describe } from "node:test"
import { SBOMParser } from "./parser.js"

describe("SBOMParser", () => {
  test("should throw error for invalid JSON", () => {
-  expect(() => {
+  assert.throws(() => {
    SBOMParser.parseSBOM("invalid json")
-  }).toThrow("Failed to parse SBOM JSON")
+  }, /Failed to parse SBOM JSON/)
  })
})
```

I'm not interested in the code golf here, but it is worth emphasizing two things:

1. Jest's support for ESM is still evolving (pinned issue since 2020), not yet with a polished developer experience.

1. Jest's support for ESM is still evolving (pinned issue since 2020), not yet with a polished developer experience.

1. Jest's support for ESM is still evolving (pinned issue since 2020), not yet with a polished developer experience.
2. Jest is slower, even with one suite. Benchmarking via `time pnpm test` against both showed the Node.js test runner to be 5 times faster.

👀 WATCHING SOURCE CODE

16.19.0 (December 2022)

Tasks re-run when as your source code changes during development.

👀 WATCHING SOURCE CODE

16.19.0 (December 2022)

Tasks re-run when as your source code changes during development.

Stopping and restarting the server each time is a pain.

Our sample project contains this script in our `package.json`:

```
"dev": "nodemon --watch src src/cli.js"
```

Our sample project contains this script in our `package.json`:

```
"dev": "nodemon --watch src src/cli.js"
```

Works great, and has for a long time.

But now, Node.js has built-in arguments `--watch` and `--watch-path`.

We can replace this with:

```
-"dev": "nodemon --watch src src/cli.js",
+"dev": "node --watch-path=src src/cli.js",
```

Not much different at the surface. It works for this use case.

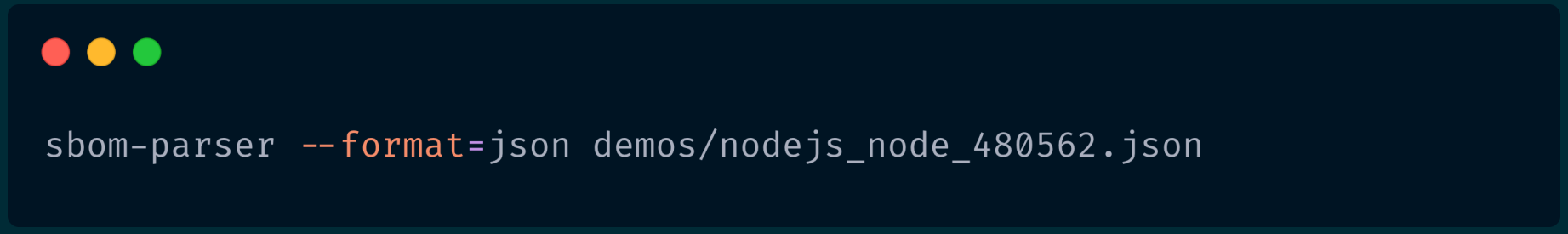
Not much different at the surface. It works for this use case.

Critically, it survives parsing errors in the JavaScript.

PARSING ARGUMENTS

`18.13.0` (January 2023)

Our CLI should support configurable, runtime usecases.


sbom-parser --format=json demos/nodejs_node_480562.json

Node.js supports `process.argv` since `0.1.27`

so what's the fuss?

Node.js supports `process.argv` since `0.1.27`

so what's the fuss?

Tools like yargs and commander have been the go-to for a long time.

Our arguments start parsed with commander:

```
import { Command } from "commander"
const program = new Command()

program
  .name("sbom-parser")
  .argument(
    "[sbom-file]",
    "Path to the SBOM JSON file (optional if using --glob)"
  )
  .option("-f, --format <format>", "Output format: stdout or js
  .action(async (sbomFilePath, options) => {
    ...
  })
await program.parseAsync()
```

We can do this:

```
import { parseArgs } from "node:util"

const parsed = parseArgs({
  options: {
    format: {
      type: "string",
      short: "f",
    },
  },
  allowPositionals: true,
})

const { values: options, positionals } = parsed
```

Lops off the two first arguments by default, synchronous by default, and enough for my needs.

Lops off the two first arguments by default, synchronous by default, and enough for my needs.

Node.js also throws an error for missing or extra params - which is nice - and again, perhaps enough.

READING ENVIRONMENT

20.6.0 (September 2023)

Environment variables provide flexibility and portability to code.

The obvious choice for years was dotenv:

```
import dotenv from "dotenv"

dotenv.config()

const format = options.format
  ? options.format
  : process.env.DEFAULT_FORMAT
```

But Node.js can do this too!

Delete that dotenv import.

```
-import dotenv from "dotenv"

-dotenv.config()

const format = options.format
  ? options.format
  : process.env.DEFAULT_FORMAT
```

We add this to our `package.json` dev script:

```
-"dev": "node --watch-path src src/server.js",
+"dev": "node --env-file=.env --watch-path src src/server.js",
```

We get multiple file support with overrides and a familiar enough syntax to dotenv files.

We get multiple file support with overrides and a familiar enough syntax to dotenv files.

Node.js can also error or gracefully handle missing env files, the choice is yours.

STYLING OUTPUT

20.12.0 (March 2024)

Terminal output styling can improve UX.

The ubiquitous module `chalk` suffices:

```
import chalk from 'chalk'
...
lines.push(
  `Total packages: ${chalk.green.bold(summary.totalPackages)}`
)
```

But look at this native Node.js code:

```
-import chalk from 'chalk'
+import { styleText } from 'node:util'
...
lines.push(
- `Total packages: ${chalk.green.bold(summary.totalPackages)}
+ `Total packages: ${styleText(["green", "bold"], summary.total
);
```

Rudimentary support for terminal color detection and environment variable overrides.

GLOB FILES

`22.0.0` (April 2024)

Operating on groups of files is a common capability.

Plenty of ways to do this.

```
import { glob } from "glob"

...

const files = await glob(options.glob)
```

Lots of tools race to be fastest.

Another Node.js internal:

```
- import { glob } from "glob"
+ import { glob } from "node:fs/promises"

...

- const files = await glob(options.glob)
+ const files = await Array.fromAsync(glob(options.glob))
```

CHECKPOINT

Nice, we got through quite a bit.

But it's always smart to save before the big boss fight.

What's left?

What's left?

TypeScript.

TYPESCRIPT

22.6.0 (August 2024)

Native TypeScript support was consistent community ask and wedge issue.

Here's a tiny snippet of our parser class:

```
static filterByLicense(summary, licenseFilter) {
  return summary.dependencies.filter((dep) =>
    dep.license.toLowerCase().includes(
      licenseFilter.toLowerCase())
    );
}
```

Let's add some types to match the SBOM schema

```
export interface SBOMDependency {
  name: string
  version: string
  license: string
  packageManager: string
  copyright?: string
  spdxId: string
  downloadLocation: string
}

export interface SBOMSummary {
  totalPackages: number
  dependencies: SBOMDependency[]
  licenses: Record<string, number>
  packageManagers: Record<string, number>
}
```

```
static filterByLicense(
  summary: SBOMSummary,
  licenseFilter: string
  ): SBOMSummary {
  return summary.dependencies.filter((dep) =>
    dep.license.toLowerCase().includes(
      licenseFilter.toLowerCase()
    );
}
```

As of v23.6.0 in January...

THIS JUST RUNS!

We still get feedback in editor.

```
static filterByLicense(
  summary: SBOMSummary,
  licenseFilter: 'GPL' | 'MIT' | 0 //zero-clause BSD mistake
): SBOMSummary {
  const filteredDependencies = summary.dependencies.filter((dep) =>
    dep.license.toLowerCase().includes(licenseFilter.toLowerCase())
  )
```

REPLY GUY: WELL ACTUALLY...

you didn't build the project at all, it only removed the typings, flow-style. Sad!

REPLY GUY: WELL ACTUALLY...

you didn't build the project at all, it only removed the typings, flow-style. Sad!

But wait, I say, my editor gave me immediate feedback of the error, without needing a build process at all.

SOME NOTES

- no type-stripping under node_modules

SOME NOTES

- no type-stripping under node_modules (perhaps ever)

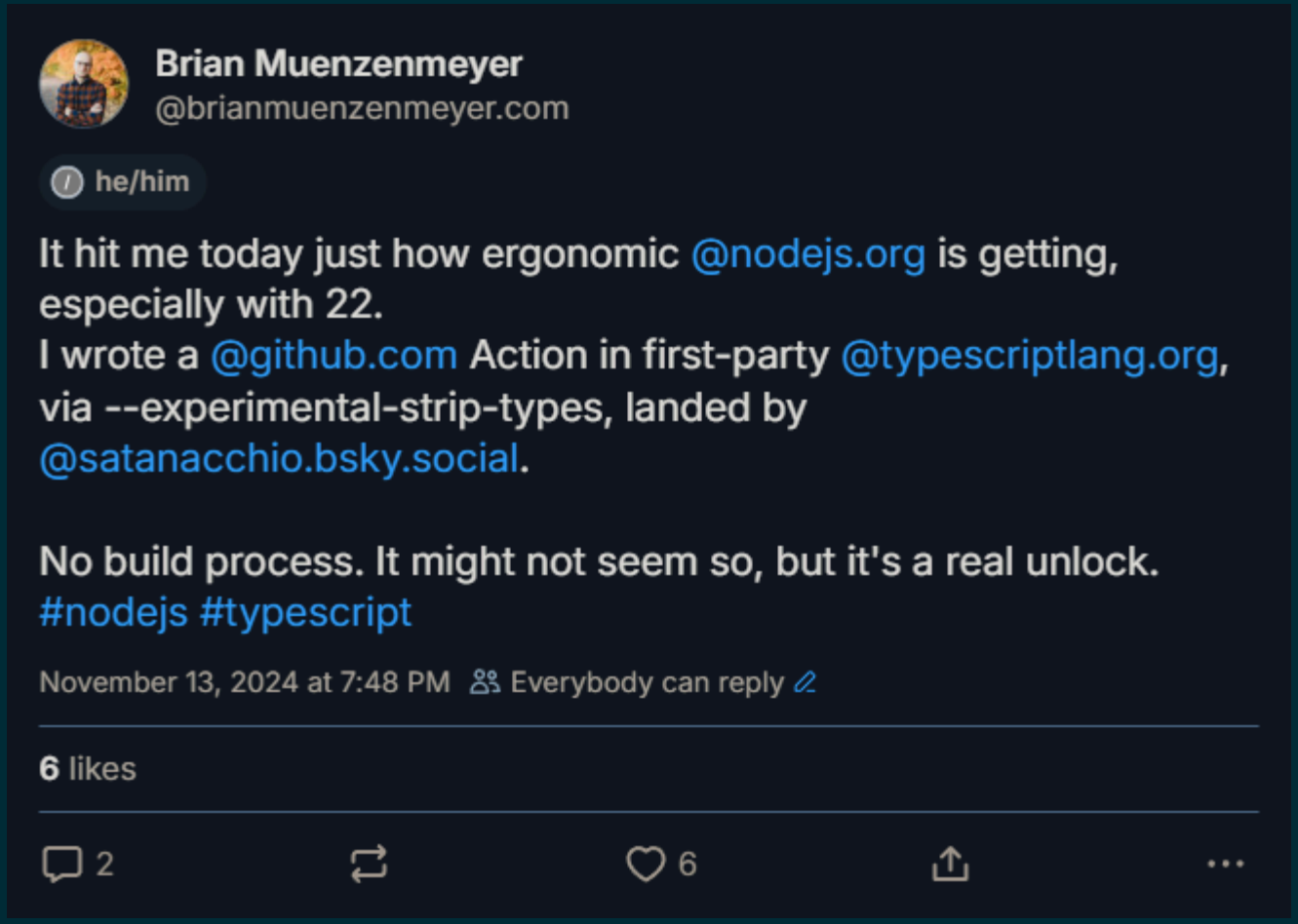
SOME NOTES

- no type-stripping under node_modules (perhaps ever)
- `--experimental-transform-types`

SOME NOTES

- no type-stripping under node_modules (perhaps ever)
- `--experimental-transform-types`
- `--erasableSyntaxOnly` for `TS@5.8`

HOPE OF 2024...



Brian Muenzenmeyer
@brianmuenzenmeyer.com

he/him

It hit me today just how ergonomic @nodejs.org is getting, especially with 22.
I wrote a @github.com Action in first-party @typescriptlang.org, via --experimental-strip-types, landed by @satanacchio.bsky.social.

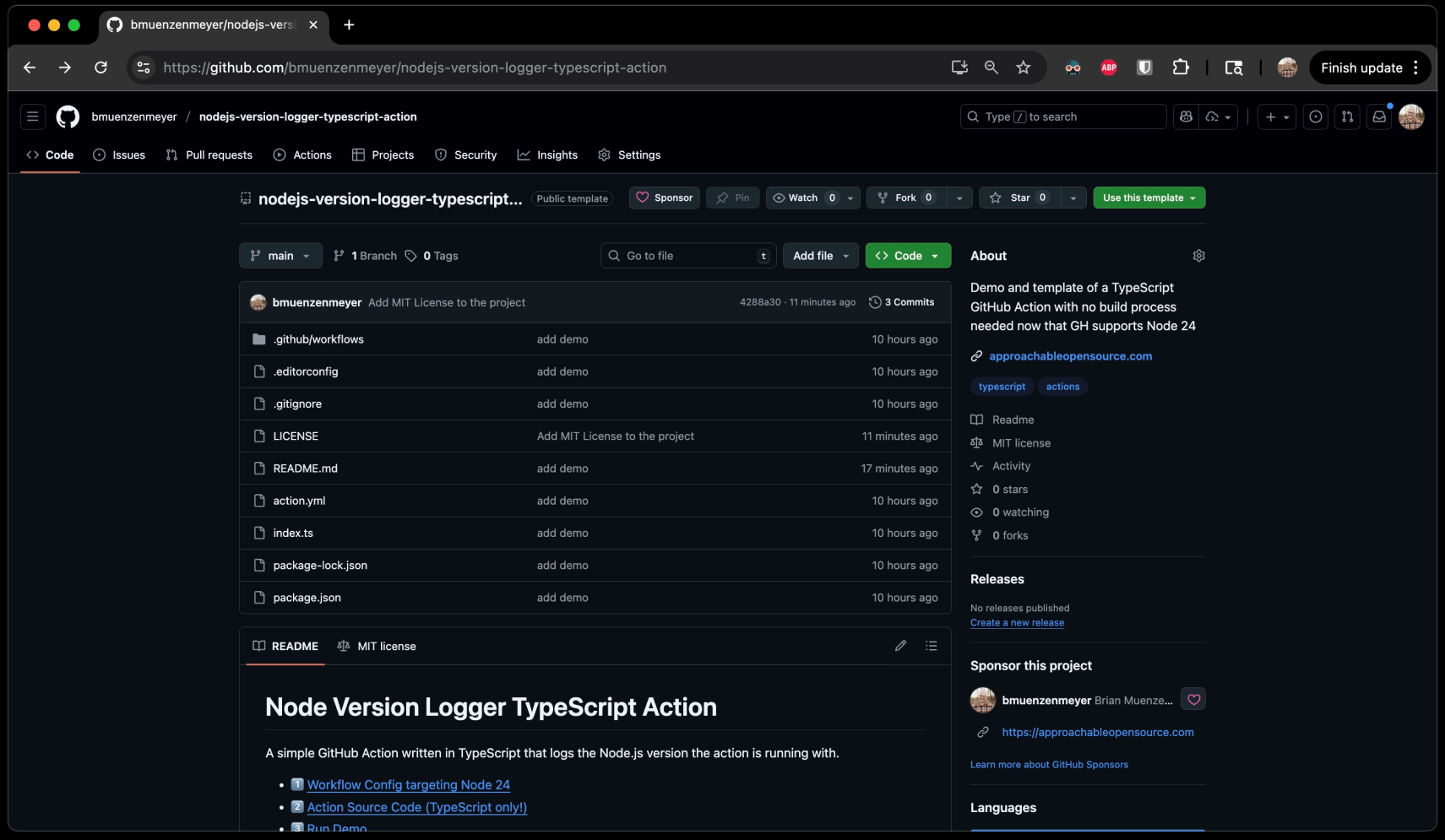
No build process. It might not seem so, but it's a real unlock.
#nodejs #typescript

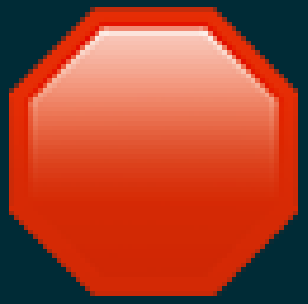
November 13, 2024 at 7:48 PM Everybody can reply

6 likes

2 6

...NOW CONFIRMED


bmuenzenmeyer/nodejs-versi
https://github.com/bmuenzenmeyer/nodejs-version-logger-typescript-action
Finish update
bmuenzenmeyer / nodejs-version-logger-typescript-action
Type / to search
Code
Issues
Pull requests
Actions
Projects
Security
Insights
Settings
nodejs-version-logger-typescript...
Public template
Sponsor
Pin
Watch 0
Fork 0
Star 0
Use this template
main
1 Branch
0 Tags
Go to file
t
Add file
Code
bmuenzenmeyer Add MIT License to the project
4288a30 · 11 minutes ago
3 Commits
.github/workflows add demo 10 hours ago
.editorconfig add demo 10 hours ago
.gitignore add demo 10 hours ago
LICENSE Add MIT License to the project 11 minutes ago
README.md add demo 17 minutes ago
action.yml add demo 10 hours ago
index.ts add demo 10 hours ago
package-lock.json add demo 10 hours ago
package.json add demo 10 hours ago
README
MIT license
Node Version Logger TypeScript Action
A simple GitHub Action written in TypeScript that logs the Node.js version the action is running with.
1 Workflow Config targeting Node 24
2 Action Source Code (TypeScript only!)
3 Run Demo
About
Demo and template of a TypeScript GitHub Action with no build process needed now that GH supports Node 24
approachableopensource.com
typescript
actions
Readme
MIT license
Activity
0 stars
0 watching
0 forks
Releases
No releases published
Create a new release
Sponsor this project
bmuenzenmeyer Brian Muenze...
https://approachableopensource.com
Learn more about GitHub Sponsors
Languages



STOP

Let's not go deeper today.

I'll spare you the ESM + TypeScript + Jest headache.

👉

Okay, well, can we do the same incremental running of our server.ts file with a dependency? Of course we can.

In fact, the ts-node and nodemon docs allude to the fact that this should just work:

```
"dev:nodemon": "nodemon --watch src src/server.js",
```

This was after temporarily dropping the watch glob, as the default is... ∗.∗ 🤝. What we uncover, however, is a problem lurking around the whole post, ESM.

```
> nodemon src/server.ts

[nodemon] 3.1.9
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: ts,json
[nodemon] starting `ts-node src/server.ts`
TypeError: Unknown file extension ".ts" for /workspaces/time-to
    at Object.getFileProtocolModuleFormat [as file:] (node:inte
    at defaultGetFormat (node:internal/modules/esm/get_format:2
    at defaultLoad (node:internal/modules/esm/load:122:22)
    at async ModuleLoader.loadAndTranslate (node:internal/modul
    at async ModuleJob._link (node:internal/modules/esm/module_
  code: 'ERR_UNKNOWN_FILE_EXTENSION'
}
[nodemon] app crashed - waiting for file changes before startin
```

Ugh. Googling around, this is potentially a "famous" problem with ESM + TypeScript. I won't even discuss Jest right now. I did get it working but we shouldn't mention it.

> 🙊 *I'm staying true to this process, so no, we aren't talking about B~~r~~un~~o~~ and Deno. That's not the point, yet. We're almost there, I promise.*

`tsx` I guess is maybe something? This worked:

```
"dev:nodemon": "nodemon --exec pnpm tsx src/server.ts"
```

AFTER

```
{
  "dependencies": {
  },
  "devDependencies": {
    "@types/node": "^24.5.1",
    "typescript": "^5.9.2"
  },
  "scripts": {
    "dev": "node --env-file=.env --watch-path=src src/cli.js",
    "start": "node --env-file=.env src/cli.js",
    "test": "node --test",
    "test:watch": "node --test --watch"
  }
}
```

15 Recent Node.js Features that Replace Popular npm Packages

by: Lizz Parody in Node.js on Oct 01 2025

N|Solid Node.js

SHARE

Over the years, Node.js developers have relied on countless npm packages to fill gaps in the platform. From HTTP utilities to file system helpers, the ecosystem has always been one of Node's greatest strengths. But as Node.js continues to evolve, many features that once required third-party packages are now built into the runtime itself.

This shift reduces dependency bloat, improves security, and makes applications easier to maintain. if you want a tool to track security risks from 3rd party packages in your Node.js check out N|Solid

FEATURED ARTICLES

Node.js 22 Features You Should Be Using
In Node.js on Oct 10 2025

Intelligent Observability: How AI is Transforming Node.js Telemetry into Actionable Optimization
In NodeSource on Oct 09 2025

Modernizing on Your Own Terms: A Strategic Guide to Managing Node.js Legacy Systems
on Sep 10 2025

CATEGORIES

NodeSource
Node.js

COMPARISONS

This ain't your entire app...

COMPARISONS

This ain't your entire app...

...and we can all caveat this with enough asterisks to call in Legal.

BUT NUMBERS ARE NUMBERS

Metric	Before	After	Delta
# node_modules	393	4	1%, or 98 times smaller
size node_modules	75 MB	26 MB	35%, or 2.5 times smaller

TWO ORDERS OF MAGNITUDE LESS DEPENDENCIES.

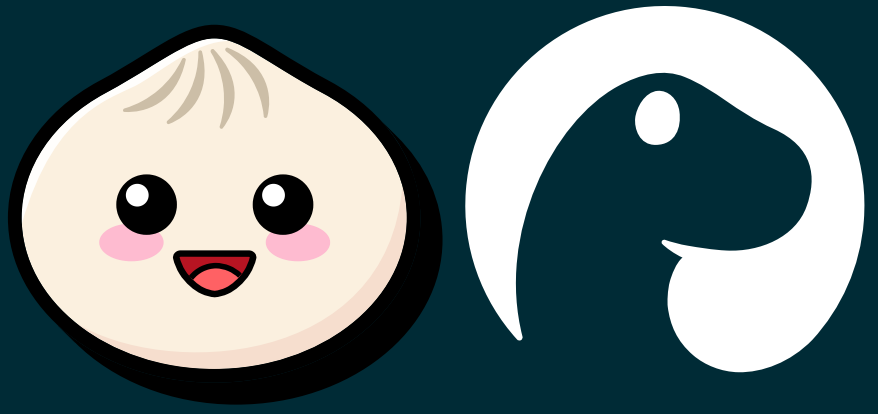
TWO ORDERS OF MAGNITUDE LESS DEPENDENCIES.

DEPENDABOT WILL BE BORED.

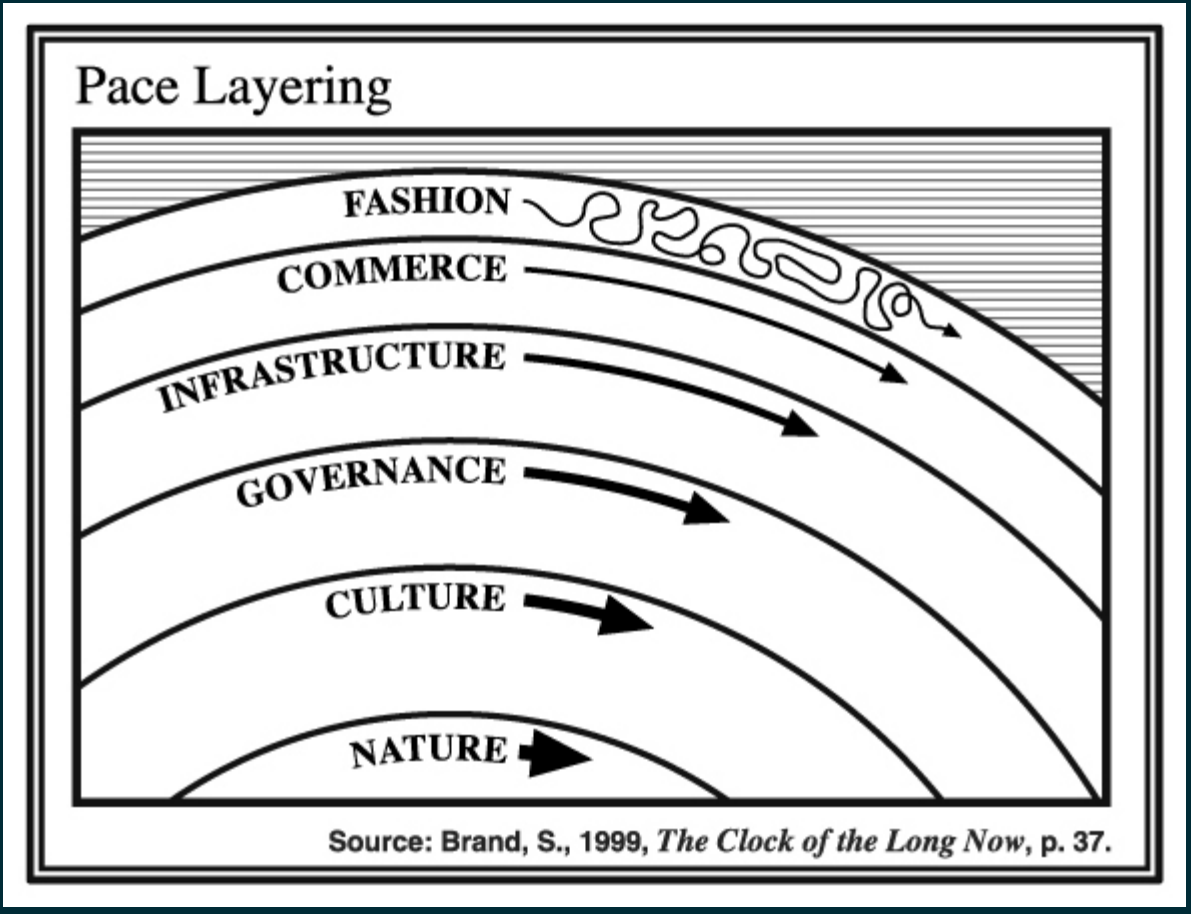
LESS IS ~~MORE~~ LESS

And, these *direct* dependencies, over the past 12 months, have had 35 releases.

~~COMPARISON~~ COMPLEMENT



PACE LAYERS


Pace Layering
FASHION
COMMERCE
INFRASTRUCTURE
GOVERNANCE
CULTURE
NATURE
Source: Brand, S., 1999, The Clock of the Long Now, p. 37.

“FAST LEARNS; SLOW REMEMBERS.”

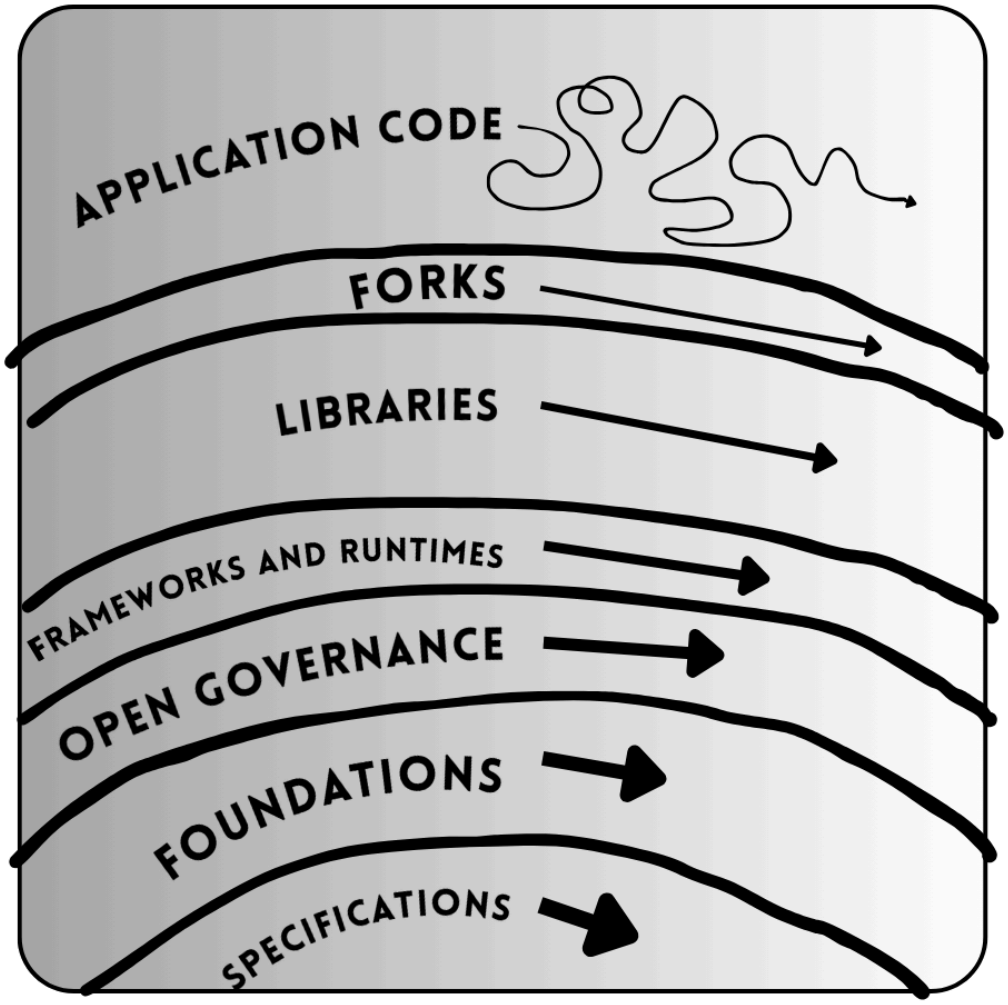
— Stewart Brand

SYM • MATHESY: TOGETHER, LEARN

An entity composed by transcontextual mutual learning through interaction

— Nora Bateson

Open Source Pace Layers, v1.0.0



Source: Brian Muenzenmeyer, 2025, brianmuenzenmeyer.com and approachableopensource.com

CHURN IS PACE LAYERS IN MOTION.

WE CAN CELEBRATE:

WE CAN CELEBRATE:

- innovation can be quick / creators have agency to explore

WE CAN CELEBRATE:

- innovation can be quick / creators have agency to explore
- competition puts pressure on established systems to improve

WE CAN CELEBRATE:

- innovation can be quick / creators have agency to explore
- competition puts pressure on established systems to improve
- maintainers craving momentum and stability have space and time to cultivate

WE CAN CELEBRATE:

- innovation can be quick / creators have agency to explore
- competition puts pressure on established systems to improve
- maintainers craving momentum and stability have space and time to cultivate
- layers exist for anyone to contribute within their means

FIND YOUR LAYER.

There's room for all of us.

THANKS!

@BRIANMUENZENMEYER.COM

@BMUENZENMEYER

@ME

GRAB A COPY NOW