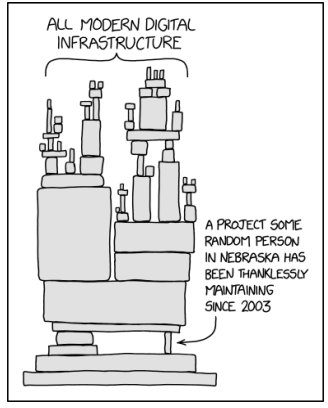
Software Dependencies and Licenses

17-313 Fall 2026

Foundations of Software Engineering

https://cmu-17313q.github.io



Administrivia

- P4B deadline: Tuesday.
- Problem Set 5 is out.
- Quiz 5: next Tuesday.
- Recitation tomorrow, in teams.

Learning Goals

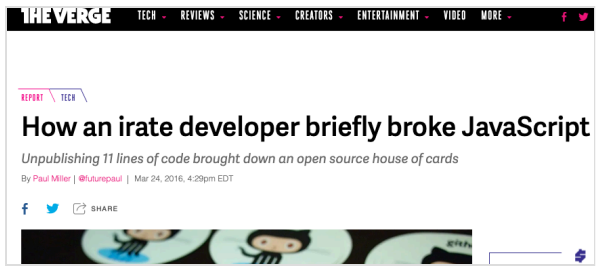
- Explain how direct and transitive dependencies create maintenance risks.
- Compare pinning, floating version constraints, and semantic versioning.
- Describe practical ways to manage updates, security, and compatibility, including build dependencies.
- Explain the main differences between permissive and copyleft licenses.
- Identify license obligations in straightforward reuse, modification, redistribution, and hosting scenarios.

Outline

- **Why Dependencies Matter**
- What Is a Dependency?
- Dependency Graphs
- Trust and Vulnerabilities
- Managing Dependencies Well
- Open Source Licenses
- Applying License Terms

The Left-Pad Incident

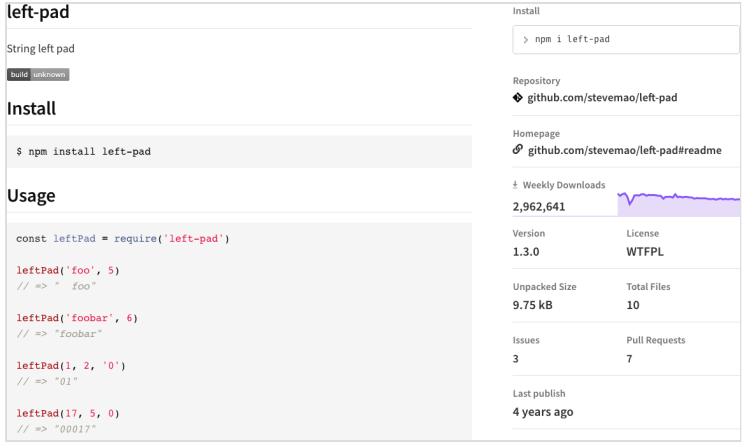
On March 22, 2016, the author of a tiny npm package called **left-pad** unpublished it. Within minutes, builds broke across the JavaScript ecosystem, for projects whose developers had never heard of it.





npm, "kik, left-pad, and npm." blog.npmjs.org/post/141577284765/kik-left-pad-and-npm

What Left-Pad Actually Did



left-pad

String left pad

build unknown

Install

$ npm install left-pad

Usage

```
const leftPad = require('left-pad')

leftPad('foo', 5)
// => "  foo"

leftPad('foobar', 6)
// => "foobar"

leftPad(1, 2, '0')
// => "01"

leftPad(17, 5, 0)
// => "00017"
```

Install

> npm i left-pad

Repository

github.com/stevemao/left-pad

Homepage

github.com/stevemao/left-pad#readme

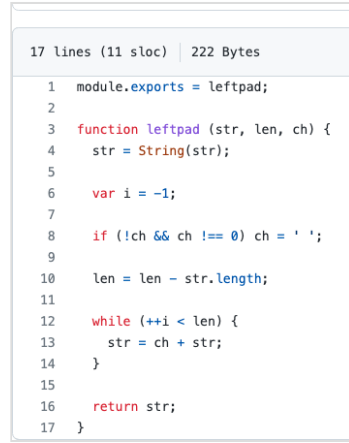
Weekly Downloads

2,962,641

Version	License
1.3.0	WTFPL
Unpacked Size	Total Files
9.75 kB	10
Issues	Pull Requests
3	7

Last publish

4 years ago



17 lines (11 sloc) | 222 Bytes

```
1  module.exports = leftpad;
2
3  function leftpad (str, len, ch) {
4    str = String(str);
5
6    var i = -1;
7
8    if (!ch && ch !== 0) ch = ' ';
9
10   len = len - str.length;
11
12   while (++i < len) {
13     str = ch + str;
14   }
15
16   return str;
17 }
```

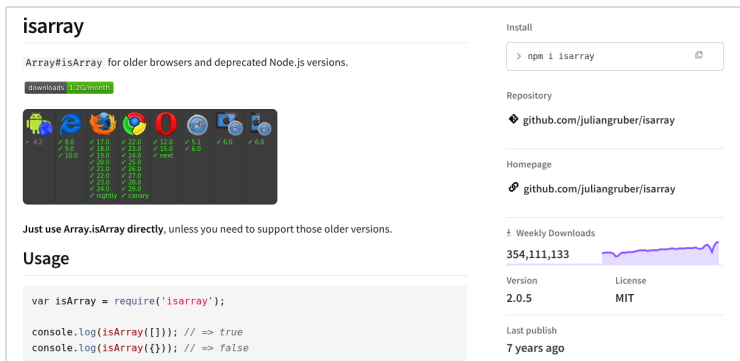
How could removing this small package break projects whose developers never chose to use it?

NOTES

left-pad padded a string to a given length with a given character. Seventeen lines of code. The amount of code does not determine the size of the disruption its disappearance can cause.

See Also: isarray

Even smaller than left-pad: **isarray** wraps `Array.isArray` for ancient browsers and deprecated Node.js versions. Five lines of code, over 350 million downloads a week.



5 lines (4 sloc) | 133 Bytes

```
1  var toString = {}.toString;
2
3  module.exports = Array.isArray || function (arr) {
4    return toString.call(arr) === '[object Array]';
5  };
```

A dependency doesn't need to be clever or substantial to become load-bearing infrastructure.

Outline

- Why Dependencies Matter
- **What Is a Dependency?**
- Dependency Graphs
- Trust and Vulnerabilities
- Managing Dependencies Well
- Open Source Licenses
- Applying License Terms

What Is a Dependency?

- A **dependency** is anything your code needs to build, run, or be tested: another module, library, or service.
- Build systems mainly track code dependencies, but runtime dependencies matter just as much.
- If a dependency is unavailable or incompatible, your system cannot function correctly, the same way a recipe fails with a missing ingredient or a broken tool.

Where Are Dependencies Hosted?

- Public repositories: **PyPI** (Python), **npm** (JavaScript), **Maven Central** (Java).
- Packages need a unique identifier, typically a name and version.
- Custom or private repositories are common too, often for internal packages.





Somebody has to run that repository: availability, access control, integrity, uniqueness, and traceability all become somebody's job.

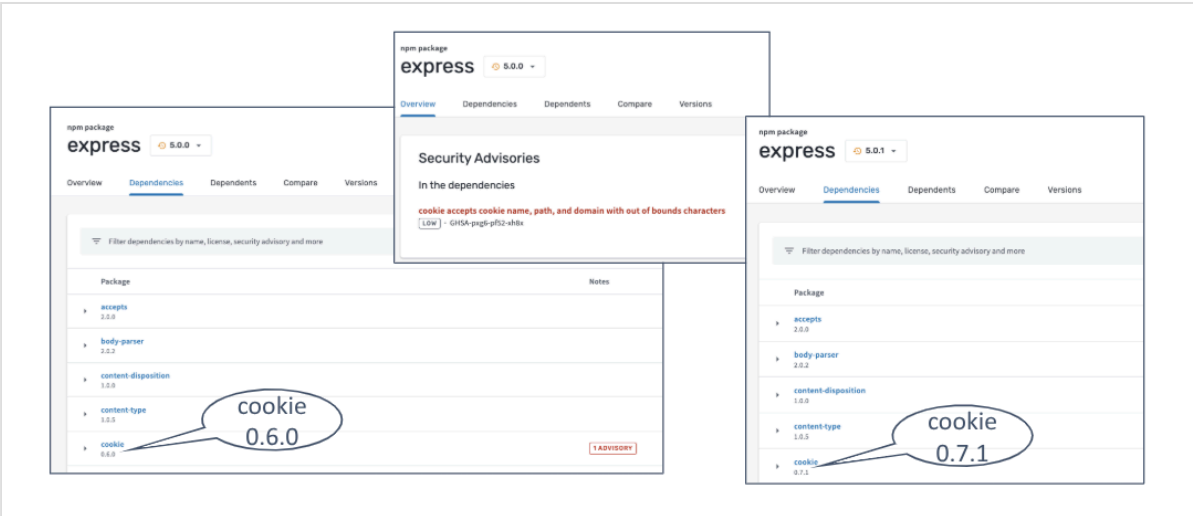
Pinning vs. Floating

- **Pinning:** "I depend on libFoo 1.5.0." Declares an exact version, frozen in time.
- **Floating:** "I depend on libFoo-latest" (or a range like `1.5.x`). Each build can resolve to a newer version.

NOTES

Controlling changes improves stability, but taking fixes requires updates. Allowing changes can deliver fixes sooner, but can also introduce new bugs.

Pinned Dependencies Need Explicit Updates



express 5.0.0 pinned a vulnerable cookie; 5.0.1 updated to the fixed version.

Updates like this can be automated, but someone still has to trigger and review them.

NOTES

Lockfiles Freeze the Resolved Tree

- An application declares `libFoo: "1.x"` (floating). Its lockfile records the version actually resolved: `1.5.0`.
- A newer `1.5.1` is published. The locked CI install still uses `1.5.0`, because it installs from the lockfile, not by re-resolving the range.
- To take the update: a pull request changes the lockfile and runs tests, the same as any other code change.

If we pin libFoo, what else might still change?

NOTES

A pinned direct dependency does not freeze its own transitive dependencies, the build tools, or the environment around it.

Pinning vs. Floating: Tradeoffs

Pinning (e.g. 1.5.3)

- Stable, selected versions, with the full dependency tree locked.
- Can stay vulnerable until someone updates it.
- Dependents must keep updating as dependencies evolve.

Floating (e.g. 1.x)

- Builds can break from changes you didn't choose.
- Can take newer fixes within the allowed range when dependencies are resolved again.
- Less manual maintenance.
- A float leaks transitively: if A pins B, and B floats C, A still sees a changing C.

Semantic Versioning

- **MAJOR:** incompatible changes to the declared public API.
- **MINOR:** backward-compatible functionality added.
- **PATCH:** backward-compatible bug fixes.

Starting from `1.5.0`: what would `1.5.1`, `1.6.0`, and `2.0.0` each communicate?



Consumers notice when a promised version contract breaks.

NOTES

SemVer's rules concern the declared public API; they are not a guarantee about binary linking. Versions in the 0.y.z range don't promise a stable public API at all. Version numbers communicate compatibility expectations, but consumers still need tests when upgrading. The screenshot is a real project (Celery) whose own users called out a SemVer violation.

semver.org

Quick Check: Classify the Change

Assuming a stable SemVer public API, is each change a PATCH, MINOR, or MAJOR?

- Removing a public method.
- Fixing a bug that didn't change any public method's behavior as documented.
- Adding a new public method.

PATCH

Backward-compatible bug fix.

MINOR

Backward-compatible new feature.

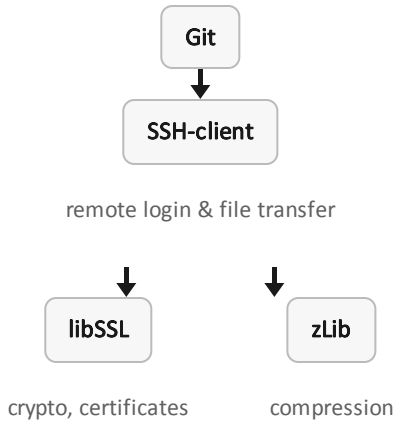
MAJOR

Breaking change.

Outline

- Why Dependencies Matter
- What Is a Dependency?
- **Dependency Graphs**
- Trust and Vulnerabilities
- Managing Dependencies Well
- Open Source Licenses
- Applying License Terms

Transitive Dependencies



Should Git use exports of libSSL or zLib directly, even though it only needs them through SSH-client?

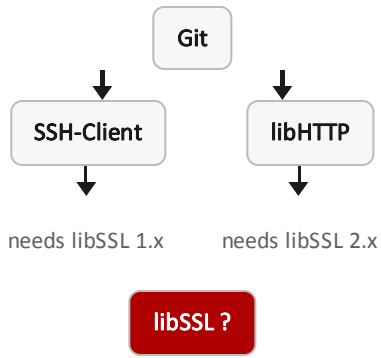
If Git relies on something from libSSL directly (e.g. certificate management), declare that dependency explicitly: an SSH-client update could otherwise remove it without warning.

NOTES

Packages can depend on other packages, and your build pulls in all of them transitively. A dependency you never directly chose, like libSSL or zLib here, can still break your build if you quietly relied on something it happened to bring along.

Diamond Dependencies

What happens when multiple intermediate dependencies need the same transitive dependency, at different versions?



Picking one compatible version may be impossible. Some ecosystems allow multiple versions side by side, which solves this but can create other integration problems.

Outline

- Why Dependencies Matter
- What Is a Dependency?
- Dependency Graphs
- **Trust and Vulnerabilities**
- Managing Dependencies Well
- Open Source Licenses
- Applying License Terms

Would You Eat This Cookie?



NOTES

Just as you wouldn't eat a cookie you found on the sidewalk, installing a package means running code you haven't reviewed, written by people you don't know, with the same privileges as the rest of your system.

Would You Insert a Found Drive Into Your Laptop?



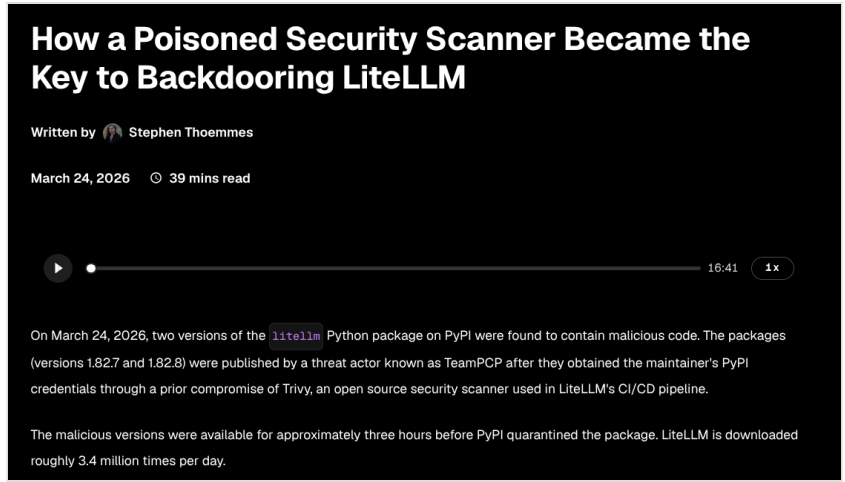
What makes us willing to run code from a package registry?

NOTES

A package install runs with the same privileges as the rest of your system, same as that drive would if plugged in.

Watch Out for Build Dependencies

- Build tools, test tools, and security scanners execute code on your machine or in CI.
- They can access credentials and files available to that environment.
- A compromised build dependency can expose publishing credentials and put your users at risk.



How a Poisoned Security Scanner Became the Key to Backdooring LiteLLM

Written by Stephen Thoemmes

March 24, 2026 39 mins read

16:41 1x

On March 24, 2026, two versions of the `litellm` Python package on PyPI were found to contain malicious code. The packages (versions 1.82.7 and 1.82.8) were published by a threat actor known as TeamPCP after they obtained the maintainer's PyPI credentials through a prior compromise of Trivy, an open source security scanner used in LiteLLM's CI/CD pipeline.

The malicious versions were available for approximately three hours before PyPI quarantined the package. LiteLLM is downloaded roughly 3.4 million times per day.

Example, LiteLLM, March 2026: a compromised security scanner reportedly exposed publishing credentials. Attackers then published malicious LiteLLM packages.

Outline

- Why Dependencies Matter
- What Is a Dependency?
- Dependency Graphs
- Trust and Vulnerabilities
- **Managing Dependencies Well**
- Open Source Licenses
- Applying License Terms

How to Manage Dependencies Well

- Regularly monitor and update dependencies.
  - Subscribe to project security advisories.
  - Update through a pull request: inspect, test, deploy deliberately.
- Have a vulnerability response plan. Don't wait for disasters.
- Tools you could use:
  - Dependabot (GitHub's automated checks for outdated packages)
  - Snyk (open source vulnerability scanning)
  - OWASP Dependency-Check
- Track the licenses of all third-party code you use.
- Maintain a Software Bill of Materials (SBOM) listing all dependencies.

NOTES

Open-source components are powerful, but managing them carefully is essential for security, compliance, and stability.

Good dependency management protects your project and speeds up your team's work.

Hyrum’s Law

"With a sufficient number of users of an API, it does not matter what you promise in the contract: all observable behaviors of your system will be depended on by somebody."

Hyrum Wright

Can an update preserve the documented API and still break a consumer?

Yes: a consumer may rely on behavior the contract never promised.

NOTES

This is an experience-based observation, not a formally proven universal law; Wright's own site calls it an observation. Go 1.1 and 1.2 happened to produce consistent iteration order for small maps, with no specification guarantee. Developers relied on it anyway. Go 1.3 randomized that order, and its release notes explicitly warned that code relying on the old behavior would break.

Part II: Open Source and Licenses

Outline

- Why Dependencies Matter
- What Is a Dependency?
- Dependency Graphs
- Trust and Vulnerabilities
- Managing Dependencies Well
- **Open Source Licenses**
- Applying License Terms

Why Learn About Licenses?

- Open-source software is released under an **open source license**.
- The Open Source Definition (OSD) describes ten traits a license must support, maintained by the Open Source Initiative (OSI).
- Around 80 OSI-approved licenses exist, but only a handful are commonly used: MIT, Apache-2.0, GPL, BSD, and a few others.

Every dependency you consume, and everything you release, carries one of these licenses. Not knowing the terms is a legal risk.

opensource.org/osd

From Dependencies to Licenses

So far, we have asked whether a dependency works and whether we can trust it. We also need to understand what its license permits and requires.

Open-source software is made available under terms that permit use, modification, and redistribution, with access to the source.

- **Access:** Can we inspect the source?
- **Permission:** Can we copy, modify, and distribute it?
- **Conditions:** What must accompany those actions?

A public GitHub repository with no license: viewing and forking are allowed by the platform, but that alone isn't permission to reuse the code.

NOTES

The Open Source Definition includes further requirements, including nondiscrimination against people and fields of use.

docs.github.com (licensing a repository)

Common Open Source Licenses

License	Type	Key traits
GPL	Copyleft	Modifications must be shared, re-licensed under the same terms, and source made available.
LGPL	Weak copyleft	Like GPL, but for libraries: a program that merely links to it doesn't inherit the copyleft.
MIT	Permissive	Retain the copyright notice. No requirement that derivative works stay open source.
BSD	Permissive	Retain the copyright notice. The original author's name can't be used to endorse derived products without permission.
Apache-2.0	Permissive	Similar to MIT, plus an explicit grant of patent rights from contributors to users.

NOTES

MIT, BSD, and Apache-2.0 are all permissive, but differ in specifics: BSD adds a non-endorsement clause, and Apache-2.0 adds an explicit patent grant. LGPL exists specifically so a library can stay copyleft without forcing copyleft onto

Outline

- Why Dependencies Matter
- What Is a Dependency?
- Dependency Graphs
- Trust and Vulnerabilities
- Managing Dependencies Well
- Open Source Licenses
- **Applying License Terms**

What Are You Planning to Do?

Different actions trigger different license obligations. Identify the action first, then check the relevant terms.

- Use a dependency internally.
- Modify its code.
- Distribute source, binaries, or a container image to others.
- Run a modified service for remote users.

Record and Fulfill the Obligations

1. Check the license for the actual version being adopted, including relevant dependencies and bundled files.
2. Read the license file and any NOTICE or third-party notices.
3. Record required notices, source availability, and modification disclosures.
4. Include those materials in the release process, and revisit them when dependencies change.

Activity: Choose the Appropriate License

In small groups: for each scenario, choose from GPL, LGPL, MIT, BSD, or Apache-2.0 and justify your choice. Consider copyleft vs. permissive, proprietary/redistribution concerns, and patent protection. You can consult choosealicense.com.

#	Scenario
1	You are creating a software tool and want others to freely use, modify, and even sell it without many restrictions. What type of open-source license would you choose?
2	You want to release your software as open source but also want to protect yourself from future patent lawsuits. What kind of license or clause should you include?
3	You want your open-source project to spread widely into both open-source and commercial products but still want to get credit for your work. How would you license it?
4	You are building open-source software and want to make sure that if someone improves it and shares it, they also have to share their improvements openly with everyone. What kind of license would you choose?
5	You want people to freely use and modify your software, but don't want them to use your name or your company's name to promote their versions without your permission. How would you set up your license?
6	You are combining open-source components with your own proprietary modules. How can you make sure you stay legally safe while keeping parts of your code private?

1 min instructions · 6 min in groups · 2 min responses

Debrief: Instructor Answer Key

#	Scenario	License	Why
1	Freely use, modify, and sell, with minimal obligations.	**MIT / BSD**	Maximum freedom, minimal restrictions, including commercial resale.
2	Open source, with explicit patent rights granted.	**Apache-2.0**	Its explicit patent grant is the distinguishing feature here.
3	Wide adoption, no copyleft, but keep credit.	**MIT / Apache-2.0**	Permissive, so commercial products can adopt it; copyright notice preserves credit.
4	Modifications must also be open-sourced.	**GPL**	Strong copyleft: modifications must be shared under the same terms.
5	Max adoption, but no unauthorized use of your name.	**BSD**	Its non-endorsement clause exists for exactly this: no use of the author's name to promote derivatives.
6	Combine OSS and proprietary code safely.	**LGPL**	Lets proprietary code link against an open-source library without the library's copyleft spreading to it.

Each scenario targets one license's distinguishing feature -- the point isn't memorizing clauses, but recognizing which feature a given goal calls for.

Takeaways

- A small, invisible dependency can still bring down systems that never chose to use it directly.
- Pinning trades agility for stability; floating trades stability for agility. Lockfiles and SemVer are how teams negotiate that tradeoff.
- Trusting a dependency means trusting code you haven't reviewed, including the build and test tools that run alongside it.
- A technically compatible update can still break a consumer who relied on behavior the contract never promised.
- A dependency's license is part of adopting it, not an afterthought: identify what you're doing with the code, then check what that action requires.