

Open Source

17-313: Foundations of Software Engineering

<https://cmu-313.github.io>

Michael Hilton and Chris Timperley

Fall 2025

Administrivia

- Teamwork survey deployed today. We will have a survey after the lecture as well
- Guest Lecture next Tuesday – PLEASE ARRIVE ON TIME!
- Project 4 Checkpoint 1 extended to Friday
- NOTE: You must be present DURING THE PARTICIPATION activity for participation points

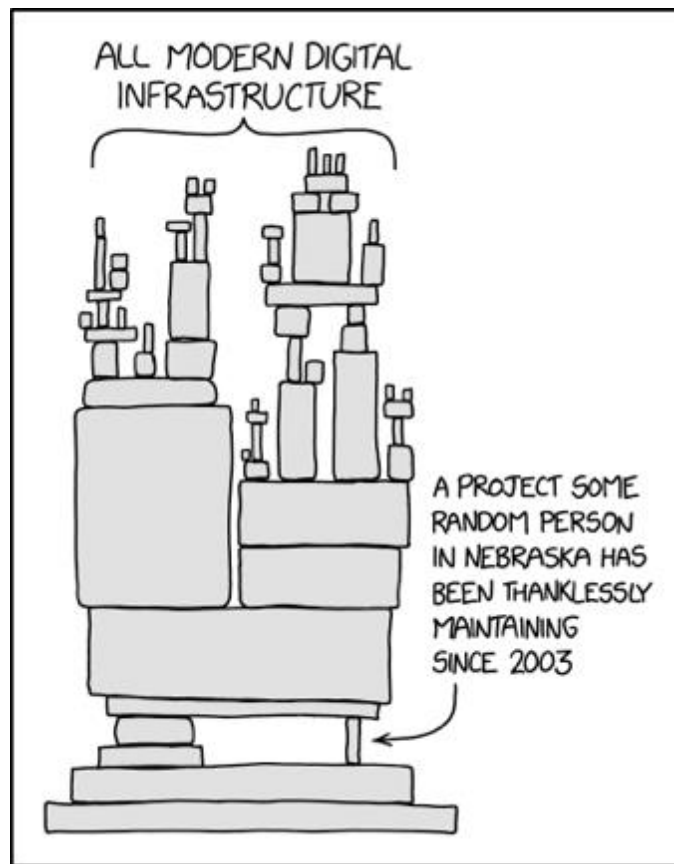
Learning Goals

- Distinguish between open-source software, free software, and commercial software.
- Identify the common types of software licenses and their implications.
- Distinguish between copyright and intellectual property.
- Express an educated opinion on the philosophical/political debate between open source and proprietary principles.
- Describe how open-source ecosystems work and evolve, in terms of maintainers, community contribution, and commercial backing
- Identify various concerns of commercial entities in leveraging open-source, as well as strategies to mitigate these.

Open Source

Background: laws and open source

- Copyright protects creative, intellectual and artistic works — including software
- Alternative: public domain (nobody may claim exclusive property rights)
- Trademark protects the name and logo of a product
- OSS is generally copyrighted, with copyright retained by contributors or assigned to entity that maintains it
- Copyright holder can grant a license for use, placing restrictions on how it can be used (perhaps for a fee)



What is Open-Source Software?

Open-source



Proprietary



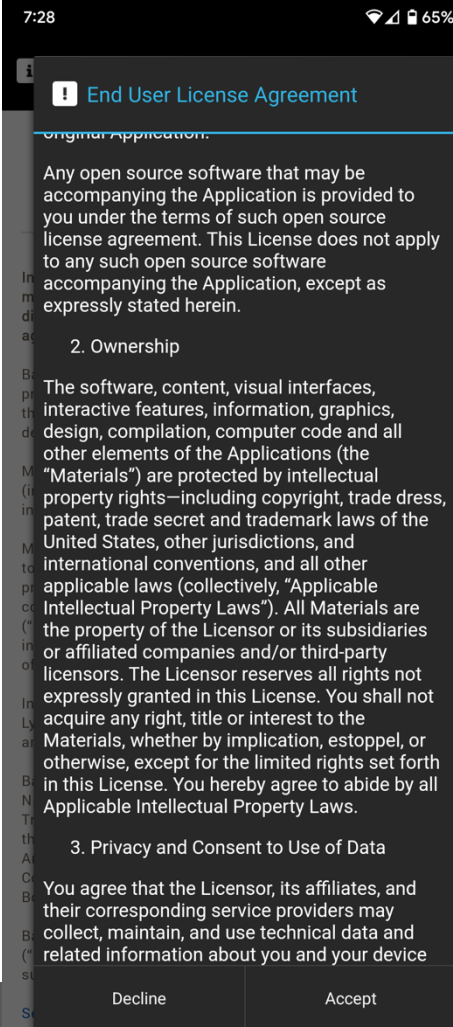
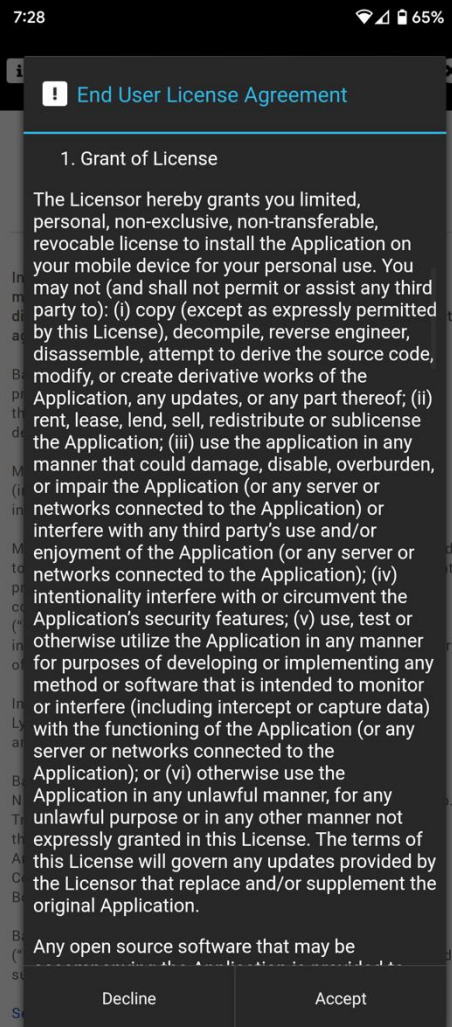
What is Open-Source Software (OSS)?

- Source code availability
- Right to modify and create derivative works
- (Often) Right to redistribute derivative works

Contrast with proprietary software: a black box

- Intention is to be used, not examined, inspected, or modified.
- No source code – only download a binary (e.g., an app) or use via the internet (e.g., a web service).
- Often contains an End User License Agreement (EULA) governing rights and liabilities.
- EULAs may specifically prohibit attempts to understand application internals.

Example: Bank
app on my phone



Why Go Open Source (vs. Proprietary) ?

Advantages

Disadvantages

- <today's activity; do in groups>

Why Go Open Source (vs. Proprietary) ?

Advantages

- Transparency, gain user trust
- Many eyes: crowd-source bug reports and fixes
- Security: more likely for vulnerabilities to be quickly identified
- Community and adoption: get others to contribute features, build stuff around you, or fork your project

Disadvantages

- Reveal implementation secrets
- Many eyes: users can find faults more easily
- Security: more likely for others to find vulnerabilities first
- Control: You may not be able to influence the long-term direction of your platform

Early open source: UNIX to BSD

- Hardware was not yet standardized, computer vendors focused on hardware, building new operating systems for each platform
- Much software development focused in academic labs, and AT&T's Bell Labs
- Unix created at Bell Labs using the new, portable language "C", licenses initially released with source code
- 1978: UC Berkeley begins distributing their own derived version of Unix (BSD)
- AT&T is prohibited from entering new telecommunications businesses (can't make OS a product)



IBM 704 at NASA Langley in 1957 (Public domain)

The BSD License is Permissive

- Authors of BSD created a license for the OS that:
 1. Required those using it to credit the university
 2. Limited liability for (mis)-use

```
Copyright (c) <year>, <copyright holder> All rights reserved.  
Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:  
1.Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.  
2.Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.  
3.All advertising materials mentioning features or use of this software must display the following acknowledgement: This product includes software developed by the <copyright holder> .  
4.Neither the name of the <copyright holder> nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.  
THIS SOFTWARE IS PROVIDED BY <COPYRIGHT HOLDER> AS IS AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND  
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.... (move waivers of liability)
```

Original BSD license

```
calling ipso_policy_init for quarantine  
Security policy loaded: Quarantine policy (Quarantine)  
Copyright (c) 1982, 1986, 1989, 1991, 1993  
The Regents of the University of California. All rights reserved.  
  
MRC Framework successfully initialized  
using 16384 buffer headers and 10240 cluster IO buffer headers  
AppleKeyStore starting (BUILT: Sep 19 2014 00:11:30)
```

BSD Copyright in OSX boot sequence

UNIX to GNU's Not Unix

- Timeline

- 1978: UC Berkeley begins distributing their own derived version of Unix (BSD)
- 1983: AT&T broken up by DOJ, UNIX licensing changed: no more source releases
- Competing commercial vendors all package and sell their derivations of UNIX (AT&T, HP, Sun, IBM, SGI)
- Also 1983: "Starting this Thanksgiving I am going to write a complete Unix-compatible software system called GNU (Gnu's Not Unix), and give it away free to everyone who can use it"



GNU logo (a gnu wildebeest)

Free software as a Philosophy

- “Free as in Speech, not as in beer”

Richard Stallman's Free Software Foundation — free as in liberties

- Freedom 0: run code as you wish, for any purpose
- Freedom 1: study how code works, and change it as you wish
- Freedom 2: redistributed copies (of original) so you can help others
- Freedom 3: distribute copies of your modified version to others



Richard M. Stallman (Licensed under GFDL)

Free software as a Philosophy

- “Free as in Speech, not as in beer”

FSF: software licensed under GNU Public License (GPL), considering questions like:

- Required to redistribute modifications (under same license)? Yes, “copyleft”
- Can you combine it with more restrictive licenses? No, not even with BSD!

Alternative (more like BSD):

“Do whatever you want with this software, but don’t blame me if it doesn’t work” freeware

Copyleft v. permissive

- Can I **combine** OSS with my product, releasing my product under a different license (perhaps not even OS)?
- **Permissive licenses** encourage adoption by permitting this practice
- **Copyleft** “protects the commons” by having all linked code under same license, transitively requiring more sharing
- Philosophy: do we force participation, or try to grow/incentivize it in other ways?

GNU/Linux (1991-Today)

- Stallman set out to build an operating system in 1983, ended up building utilities needed by an operating system (compiler, etc)
- Linux is built around and with the GNU utilities, licensed under GPL
- Rise of the internet, demand for internet servers drives demand for cheap/free OS
- Companies adopted and support Linux for enterprise customers
- IBM committed over \$1B; Red Hat and others



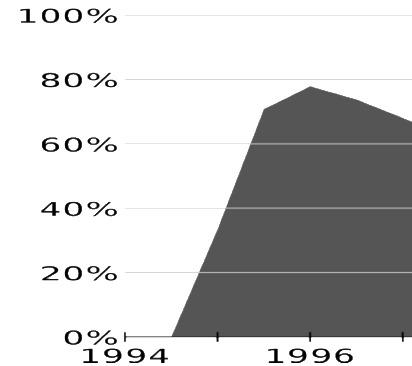
Free Software vs. Open Source

- Free software origins (70-80s ~Stallman)
 - ~~Cultish~~ Political goal
 - Software part of free speech
 - free exchange, free modification
 - proprietary software is unethical
 - security, trust
 - GNU project, Linux, GPL license
- Open source (1998 ~O'Reilly)
 - Rebranding without political legacy
 - Emphasis on internet and large dev/user involvement
 - Openness toward proprietary software/coexist
 - (Think: Netscape becoming Mozilla)



Netscape's open source gambit

- Netscape was dominant web browser early 90's
- Business model: free for home and education use, companies pay
- Microsoft entered browser market with Internet Explorer, bundled with Windows95, soon overtakes Netscape in usage (free with Windows)
- January 1998: Netscape first company to open source code for proprietary product (Mozilla)

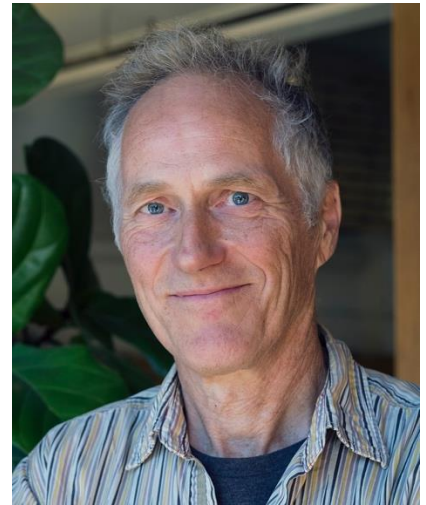


Netscape creates a new license and model

- Until Netscape, much of OSS was the FSF and its GPL
- **Open Source** coined in 1998 by the Open Source Initiative to capture Netscape's aim for an open development process
- New licenses follow, e.g. MIT, Apache, etc. just like BSD, but without the advertising part
- Publisher Tim O'Reilly organizes a **Freeware Summit** later in 1998, soon rebranded as **Open Source Summit**
- Open Source is a development methodology; free software is a social movement
— Richard Stallman



Open source initiative logo



Tim O'Reilly
Photo via Christopher Michel/Flickr, CC BY 2.0



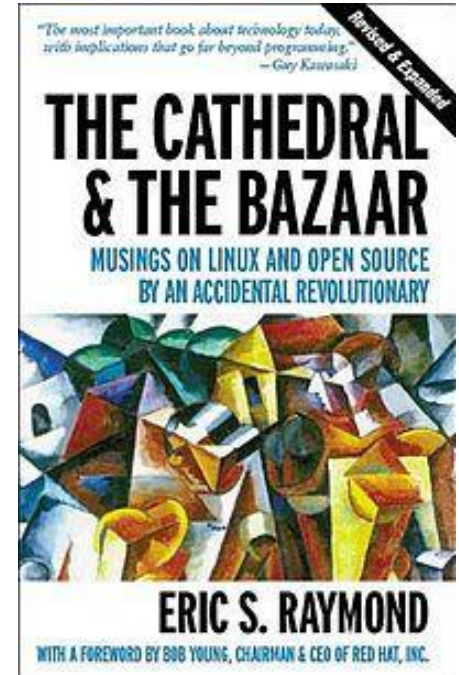
Perception (from some):

- Anarchy
- Demagoguery
- Ideology
- Altruism

Open-Source Ecosystems

How OSS is developed

The Cathedral and the Bazaar



The Bazaar won

Cathedral

- Developed centrally by a core group of members
- Available for all once complete (or at releases)
- Examples: GNU Emacs, GCC (back in the 1990s)
- “Sort-of” examples today: Chrome, IntelliJ

Bazaar

- Developed openly and organically
- Wide participation (in theory, anyone can contribute)
- Examples: Linux

OSS has many stakeholders / contributors

- Core members
 - Often (but not always) includes the original creators
 - Direct push access to main repository
 - May be further split into admin roles and developers
- External contributors
 - File bug reports and report other issues
 - Contribute code and documentation via pull requests
- Other supporters
 - Beta testers (users)
 - Sponsors (financial or platform)
 - Steering committees or public commenters (for standards and RFCs)
- Spin-offs
 - Maintainers of forks of the original repository

Contributing processes

- Mature OSS projects often have strict contribution guidelines
 - Look for CONTRIBUTING.md or similar
- Common requirements:
 - Coding style (recall: linters) and passing static checks
 - Inclusion of test cases with new code
 - Minimum number of code reviews from core devs
 - Standards for documentation
 - Contributing licensing agreements (more on that later)

Governance

- Some OSS projects are managed by for-profit firms
 - Examples: Chromium (Google), Moby (Docker), Ubuntu (Canonical), TensorFlow (Google), PyTorch (Meta), Java (Oracle)
 - Contributors may be a mix of employees and community volunteers
 - Corporations often fund platforms (websites, test servers, deployments, repository hosting, etc.)
 - Corporations usually control long-term vision and feature roadmap
- Many OSS projects are managed by non-profit foundations or ad-hoc communities
 - Examples: Apache Hadoop/Spark/Hbase/Kafka/Tomcat (ASF), Firefox (Mozilla), Python (PSF), NumPy (community)
 - Foundations fund project infrastructure via charitable donations
 - Long-term vision often developed via a collaborative process (e.g., Apache) or by benevolent dictators (e.g., Python, Linux)
- Corporations still heavily rely on community-owned OSS projects
 - Many OSS non-profits are funded by Big Tech (e.g., Mozilla by Google)

Example: Apache

WHAT MAKES THE APACHE WAY SO HARD TO DEFINE?

The Apache Way is a living, breathing interpretation of one's experience with our community-led development process. Apache is unique, diverse, and focused on the activities needed at a particular stage of the project's lifetime, including nurturing community building awareness. What is important is that they embrace:

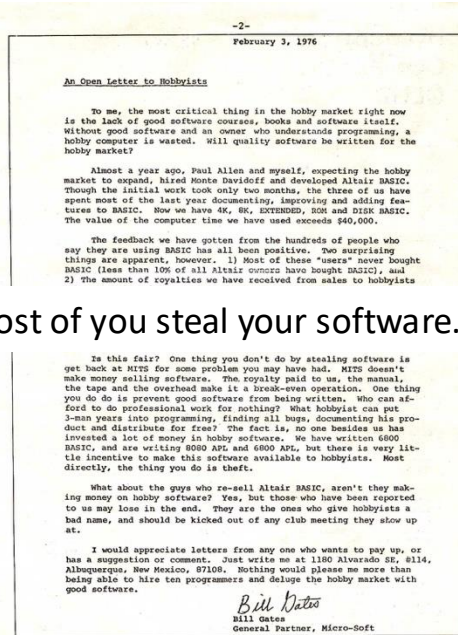
- **Earned Authority:** all individuals are given the opportunity to participate, but their influence is based on publicly earned community. Merit lies with the individual, does not expire, is not influenced by employment status or employer, and is not project cannot be applied to another). [More on merit.](#)
- **Community of Peers:** individuals participate at the ASF, not organizations. The ASF's flat structure dictates that roles are equal weight, and contributions are made on a volunteer basis (even if paid to work on Apache code). The Apache community with respect in adherence to our [Code of Conduct](#). Domain expertise is appreciated; Benevolent Dictators For Life are [discouraged from participation.](#)
- **Open Communications:** as a virtual organization, the ASF requires all communications related to code and decision-making to be asynchronous collaboration, as necessitated by a globally-distributed community. Project mailing lists are archived, public, and searchable.
 - dev@ (primary project development)
 - user@ (user community discussion and peer support)
 - commits@ (automated source change notifications)
 - occasionally supporting roles such as marketing@ (project visibility)

...as well as restricted, day-to-day operational lists for Project Management Committees. Private decisions on code, policies, or discourse and transactions must be brought on-list. [More on communications](#) and the [use of mailing lists](#).

- **Consensus Decision Making:** Apache Projects are overseen by a self-selected team of active volunteers who are contributors. Projects are auto-governing with a heavy slant towards driving consensus to maintain momentum and productivity. When necessary, establish at all times, holding a vote or other coordination may be required to help remove any blocks with binding decisions. [More on decision making and voting.](#)
- **Responsible Oversight:** The ASF governance model is based on trust and delegated oversight. Rather than detailed rule-based governance is principles-based, with self-governing projects providing reports directly to the Board. Apache Committers are reviewed and approved, employing mandatory security measures, ensuring license compliance, and protecting the Apache community from abuse. [More on responsibility.](#)



Corporate outlook towards open-source has evolved over the years



“...most of you steal your software...”

Redmond top man Satya Nadella: 'Microsoft LOVES Linux'

Open-source 'love' fairly runneth over at cloud event



Risks in not open-sourcing?

MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat

jeff@google.com, sanjay@google.com

Google, Inc.

Abstract

MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a *map* function that processes a key/value pair to generate a set of intermediate key/value pairs, and a *reduce* function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model, as shown in the paper.

Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. The run-time system takes care of the details of partitioning the input data, scheduling the program's execution across a set of machines, handling ma-

given day, etc. Most such computations are conceptually straightforward. However, the input data is usually large and the computations have to be distributed across hundreds or thousands of machines in order to finish in a reasonable amount of time. The issues of how to parallelize the computation, distribute the data, and handle failures conspire to obscure the original simple computation with large amounts of complex code to deal with these issues.

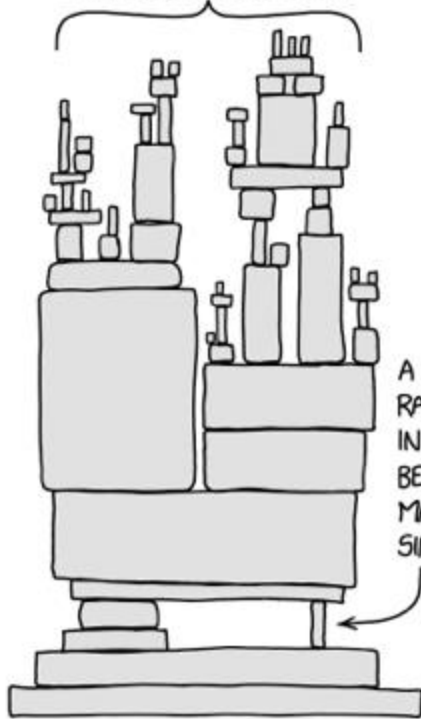
As a reaction to this complexity, we designed a new abstraction that allows us to express the simple computations we were trying to perform but hides the messy details of parallelization, fault-tolerance, data distribution and load balancing in a library. Our abstraction is inspired by the *map* and *reduce* primitives present in Lisp



Use of open source software within companies

- Is the license compatible with our intended use?
 - More on this later
- How will we handle versioning and updates?
 - Does every internal project declare its own versioned dependency or do we all agree on using one fixed (e.g., latest) version?
 - Sometimes resolved by assigning internal “owners” of a third-party dependency, who are responsible for testing updates and declaring allowable versions.
- How to handle customization of the OSS software?
 - Internal forks are useful but hard to sync with upstream changes.
 - One option: Assign an internal owner who keeps internal fork up-to-date with upstream.
 - Another option: Contribute all customizations back to upstream to maintain clean dependencies.
- Security risks? Supply chain attacks on the rise.

ALL MODERN DIGITAL INFRASTRUCTURE



A PROJECT SOME
RANDOM PERSON
IN NEBRASKA HAS
BEEN THANKLESSLY
MAINTAINING
SINCE 2003

QUARTZ

Make business better.™



Send us a Tip!

MEMBERSHIP



HOME

LATEST

BUSINESS NEWS

MONEY & MARKETS

TECH & INNOVATION

LIFESTYLE

LEADERSHIP

EMAILS

PODCASTS

EN ESPAÑOL

We may earn a commission from links on this page.

TECH & INNOVATION

NPM ERROR

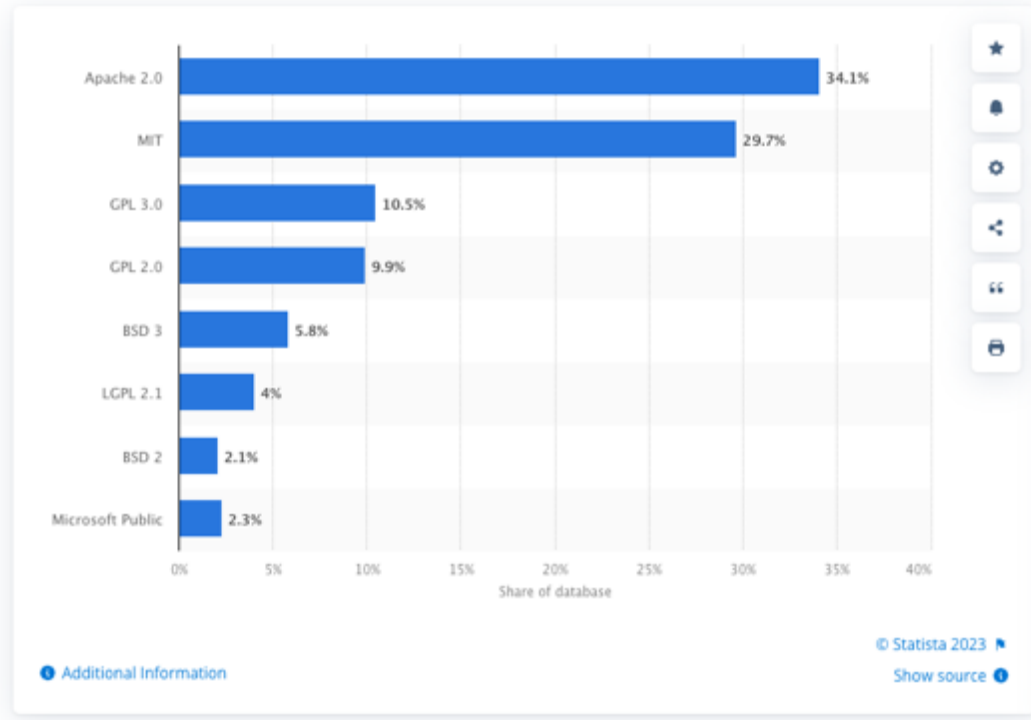
How one programmer broke the internet by deleting a tiny piece of code

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

Software Licenses

Note: I am not a lawyer (this is not legal advice)

Most popular open source licenses worldwide in 2021




Which license to choose?

choosealicense.com

Choose an open source license

An open source license protects contributors and users. Businesses and savvy developers won't touch a project without this protection.


Which of the following best describes your situation?



I need to work in a community.

Use the license preferred by the community you're contributing to or depending on. Your project will fit right in.


If you have a dependency that doesn't have a license, ask its maintainers to add a license.



I want it simple and permissive.

The MIT License is short and to the point. It lets people do almost anything they want with your project, like making and distributing closed source versions.

React, MET, and Rails use the MIT License.



I care about sharing improvements.

The GNU GPLv3 also lets people do almost anything they want with your project, except distributing closed source versions.

Anaconda, Bash, and GIMP use the GNU GPLv3.

What if none of these work for me?

My project isn't software.

There are licenses for that.

I want more choices.

More licenses are available.

I don't want to choose a license.

Here's what happens if you don't.

GNU General Public License: The Copyleft License

- Nobody should be restricted by the software they use. There are four freedoms that every user should have:
 - the freedom to use the software for any purpose,
 - the freedom to change the software to suit your needs,
 - the freedom to share the software with your friends and neighbors, and
 - the freedom to share the changes you make.
- Code must be made available
- Any modifications must be relicensed under the same license (copyleft)

Risks of “copyleft” licenses

- Example: GNU GPL
- Require licensing derivative works also with same license
 - This is intentional!
- Depending on a GPL project from within a proprietary or differently-licensed codebase is disaster
 - Viral effect of polluting everything else with GPL requirement
- Most companies will avoid GPL code with a ten-foot pole
 - Expect vetting process before engineers are allowed to use third-party libraries from GitHub, etc.

Lesser GNU Public License (LGPL)

- Software must be a library
- Similar to GPL but does not consider dynamic binding as “derivative work”
- So, proprietary code can depend on LGPL libraries as long as they are not being modified
- See also: GPL with classpath exception (e.g., Oracle JDK)

MIT License

- Simple, commercial-friendly license
- Must retain copyright credit
- Software is provided as is
- Authors are not liable for software
- No other restrictions

Apache License

- Similar to MIT license
- Not copyleft
- Not required to distribute source code
- Does not grant permission to use project's trademark
- Does not require modifications to use the same license

BSD License

- No liability and provided as is.
- Copyright statement must be included in source and binary
- The copyright holder does not endorse any extensions without explicit written consent

Creative Commons (CC)

- More common for licensing data-sets instead of code
 - Examples: images, websites, documentation, slides, plots, videos
- CC-BY (attribution only; derivatives allowed)
- CC-BY-SA (attribution and share-alike for derivatives)
- CC-BY-ND (attribution and no derivatives)

Dual License Business Model



- Released as GPL which requires a company using the open source product to open source it's application
- Or companies can pay \$2,000 to \$10,000 annually to receive a copy of MySQL with a more business friendly license

Risk: Incompatible Licenses

- Sun open-sourced OpenOffice, but when Sun was acquired by Oracle, Oracle temporarily stopped the project.
- Many of the community contributors banded together and created LibreOffice
- Oracle eventually released OpenOffice to Apache
- LibreOffice changed the project license so LibreOffice can copy changes from OpenOffice but OpenOffice cannot do the same due to license conflicts

Copyright vs. Intellectual Property (IP)

- IP and Patents cover an idea for solving a problem
 - Examples: Machine designs, pharma processes to manufacture certain drugs, (controversially) algorithms
 - Have expiry dates. IP can be licensed or sold/transferred for \$\$\$.
- Copyrights cover particular expressions of some work
 - Examples: Books, music, art, source code
 - Automatic copyright assignment to all new work unless a license authorizes alternative uses.
- Exceptions for trivial works and ideas.

Contributor Licensing Agreements (CLA)

- Often a requirement to sign these before you can contribute to OSS projects
 - Scoped only to that project
- Assigns the maintainers specific rights over code that you contribute
 - Without this, you own the copyright and IP for even small bug fixes and that can cause them legal headaches in the future

Dependency Management

Left-pad (March 22, 2016)

The screenshot shows a news article from The Register. At the top, there are labels 'OBSESSIONS' and 'QUARTZ'. Below them, a blue link 'NPM ERR!' is visible. The main headline reads: 'How one programmer broke the internet by deleting a tiny piece of code'. To the left of the headline is a black box with 'THE VERGE' in white, followed by 'TECH' and 'REVIEWS' in smaller text. Below the headline, a sub-headline says 'How an irate developer briefly broke JavaScript', followed by a quote: 'Unpublishing 11 lines of code brought down an open source house of cards.' and the byline 'By Paul Miller | @futurepaul | Mar 24, 2016, 4:25pm EDT'. A red banner with 'The Register' logo is below the byline. To the left of the banner are social media icons for Facebook and Twitter, and a 'SIGN IN' button. Below the banner, a black box contains the text '(* SOFTWARE *)'. The main body text starts with 'How one developer just broke Node, Babel and thousands of projects in 11 lines of JavaScript', followed by a sub-headline: 'Code pulled from NPM – which everyone was using'.

OBSESSIONS QUARTZ

NPM ERR!

How one programmer broke the internet by deleting a tiny piece of code

THE VERGE TECH REVIEWS

REPORT TECH

How an irate developer briefly broke JavaScript

Unpublishing 11 lines of code brought down an open source house of cards.

By Paul Miller | @futurepaul | Mar 24, 2016, 4:25pm EDT

f Twitter SIGN IN The Register

(* SOFTWARE *)

How one developer just broke Node, Babel and thousands of projects in 11 lines of JavaScript

Code pulled from NPM – which everyone was using

Left-pad (March 22, 2016)

npmjs.org tells me that left-pad is not available (404 page) #4



silkenfrance opened this issue on Mar 22, 2016 - 193 comments



silkenfrance commented on Mar 22, 2016

When building projects on travis, or when searching for left-pad on npmjs.com, both will report that the package cannot be found.

Here is an excerpt from the travis build log

```
npm ERR! Linux 3.13.0-40-generic
npm ERR! argv "/home/travis/.nvm/versions/node/v4.2.2/bin/node" "/home/travis/.nvm/versions/node/v4.2.2/bin/npm"
npm ERR! node v4.2.2
npm ERR! npm v2.14.7
npm ERR! code E404
npm ERR! 404 Registry returned 404 for GET on https://registry.npmjs.org/left-pad
npm ERR! 404
npm ERR! 404 'left-pad' is not in the npm registry.
npm ERR! 404 You should bug the author to publish it (or use the name yourself!)
npm ERR! 404 It was specified as a dependency of 'line-numbers'
npm ERR! 404
npm ERR! 404 Note that you can also install from a
npm ERR! 404 tarball, folder, http url, or git url.
npm ERR! Please include the following file with any support request:
npm ERR! /home/travis/build/coldrye-es/pingo/npm-debug.log
make: *** [deps] Error 1
```

And here is the standard npmjs.com error page <https://www.npmjs.com/package/left-pad>

However, if I remove left-pad from my local npm cache and then reinstall it using npm it will happily install left-pad@0.0.4.



88 3

Left-pad (Docs)

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

1.3.0

License

WTFPL

Unpacked Size

9.75 kB

Total Files

10

Issues

3

Pull Requests

7

Last publish

4 years ago

Left-pad (Source Code)

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 }
```

See also: isArray

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 };
```

isArray

Array.isArray for older browsers and deprecated Node.js versions.

build **passing** downloads **227M/month**



Just use `Array.isArray` directly, unless you need to support those older versions.

Usage

```
var isArray = require('isArray');

console.log(isArray([])); // => true
console.log(isArray({})); // => false
```

Install

```
> npm i isArray
```

Repository

github.com/juliangruber/isarray

Homepage

github.com/juliangruber/isarray

Weekly Downloads

50,913,317

Version

2.0.5

License

MIT

Unpacked Size

3.43 kB

Total Files

4

Issues

4

Pull Requests

3



Michael Hilton 7:28 PM

Did I break something? I was pushing changes for P4, I thought they were all in a branch...



Sarah Cross 7:29 PM

No, I think NodeBB broke something



Open the home page
Commits on Nov 1, 2025

Merge pull request #457 from CMU-313/fix_issues_before_p4

Verified

ec71934



MichaelHilton authored 4 days ago · ✓ 2 / 2

changes to get tests to pass

645bc18



MichaelHilton committed 4 days ago · ✓ 2 / 2

resolved conflict

3a4275a



MichaelHilton committed 4 days ago · ✗ 1 / 2

bug: resolved font path issues

f1e7e5c



html1101 authored and MichaelHilton committed 4 days ago

perf: updated to sync with bugfix + pass tests

3199cac



html1101 authored and MichaelHilton committed 4 days ago

Commits on Oct 30, 2025

Upgrade nodebb-theme-harmony (#455)

Verified

c0e2f82



html1101 authored last week · ✓ 2 / 2

Dependency Management

- It's hard
- It's mostly a mess (everywhere)
- But it's critical to modern software development

What is a Dependency?

- Core of what most build systems do
 - “Compile” and “Run Tests” is just a fraction of their job
- Examples: Maven, Gradle, NPM, Bazel, ...
- **Foo->Bar: To build Foo, you may need to have a built version of Bar**
- Dependency Scopes:
 - **Compile:** Foo uses classes, functions, etc. defined by Bar
 - **Runtime:** Foo uses an abstract API whose implementation is provided by Bar (e.g. logging, database, network or other I/O)
 - **Test:** Foo needs Bar only for tests (e.g. JUnit, mocks)
- Internal vs. External Dependencies
 - Is Bar also built/maintained by your org or is it pulled from elsewhere using a package manager?

Dependencies: Example

```
github.com/CMU-213/tevybits/main.xml
...
152 <dependencyManagement>
153 <dependencies>
154 <dependency>
155 <groupId>com.sismics.docs</groupId>
156 <artifactId>docs-core</artifactId>
157 <version>${project.version}</version>
158 </dependency>
159
160 <dependency>
161 <groupId>com.sismics.docs</groupId>
162 <artifactId>docs-web-common</artifactId>
163 <version>${project.version}</version>
164 </dependency>
165
166 <dependency>
167 <groupId>com.sismics.docs</groupId>
168 <artifactId>docs-web-common</artifactId>
169 <type>test-jar</type>
170 <version>${project.version}</version>
171 </dependency>
172
173 <dependency>
174 <groupId>com.sismics.docs</groupId>
175 <artifactId>docs-web</artifactId>
176 <version>${project.version}</version>
177 </dependency>
178
179 <dependency>
180 <groupId>org.eclipse.jetty</groupId>
181 <artifactId>jetty-server</artifactId>
182 <version>${org.eclipse.jetty.jetty-server.version}</version>
183 </dependency>
184
185 <dependency>
186 <groupId>org.eclipse.jetty</groupId>
187 <artifactId>jetty-webapp</artifactId>
188 <version>${org.eclipse.jetty.jetty-webapp.version}</version>
189 </dependency>
```

Package: git (1:2.17.1-1ubuntu0.9)

fast, scalable, distributed revision control system

Other Packages Related to git

- depends recommends suggests enhances
- **git-man** (< 1:2.17.0-1) [not amd64, i386]
fast, scalable, distributed revision control system (manual pages)
- **git-man** (< 1:2.17.1-1) [amd64, i386]
- **git-man** (>= 1:2.17.0) [not amd64, i386]
- **git-man** (>= 1:2.17.1) [amd64, i386]
- **libc6** (>= 2.18) [not amd64, ppc64el]
GNU C Library. Shared libraries
also a virtual package provided by **libc6-udeb**
- **libc6** (>= 2.17) [arm64, ppc64el]
- **libcurl3-gnutls** (>= 7.16.2)
easy-to-use client-side URL transfer library (GnuTLS flavour)
- **liberror-perl**
Perl module for error/exception handling in an OO-ish way
- **libexpat1** (>= 2.0.1)
XML parsing C library - runtime library
- **libpcre3**
Old Perl 5 Compatible Regular Expression Library - runtime files
- **perl**
Larry Wall's Practical Extraction and Report Language
- **zlib1g** (>= 1:1.2.8)
compression library - runtime
- **less**
pager program similar to more
- **patch**
Apply a diff file to an original
- **ssh-client**
virtual package provided by **openssh-client**

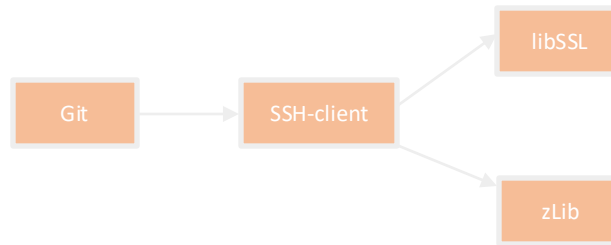
Where are the dependencies hosted?

- Typically downloaded from dependency servers:
 - Maven Central (<https://repo.maven.apache.org/maven2/>)
 - Ubuntu Packages for Apt (<https://packages.ubuntu.com/>)
 - Python Package Index (<https://pypi.org/>)]
 - NPM Public Registry (<https://registry.npmjs.org/>)
- Packages need a unique identifier
 - Typically a package name (sometimes owner name) and version
- Custom repositories allowed by most package managers
 - Often used for company-internal packages or cache mirroring
 - Note problems with duplicates (same package name in different repositories; some priority order is needed)
- Somebody needs to manage repositories
 - Availability: Repository needs to be running
 - Access Control: Packages should only be published by owners
 - Integrity: Packages should be signed or otherwise verifiable
 - Uniqueness and archival: Only one artifact per version
 - Traceability: Packages can have metadata pointing to source or tests
 - Security: ???



Transitive Dependencies

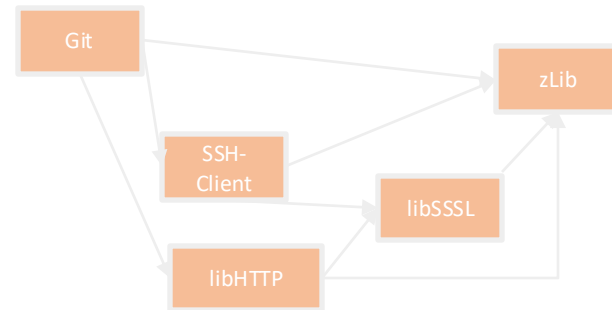
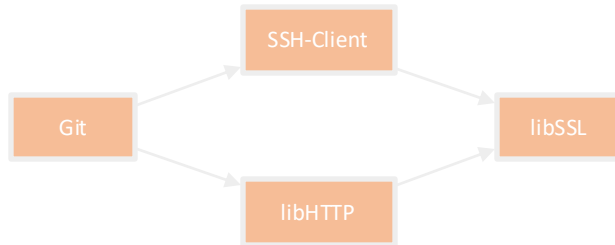
Packages can depend on other packages



Q: Should Git be able to use exports of libSSL (e.g. certificate management) or zLib (e.g. gzip compression)?

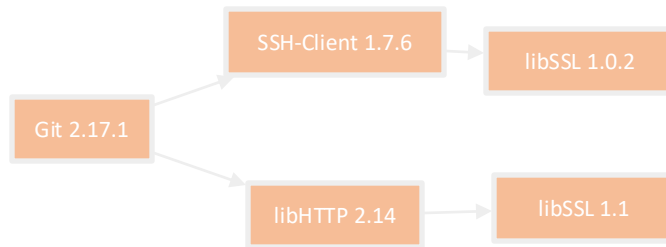
Diamond Dependencies

What are some problems when multiple intermediate dependencies have the same transitive dependency?



Diamond Dependencies

What are some problems when multiple intermediate dependencies have the same transitive dependency?



Resolutions to the Diamond Problem

1. Duplicate it!
 - Doesn't work with static linking (e.g. C/C++), but may be doable with Java (e.g. using ClassLoader hacking or package renaming)
 - Values of types defined by duplicated libraries cannot be exchanged across
2. Ban transitive dependencies; just use a global list with one version for each
 - Challenge: Keeping things in sync with latest
 - Challenge: Deciding which version of transitive deps to keep
3. Newest version (keep everything at latest)
 - Requires ordering semantics
 - Intermediate dependency may break with update to transitive
4. Oldest version (lowest denominator)
 - Also requires ordering semantics
 - Sacrifices new functionality
5. Oldest non-breaking version / Newest non-breaking version
 - Requires faith in tests or semantic versioning contract

Semantic Versioning

- Widely used convention for versioning releases
 - E.g. 1.2.1, 3.1.0-alpha-1, 3.1.0-alpha-2, 3.1.0-beta-1, 3.1.0-rc1
- Format: {MAJOR} . {MINOR} . {PATCH}
- Each component is ordered (numerically, then lexicographically; release-aware)
 - $1.2.1 < 1.10.1$
 - $3.1.0\text{-alpha-1} < 3.1.0\text{-alpha-2} < 3.1.0\text{-beta-1} < 3.1.0\text{-rc1} < 3.1.0$
- Contracts:
 - MAJOR updated to indicate breaking changes
 - Same MAJOR version \Rightarrow backward compatibility
 - MINOR updated for additive changes
 - Same MINOR version \Rightarrow API compatibility (important for linking)
 - PATCH updates functionality without new API
 - Ninja edit; usually for bug fixes

<https://semver.org/>

2.0.0 2.0.0-rc.2 2.0.0-rc.1 1.0.0 1.0.0-beta

Semantic Versioning 2.0.0

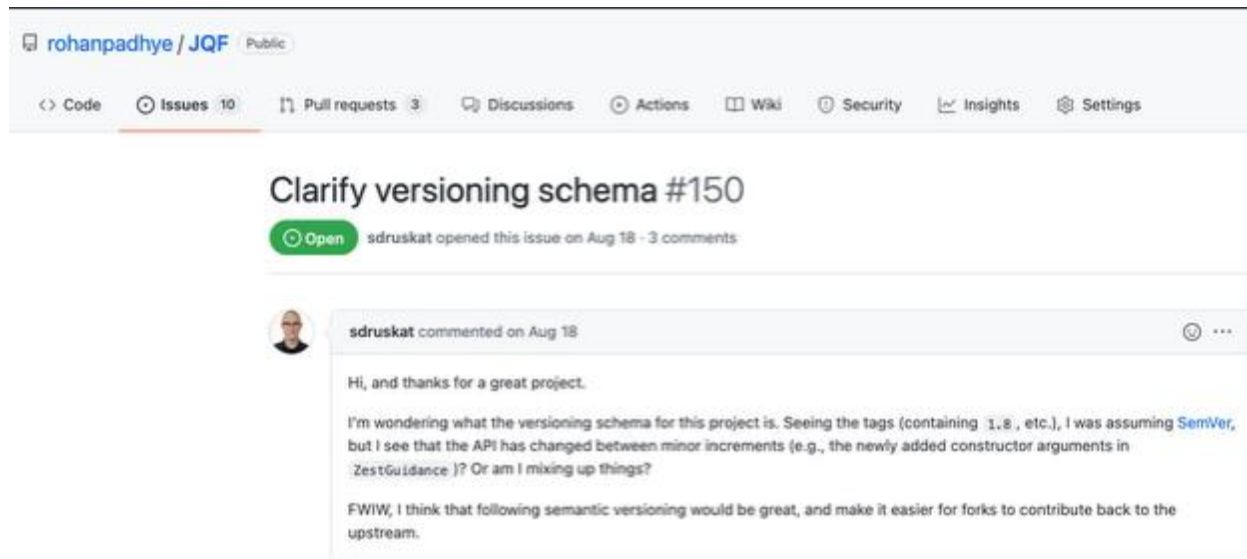
Summary

Given a version number MAJOR.MINOR.PATCH, increment the:

1. MAJOR version when you make incompatible API changes,
2. MINOR version when you add functionality in a backwards compatible manner, and
3. PATCH version when you make backwards compatible bug fixes.

Additional labels for pre-release and build metadata are available as extensions to the MAJOR.MINOR.PATCH format.

People rely on SemVer contracts



The screenshot shows a GitHub repository page for 'rohanpadhye / JQF' with a 'Public' label. The navigation bar includes links for Code, Issues (10), Pull requests (3), Discussions, Actions, Wiki, Security, Insights, and Settings. The 'Issues' tab is selected, displaying an issue titled 'Clarify versioning schema #150'. The issue is marked as 'Open' and was opened by 'sdruskat' on August 18, with 3 comments. A comment from 'sdruskat' is visible, dated August 18. The comment text is as follows:

Hi, and thanks for a great project.

I'm wondering what the versioning schema for this project is. Seeing the tags (containing `1.8`, etc.), I was assuming [SemVer](#), but I see that the API has changed between minor increments (e.g., the newly added constructor arguments in `ZestGuidance`)? Or am I mixing up things?

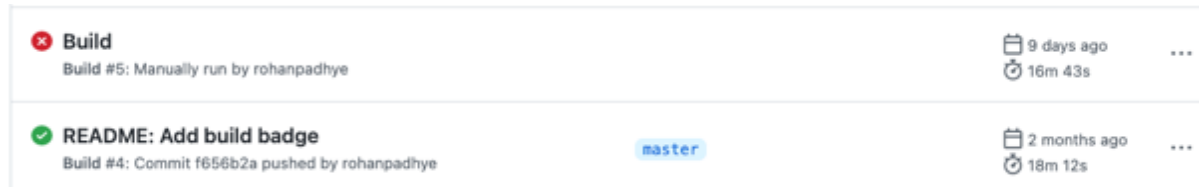
FWIW, I think that following semantic versioning would be great, and make it easier for forks to contribute back to the upstream.

Dependency Constraints

- E.g. Declare dependency on "Bar > 2.1"
 - Bar 2.1.0, 2.1.1, 2.2.0, 2.9.0, etc. all match
 - 2.0.x does NOT match
 - 3.0.x does NOT match
- Diamond dependency problem can be resolved using SAT solvers
 - E.g. Foo 1.0.0 depends on "Bar >= 2.1" and "Baz 1.8.x"
 - Bar 2.1.0 depends on "Qux [1.6, 1.7]"
 - Bar 2.1.1 depends on "Qux 1.7.0"
 - Baz 1.8.0 depends on "Qux 1.5.x"
 - Baz 1.8.1 depends on "Qux 1.6.x"
 - Find an assignment such that all dependencies are satisfied
 - Solution: Use Bar 2.1.0, Baz 1.8.1, and Qux 1.6.{latest}

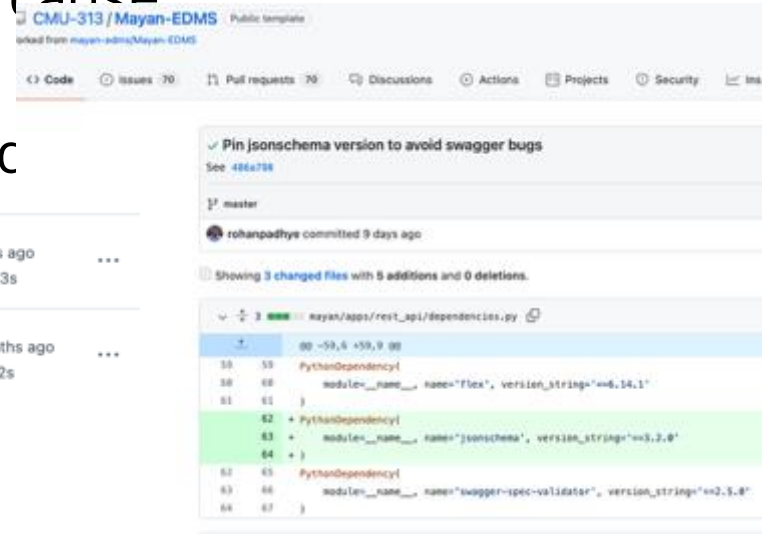
Semantic Versioning Contracts

- Largely trusting developers to maintain them
- Constrained/range dependencies can cause unexpected build failures
- Automatic validation of SemVer is hard



A screenshot of the GitHub Actions workflow runs for a repository. It shows two workflow runs:

- Build** (marked with a red 'x' icon): Build #5: Manually run by rohanpadhye. It was run 9 days ago and took 16m 43s. A 'master' branch badge is visible.
- README: Add build badge** (marked with a green checkmark icon): Build #4: Commit f656b2a pushed by rohanpadhye. It was run 2 months ago and took 18m 12s.



A screenshot of a GitHub pull request titled "Pin jsonschema version to avoid swagger bugs" (See 486a798). The pull request is from the 'master' branch and was committed 9 days ago by rohanpadhye. It shows 3 changed files with 5 additions and 0 deletions. The diff view shows changes to the file `mayan/apps/rest_api/dependencies.py`:

```
50 50 PythonDependency(
50 60     module=__name__, name="flex", version_string="=6.14.1"
61 61 )
62 + PythonDependency(
63 +     module=__name__, name="jsonschema", version_string="=3.2.0"
64 + )
62 63 PythonDependency(
63 64     module=__name__, name="swagger-spec-validator", version_string="=2.5.8"
64 67 )
```

Cyclic Dependencies

- A very bad thing
- Avoid at all costs
- Sometimes unavoidable or intentional
 - E.g. GCC is written in C (needs a C compiler)
 - E.g. Apache Maven uses the Maven build system
 - E.g. JDK tested using JUnit, which requires the JDK to compile



Cyclic Dependencies

- Bootstrapping: Break cycles over time
- Assume older version exists in binary (pre-built form)
- Step 1: Build A using an older version of B
- Step 2: Build B using new (just built) version of A
- Step 3: Rebuild A using new (just built) version of B
- Now, both A and B have been built with new versions of their dependencies
- Doesn't work if both A and B need new features of each other at the same time (otherwise Step 1 won't work)
 - Assumes incremental dependence on new features
- How was the old version built in the first place? (it's turtles all the way down)
 - Assumption: cycles did not exist in the past
 - Successfully applied in compilers (e.g. GCC is written in C)

Dependency Security

- Will you let strangers execute arbitrary code on your laptop?
 - Think about this every time you do “pip install” or “npm install” or “apt-get upgrade” or “brew upgrade” or whatever (esp. with sudo)
 - Scary, right? Who are you trusting? Why?
- Typo squatting (“pip install numpi”)
- Outright malice (remember the event-stream incident?)
- Genuine security vulnerabilities due to software bugs

Dependency alerts / #74

Deserialization of Untrusted Data in Apache Log4j #74

 Open - Opened 3 days ago on log4j/log4j (Maven) - go on err

Package	Affected versions	Patched version
 log4j/log4j (Maven)	≤ 1.2.17	None

CVE-2020-9493 identified a deserialization issue that was present in Apache Chainsaw. Prior to Chainsaw V2.0 Chainsaw was a component of Apache Log4j 1.2.x where the same issue exists.

Users are advised to migrate from `log4j:log4j` to `org.apache.logging.log4j:log4j` for an updated version of the library.

 dependabot bot · opened this 3 days ago

Severity

Critical 9.8 / 10

Metric	Value
Attack vector	Network
Attack complexity	Low
Privileges required	None
User interaction	None
Scope	Unchanged
Confidentiality	High
Integrity	High
Availability	High

CVSS 3.1 SAIR:NAC:L:PRN:N:N:N:N:UC:REP:N:K:K:K

Weaknesses

CWE-502

NOTE: Beware adding dependencies

- LLM's love to add new dependencies. This can:
 - Increase complexity + technical debt
 - Cause license problems
 - Duplicate effort

Adding dependencies is unlikely to be the right answer. Be VERY CAREFUL of added dependencies.

