Upgrading NPM packages

and how to ensure it’s not so painful next time.

Livvy Mackintosh

Storytime

You inherit a project from another team and notice a lot of packages are very out of date. You attempt to upgrade the packages but all of your tests go red; and after battling for several hours you decide to throw in the towel and write a tech debt ticket.

Eventually, the knowledge of the hundreds of security vulnerabilities is on your mind. You decide to fix the packages even though it’s not a priority in your sprint. Left exhausted and tired, you wonder how you can prevent such a situation in the future.

Things you should ask yourself before you `$ yarn add foo`

Do you *really* need that package?

Seriously. Have you actually read the source code?

If it’s a single module containing 20 lines, consider re-implementing it in your project.

Think like a minimalist.

That package probably does more than you need. Software is already bloated, please don’t make the problem worse. If you can achieve your goal with less code, do it. Your codebase will have a longer lifespan.

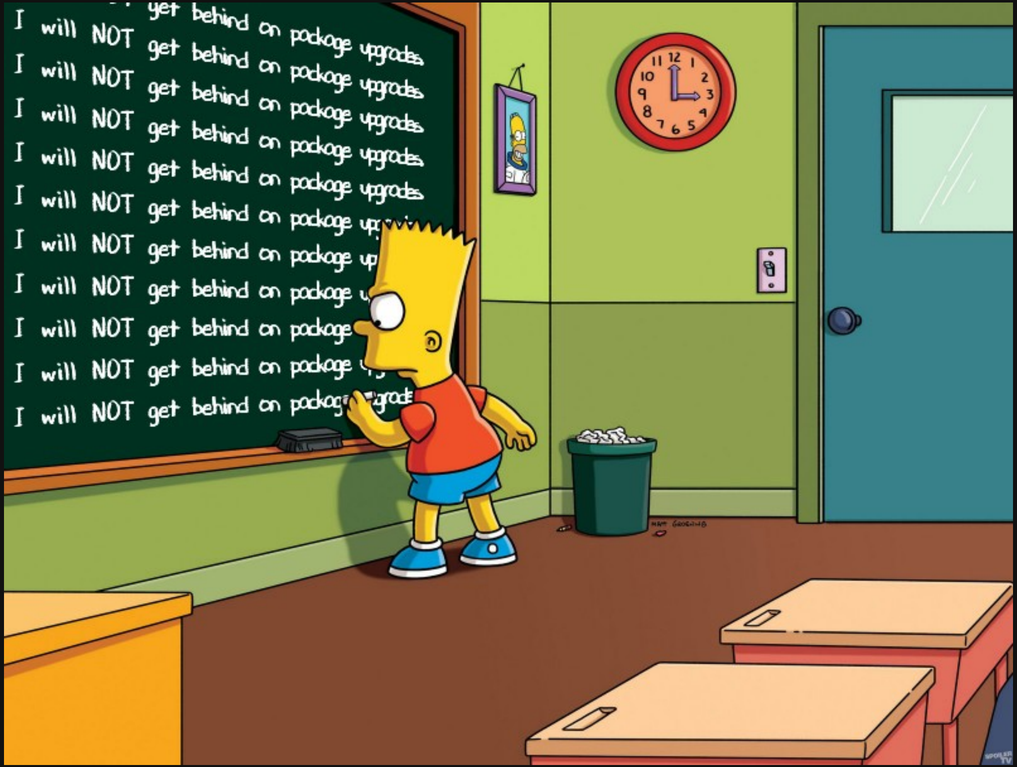
https://suckless.org/philosophy/

Storytime 2.0

Kevin decided he was smart and has created a document converter that uses 5 libraries and converts Markdown to XML and then to PDF.

After some serious consideration, you decide you need that package.

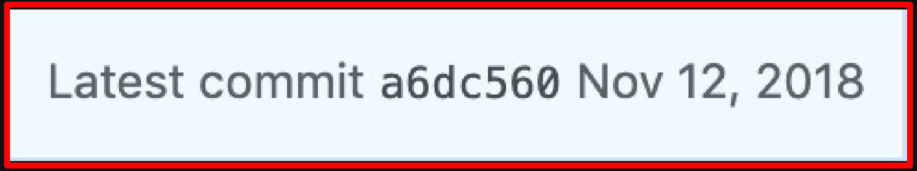
What do you need to be telling yourself now?


I will NOT get behind on package upgrades
I will NOT get behind on package upgrades
I will NOT get behind on package upgrades
I will NOT get behind on package upgrades
I will NOT get behind on package upgrades
I will NOT get behind on package upgrades

Even if there is no security holes now, you can bet there will be soon.

If you're using packages that aren't backed by a company or *paid person*, keep an eye on GitHub.

99.9%* of projects will be abandoned. Anticipate this in advance and have a plan.



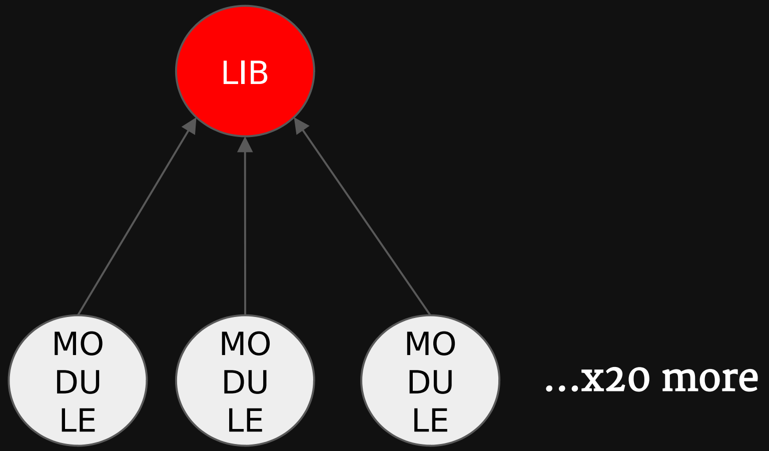
Latest commit a6dc560 Nov 12, 2018

If you see this, start getting worried and **contact the maintainer**. If it’s a small library, consider merging it into your codebase or offering to maintain it.

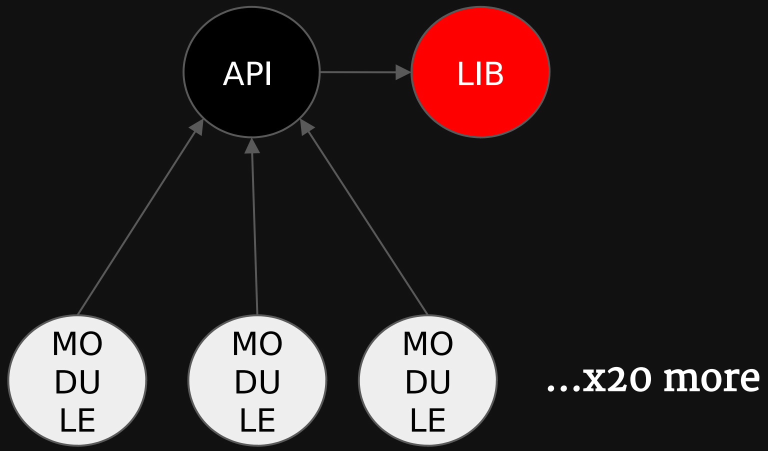
Consider creating an internal API you can use for external dependencies.

(Changing calls to a library in 100 places isn't fun.)

Painful



Less Painful



Take advantage of *loose dependencies.*

(But still pin your dependencies separately).
This way, you can leverage the power of your package manager to find the versions of a package that are compatible with each other.

“That’s all well and good Livvy but I’m already in the quagmire. How do I get out?”

It’s not an exact science.

1. Stay calm
2. Drink some coffee and get in the zone
3. Be like a watchmaker

1. Pick one dependency
2. Bump the version *a bit*
3. Read the changelog
4. Fix breaking API changes
5. Keep running tests and repeat 3+4 until they pass
6. Celebrate lightly
7. Rinse and repeat

Remember: It’s much easier to not add the dependency in the first place. Be kind to yourself and your colleagues.

JUST SAY NO :)

Fin.