
shrinkpack

Fast, resilient, reproducible builds with npm install.

downloads 1.1k/month downloads 1.1k/month  Follow 417  Follow

What

`shrinkpack` points your `package-lock.json` at npm tarballs checked into your project's source control, so you can install while offline, during a registry outage, or during the next left-pad incident.

How

1. Read `package-lock.json` or `npm-shrinkwrap.json`.
2. Download the exact same .tgz files that `npm install` fetches from registry.npmjs.org.
3. Decompress the .tgz files into .tar files. This avoids storing binary files in Git and removes the cost of decompression during `npm install`.
4. Store the .tar files in your project at `node_shrinkpack/*.tar`.
5. Rewrite `package-lock.json` to point at those instead of the registry.

Now your project can be installed while completely offline:

```
1 - npm install
2 + npm ci --offline
```

The rest of the npm installation process is exactly the same. The only difference is that no network activity is necessary when installing and building your project. The `node_shrinkpack` directory can be ignored in your editor (much like is done with the `node_modules` directory), but is instead checked into source control.

Why

For context, please see the target problem and justification sections of this README.

Installation

Requires npm@7 or higher.

```
1 npm install --global shrinkpack
```

Usage

Run `shrinkpack` every time you have modified and installed your dependencies to produce a new `package-lock.json`.

```
1 Usage: shrinkpack [options] [directory]
2
3 Options:
4   -V, --version  output the version number
5   -h, --help     display help for command
6
7 Icons:
8   + Added
9   - Removed
10  i Information
11  12:34 Time Taken
```

History

1. **Feb 2015:** shrinkpack was created.
2. **Mar 2016:** The left-pad incident happened (shrinkpack users were unaffected!).
3. **Jun 2016:** yarn added “offline mirror” support.
4. **May 2017:** npm@5 added package-lock.json but npm@5 broke support for installing local files from a lockfile. Subsequent fixes were released for npm, but the issue was not resolved.
5. **Apr 2018:** npm announced plans to integrate shrinkpack functionality into npm.
6. **May 2018:** Work on shrinkpack is abandoned after the regression in npm@5 is still not fixed after a year.
7. **Dec 2021:** Jack Franklin wrote why you should check-in your node dependencies and I’m reminded of why I wrote shrinkpack in the first place.
8. **Dec 2021:** Work resumes on shrinkpack and support is added for npm >= 7.

Target Problem

Back in 2015 I was working at skysports.com. Each time we pushed code, our continuous integration environment created a clean workspace, installed, configured, and built the latest version of the project, then ran various tests and tasks.

We were happy with this process and the convenience of npm in particular, but the phase of our builds where `npm install` listed a *huge* amount of network traffic would always raise the same concerns:

- This seems slow, wasteful, and inefficient.

-
- We *really* depend on `registry.npmjs.org`, what do we do if it goes down?

The first suggestion was always to check in the `node_modules` directory, but the idea of large and chatty commits whenever we chose to upgrade or change a dependency put us off.

Other teams felt they could live with that and decided to proceed, only to find that packages such as `phantomjs` and `node-sass` will helpfully install the appropriate binary depending on which operating system you're running.

This meant that if Chris added `phantomjs` or `node-sass` to the project on his Mac and checked it into the repository, Helen wouldn't be able to use it on her Windows Machine.

The remaining alternatives were caching proxies or self-hosted registry mirrors, and caches-of-sorts. None of which appealed to us and, grudgingly, we continued as we were until later creating `shrinkpack`.

Justification

Note: This section was first written in 2015, before lockfiles were the default in `npm`, `pnpm`, and `yarn`. You had to opt-in to using a lockfile by running `npm shrinkwrap` to generate an `npm-shrinkwrap.json` file.

This text has been updated to reflect the situation today, where the need for lockfiles is more widely understood.

Whenever we add, remove, or update an `npm` dependency — we should test our application for regressions before locking down our dependencies with a lockfile. A tagged release should be a locked-down, frozen snapshot of the codebase which has been tested sufficiently enough that it is approved and trusted. When fed into a repeatable, automated deployment process it should always result in the same output.

- Without a lockfile your dependency graph will mutate on a regular basis.
- Checking in `node_modules` fixes this, but there are some issues which we discussed earlier.
- You can be reasonably sure your dependency graph will remain consistent with a lockfile.
- You can be completely sure with a lockfile *and* an offline cache.

A lockfile is something I would recommend you use anyway, even if you don't decide to use `shrinkpack`. It increases (but doesn't guarantee) certainty and confidence over exactly what versions of every nested dependency you've tested against and approved.

Without a lockfile and an offline cache, that's not guaranteed.

Consider this snippet from the `package.json` of a nested dependency in your project as an example. It's not even a package you directly control, it's a dependency of a dependency of a dependency:

```
1 "dependencies": {  
2   "lolwut": ">=0.1.0"  
3 }
```

If `lolwut@0.2.4` contains a regression and you're not using a lockfile, your project will contain that regression the next time you install it.

shrinkpack

With you hopefully convinced of the merits of lockfiles, `shrinkpack` will hopefully be seen as a small and complementary addition.

`shrinkpack` takes the tarballs of the specific dependency graph described by your lockfile and stores them within your project.

This means;

- No need for repeated requests to `registry.npmjs.org`.
- Each package/version pair can be checked in as a single tarball, avoiding commits with all kinds of noisy diffs.
- Packages can be checked in, while still being installed by members of the team on different operating systems.

Suitability to your project

`shrinkpack` is best suited to a project which is the root consumer of dependencies and not a dependency itself. If your project is intended to be installed as a dependency of another project using `npm install`, let those downstream projects make their own decisions on bundling.

That said, if you're developing an npm package and want to use `shrinkpack` to speed up and harden your development and CI environments, adding `package-lock.json` and `node_shrinkpack` to your `.npmignore` file will allow you to do that, without publishing your shrinkpacked dependencies to the registry.

It's not recommended to publish a project with bundled or shrinkpacked dependencies to the registry, which would become bloated with duplicate copies of packages, bundled amongst various other ones.

Getting Help

- Get help with issues by creating a Bug Report.
- Discuss ideas by opening a Feature Request.