

# Offline installation of npm packages

JANUARY 2, 2016

Occasionally, you may need to install [npm](#) packages while offline. This could be due to a flaky network connection, being on a flight or during a workshop. Getting this working has been a dream for a while.

They say you should *follow* your dreams. Unless it's been a while and they haven't followed you back, in which case unfollow them and move on. Luckily, there are a few options available to get npm working offline today.

## --cache-min

First, a built-in (but ultimately incomplete) option. In theory, when you want to [force](#) installation from the npm cache, you can use the `--cache-min` flag with a high value. To install a package from the cache, run:

```
$ npm --cache-min 9999999 install <package-name>
```

Alternatively, use `npm --cache-min Infinity`. Typing this out every time you need it is a little tedious, so alias it in your dotfiles. In my bash files, I've previously used:

```
npmc="npm --cache-min 9999999 "
```

In an ideal world, the npm client would just alias this to `--offline`. So, why isn't this just a default in npm3?

## It's the hero Node deserves, but not the one it needs right now

The above `--cache-min` hack has a number of pretty undesirable shortcomings.

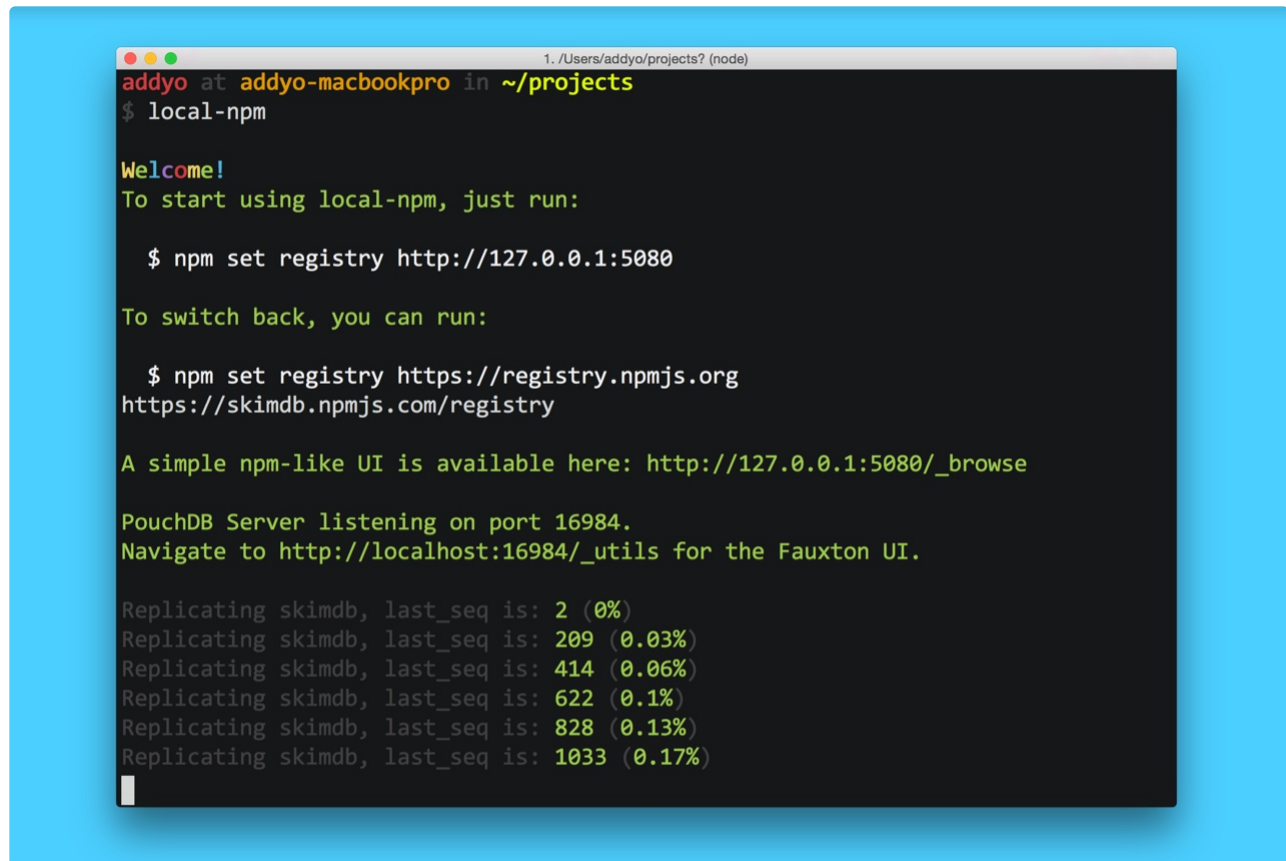
When using it, npm **still connects** to the registry over the network if a package has dependencies that aren't already in the cache. It will also **fail** if you install a package that depends on a newer version of a package than the one installed in the past.

The npm team have **concerns** about shipping a proper offline flag as-is as most users aren't going to understand the subtleties of how the npm cache works. This is understandable. From what I gather, they are open to eventually shipping full support for an offline experience but until then, let's look at some third party stop gap solutions.

## local-npm

A significantly more robust solution to `--cache-min` is Nolan Lawson's offline-first **local-npm** package (which I love). It's a Node server that acts like a local npm mirror (without needing to do a complete replication of the whole npm registry).

Using `local-npm`, your `npm install`s are fetched from the registry and then modules and their deps get stored in a local **PouchDB** database. This caches them so subsequent `npm install`s use the local cache rather than calling to the network. `local-npm` also takes care of keeping modules updated when they change. It does this by listening for changes to the remote registry so you don't have to worry about staleness.

A terminal window titled '1. /Users/addyo/projects? (node)' showing the installation and initial setup of local-npm. The user runs '\$ local-npm', which outputs a welcome message and instructions. The user then sets the npm registry to 'http://127.0.0.1:5080'. The terminal shows the replication progress of the skimdb database, with the last sequence number increasing from 2 to 1033 over time.

```
addyo at addyo-macbookpro in ~/projects
$ local-npm

Welcome!
To start using local-npm, just run:

$ npm set registry http://127.0.0.1:5080

To switch back, you can run:

$ npm set registry https://registry.npmjs.org
https://skimdb.npmjs.com/registry

A simple npm-like UI is available here: http://127.0.0.1:5080/_browse

PouchDB Server listening on port 16984.
Navigate to http://localhost:16984/_utils for the Fauxton UI.

Replicating skimdb, last_seq is: 2 (0%)
Replicating skimdb, last_seq is: 209 (0.03%)
Replicating skimdb, last_seq is: 414 (0.06%)
Replicating skimdb, last_seq is: 622 (0.1%)
Replicating skimdb, last_seq is: 828 (0.13%)
Replicating skimdb, last_seq is: 1033 (0.17%)
```

To get local-npm installed, run:

```
$ npm install -g local-npm
```

npm is built on top of [CouchDB](#). local-npm replicates the skimdb part of this database to a local PouchDB instance. Running:

```
$ local-npm
```

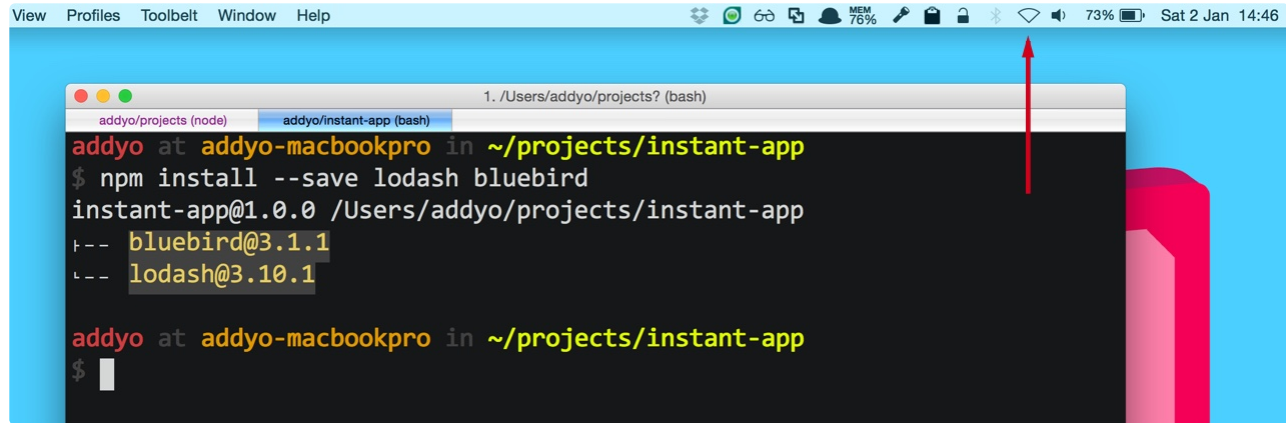
will begin the replication process. There's no need to wait for the complete skimdb download (which can take a few hours). It will fall back to the network for any libraries that aren't yet replicated. As mentioned, local-npm is offline-first. It has upfront replication for metadata and tarballs download the first time you install a specific version.

To complete setup, you'll want to go ahead and set npm to point to the local server local-npm is running:

```
$ npm set registry http://127.0.0.1:5080
```

Awesome. You should now be able to open up a new tab and just `npm install` any dependencies you require using the caching proxy.

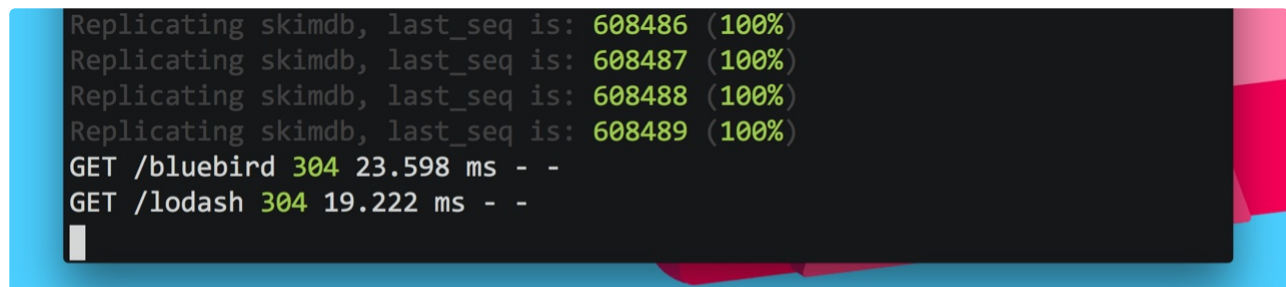
Here's what a successful offline `npm install` should look like:

A screenshot of a macOS terminal window. The window title is "1. /Users/addyo/projects? (bash)". The prompt is "addyo at addyo-macbookpro in ~/projects/instant-app". The command executed is "npm install --save lodash bluebird". The output shows "instant-app@1.0.0 /Users/addyo/projects/instant-app" followed by a tree view of installed dependencies: "bluebird@3.1.1" and "lodash@3.10.1". The prompt returns to "addyo at addyo-macbookpro in ~/projects/instant-app". A red arrow points to the top of the terminal window.

```
addyo at addyo-macbookpro in ~/projects/instant-app
$ npm install --save lodash bluebird
instant-app@1.0.0 /Users/addyo/projects/instant-app
└-- bluebird@3.1.1
└-- lodash@3.10.1

addyo at addyo-macbookpro in ~/projects/instant-app
$
```

and for posterity, here's a tab where I'm running `local-npm` and we can see it correctly proxying through requests for both the `bluebird` and `lodash` modules:

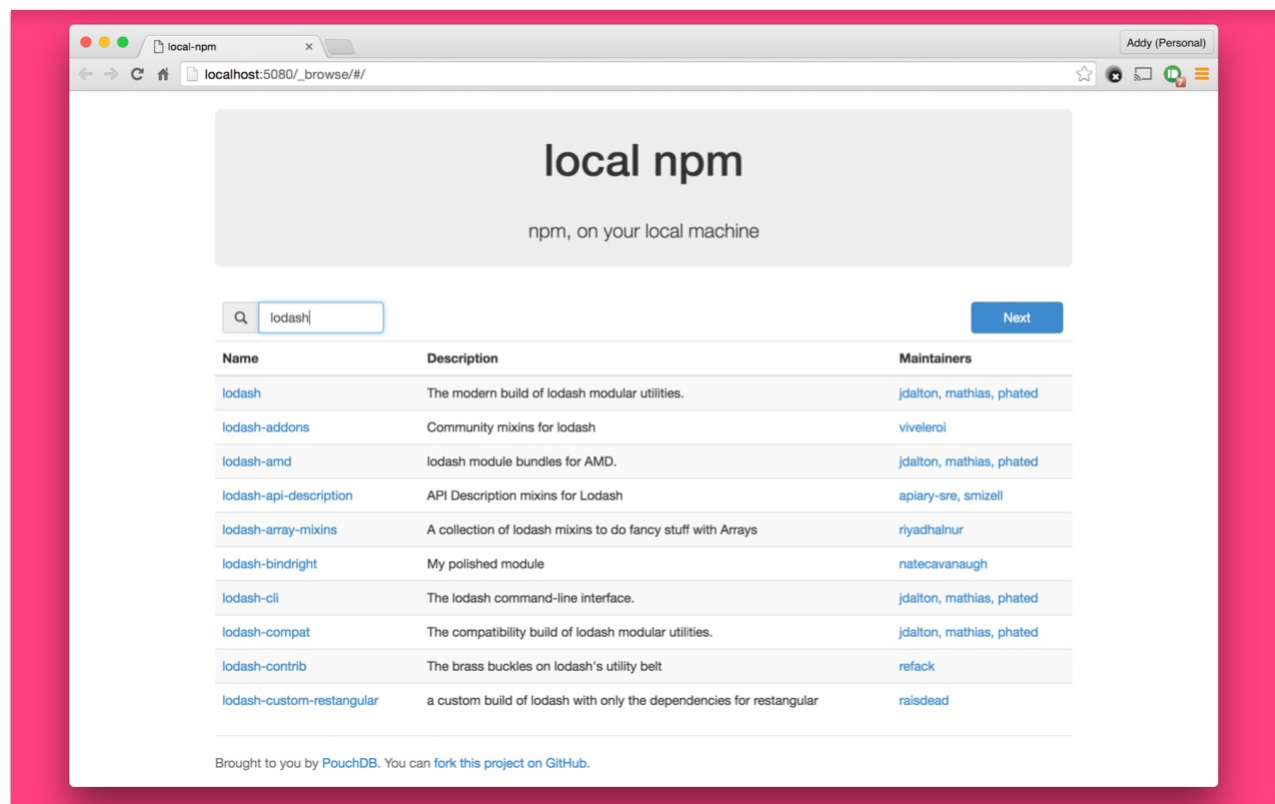
A screenshot of a terminal window showing the output of the `local-npm` command. It displays four lines of "Replicating skimdb" with increasing last\_seq values (608486 to 608489) and 100% completion. Below these, it shows two GET requests: "GET /bluebird 304 23.598 ms - -" and "GET /lodash 304 19.222 ms - -".

```
Replicating skimdb, last_seq is: 608486 (100%)
Replicating skimdb, last_seq is: 608487 (100%)
Replicating skimdb, last_seq is: 608488 (100%)
Replicating skimdb, last_seq is: 608489 (100%)
GET /bluebird 304 23.598 ms - -
GET /lodash 304 19.222 ms - -
```

To test everything works as expected:

- Check your network connection is up
- Run an `npm install` to grab a module or collection of dependencies from a `package.json` file. For example, `lodash`.
- At this point, `local-npm` will have a cached version of these modules stored
- Clear the `npm` cache using `npm cache clean` or delete the modules from your `node_modules` directory
- Turn off your wifi/kill your network connection
- Try running that `npm install` again. Everything should install correctly through the `local-npm` cache without hitting the network at all

A handy extra that comes with `local-npm`'s server setup is a simple in-browser UI for browsing local modules and searching for them. You can access this at [http://localhost:5080/\\_browse](http://localhost:5080/_browse).



Publishing packages from `local-npm` doesn't require any [change](#) of registry configuration. Just ensure you're online and everything should work as expected.

As a reminder, in case you need to switch back to the main registry or your preferred mirror for any reason just use `npm set` once again:

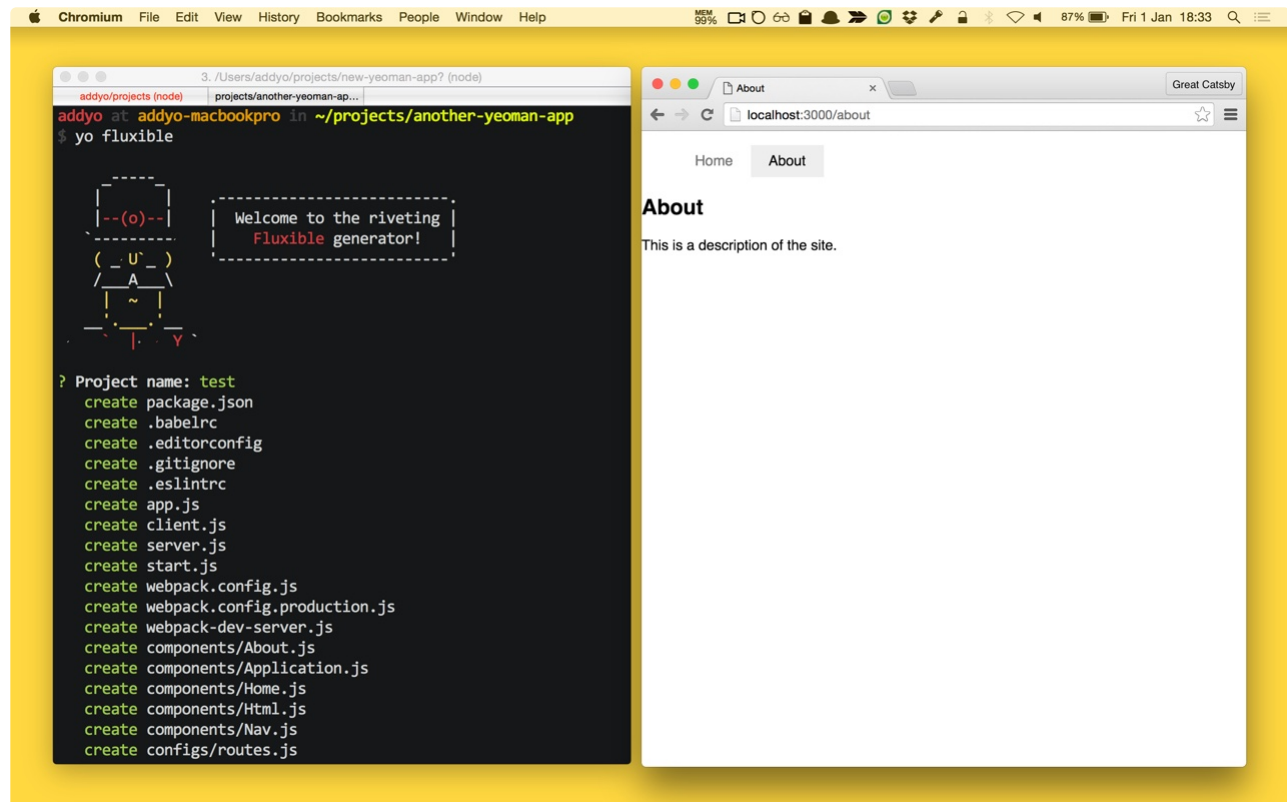
```
$ npm set registry https://registry.npmjs.org
```

Although `local-npm` is pretty great, it's not specifically the [direction](#) the npm team want to head in for their offline support. Regardless, it's still a pretty solid stop gap for today and I strongly recommend giving it a shot.

Since publishing this post, Nolan has also shared some [speed tests](#) comparing `local-npm` to regular npm. In short, regular is faster the first time you `npm install` a package, but after this `local-npm` is consistently faster by an order of 2-3 times.

## Using local-npm with other tools

I tested local-npm with Yeoman and Yahoo's [generator-fluxible](#), which only uses npm for package management. As you can see below, everything including dependency installation worked fine and I was able to preview the output from a cached install without issues:

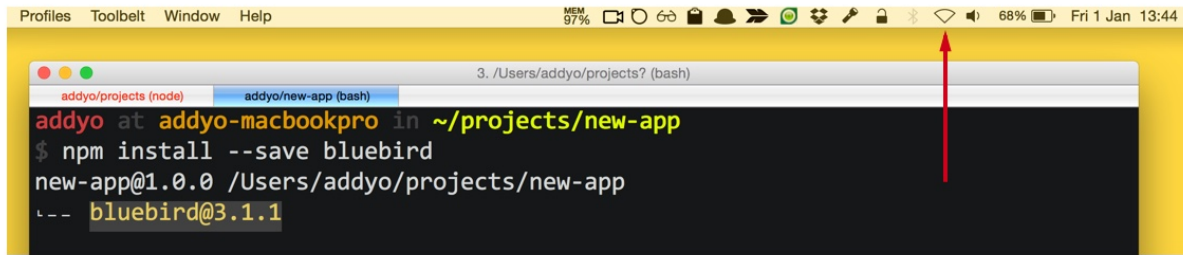


## npm\_lazy

Another popular lazy caching proxy for npm is [npm\\_lazy](#). Similar to local-npm, once setup any modules requested that aren't in the local cache are fetched from the npm registry lazily.

In npm\_lazy, the only things cached are package index metadata and package tarfiles. Pretty much all other endpoints get transparently proxied. This means an npm install for cached packages will always work and (mostly the same as local-npm), more exotic endpoints won't work if the main registry or mirrors are down - they'll act as their non-npm\_lazy equivalents.

## Attempt offline install



```

Profiles  Toolbelt  Window  Help
MEM 97%
3. /Users/addyo/projects? (bash)
addyo at addyo-macbookpro in ~/projects/new-app
$ npm install --save bluebird
new-app@1.0.0 /Users/addyo/projects/new-app
└─┬ bluebird@3.1.1

```

## npm\_lazy falls back to the cache when the network fails



```

app debug [GET] https://registry.npmjs.org/bluebird
app debug Fetch failed (5/5): https://registry.npmjs.org/bluebird { [Error: getaddrinfo ENOTFOUND registry.npmjs.org registry.npmjs.org:443]
  code: 'ENOTFOUND',
  errno: 'ENOTFOUND',
  syscall: 'getaddrinfo',
  hostname: 'registry.npmjs.org',
  host: 'registry.npmjs.org',
  port: 443,
  contentStream: undefined }
app debug [OK] Reusing cached result for https://registry.npmjs.org/bluebird

```

To install:

```
$ npm install -g npm_lazy
```

To start the server, run:

```
$ npm_lazy
```

Similar to local-npm, the registry can be set to the local server running as follows:

```
$ npm config set registry http://localhost:8080/
```

npm\_lazy has a relatively decent level of configuration, allowing customisation of the lifespan of the cache, maximum retries, HTTP timeouts and whether HTTPS requests are checked against Node's list of certificate authorities.

## Alternatives

There have been numerous other packages written that aim to offer a caching proxy for npm (such as [Sinopia](#)), but I've tried to keep the list short to avoid [choice paralysis](#).

## Bundling

In this post I've looked at offline package installation, but you may also be interested in offline package bundling. This is where you want to bundle a package and all of its dependencies into a single archive for sharing with a system that might be offline. This comes up a lot in workshops.

There's an [open issue](#) on npm to try improving the bundling story around [npm pack](#) and [bundledDependencies](#). Until this comes to fruition, I recommend trying out either:

- [Freight](#) - can bundle both npm and bower dependencies into a compressed archive
- [npmbox](#) - similarly supports creating/installing dependencies from a single archive
- [bundle-dependencies](#) - deeply bundles all module dependencies for a package into a monolithic package you would later publish to npm

---

### Subscribe to my newsletter for moar tips

Subscribe



Addy Osmani

Addy Osmani (Adnan Osmani) is an engineering manager at Google working on Chrome. His teams work on tools like Lighthouse and PageSpeed Insights. He is author of open-source projects like Yeoman, TodoMVC and Material Design Lite. He has also written books like Learning JavaScript Design Patterns with O'Reilly.

 [Tweet](#)  [Share](#)



