npm command to uninstall or prune unused packages in Node.js

Asked 5 years, 6 months ago Active 11 months ago Viewed 149k times

370

Is there a way to simply uninstall all unused(undeclared) dependencies from a Node.js project (ones that are no longer defined in my `package.json` .) When I update my application I like to have the unreferenced packages removed automatically.

node.js npm uninstall

57

edited Mar 14 '18 at 22:45

asked Jan 28 '14 at 21:15

Tarion
7,680 8 54 86

1 Unused by what? Do you mean to remove folders from `node_modules` when they're removed from the respective `package.json` ? – SLaks Jan 28 '14 at 21:19

1 exactly, mhm npm ll already gives a good hint which are the candidates. – Tarion Jan 28 '14 at 21:20

3 Answers

628

Run `npm prune` to unbuild modules not listed in `package.json` .

From `npm help prune` :

> This command removes "extraneous" packages. If a package name is provided, then only packages matching one of the supplied names are removed.
>
> Extraneous packages are packages that are not listed on the parent package's dependencies list.
>
> If the `--production` flag is specified, this command will remove the packages specified in your devDependencies.

edited Dec 9 '15 at 10:02

Mark Amery

answered Jan 28 '14 at 21:20

Darkhogg


70.9k 33 275 321


10.9k 4 18 24

2 If I read that correctly, this would remove all sub-dependencies, since they're not listed in `package.json`. Is that right? So, the next update or install would have to reinstall them. – N13 Apr 18 '14 at 14:14

1 Yes, it will remove sub-dependencies. Sub-dependencies are actually stored inside the module's own `node_modules` directory, so they are removed with the module. – Darkhogg Apr 18 '14 at 14:19

1 Let me give an example. I remove karma from my `package.json`, but leave bower. When I run `npm prune`, I expect all of karma, including its own `node_modules` folder containing its dependencies, to be removed. What about bower's dependencies (bower-json, bower-logger, chmodr, fstream, glob, et al.). Technically, those aren't listed in my project's `package.json`. Are those removed or not? – N13 Apr 18 '14 at 14:27

2 No, they are not. Note that they're *not* in your own `node_modules`, but *inside* `node_modules/bower/node_modules`, "protected" by `node_modules/bower/package.json`. Dependencies of your package and that of your package's dependencies *are not mixed*. – Darkhogg Apr 18 '14 at 14:30

2 **and** delete your shrinkwrap before npm install, should have been in above instructions. – Andy Ray May 4 '15 at 22:30

277

If you're not worried about a couple minutes time to do so, a solution would be to `rm -rf node_modules` and `npm install` again to rebuild the local modules.

answered Jan 28 '14 at 21:19


Pyrce
6,569 2 24 42

85 It would be nice if people stopped downvoting this without comment.. it's a valid strategy to resetting a node project dependencies as an alternative to the accepted answer. If you damaged a node_modules sub-directory contents (easy to do with sym-linked dependencies) or if you've had additional changes like node or npm version bumps prune will not properly cleanup the node_modules folder but this answer will. – Pyrce Jan 12 '16 at 19:06

36 Rebuilding `node_modules` also verifies the `package.json` file describes a reproducible dependency graph. Removing and re-installing your `node_modules` is basically a deploy test. – joemaller Jan 24 '16 at 18:26

2 @joemaller Not necessarily, most deployment workflows have, either implicitly or explicitly, some kind of cache. If a package is already installed and fits the specification, it's kept. Removing then reinstalling will bump that package(s) to the latest version that matches. – Darkhogg Mar 9 '16 at 14:15

6 `npm prune` didn't help one iota, but this did. My problem was a broken symlink. – Eirik Birkeland Apr 22 '16 at 0:34

7 Under many non-ideal circumstances that's currently infeasible with npm. Also the question definitely did not specify some constraint on repeated work or additional fetching, just how to achieve the end goal. This answer satisfies the question as stated, despite what others may want beyond that goal. – Pyrce May 4 '16 at 18:41

9

You can use npm-prune to remove extraneous packages.

```
npm prune [[<@scope>/]<pkg>...] [--production] [--dry-run] [--json]
```

This command removes "extraneous" packages. If a package name is provided, then only packages matching one of the supplied names are removed.

Extraneous packages are packages that are not listed on the parent package's dependencies list.

If the **--production** flag is specified or the **NODE_ENV** environment variable is set to **production**, this command will remove the packages specified in your **devDependencies**. Setting **--no-production** will negate **NODE_ENV** being set to **production**.

If the **--dry-run** flag is used then no changes will actually be made.

If the **--json** flag is used then the changes **npm prune** made (or would have made with **--dry-run**) are printed as a JSON object.

In normal operation with package-locks enabled, extraneous modules are pruned automatically when modules are installed and you'll only need this command with the **--production** flag.

If you've disabled package-locks then extraneous modules will not be removed and it's up to you to run **npm prune** from time-to-time to remove them.

Use npm-dedupe to reduce duplication

```
npm dedupe
npm ddp
```

Searches the local package tree and attempts to simplify the overall structure by moving dependencies further up the tree, where they can be more effectively shared by multiple dependent packages.

For example, consider this dependency graph:

```
a
+-- b <-- depends on c@1.0.x
|    `-- c@1.0.3
`-- d <-- depends on c@~1.0.9
     `-- c@1.0.10
```

In this case, **npm-dedupe** will transform the tree to:

```
a
+-- b
+-- d
`-- c@1.0.10
```

Because of the hierarchical nature of node's module lookup, b and d will both get their dependency met by the single c package at the root level of the tree.

The deduplication algorithm walks the tree, moving each dependency as far up in the tree as possible, even if duplicates are not found. This will result in both a flat and deduplicated tree.

answered Sep 6 '18 at 8:47

Igor Litvinovich
1,181 4 13