Do I commit the package-lock.json file created by npm 5?

Asked 2 years, 7 months ago Active 1 month ago Viewed 409k times

1201

184

npm 5 was released today and one of the new features include deterministic installs with the creation of a `package-lock.json` file.

Is this file supposed to be kept in source control?

I'm assuming it's similar to `yarn.lock` and `composer.lock`, both of which are supposed to be kept in source control.

node.js git npm version-control lockfile

asked May 26 '17 at 17:03
rink.attendant.6
28.2k 20 75 127

14 Short answer: yes. One comment: when package-lock.json changes you can make a commit of just that change, separate from other source changes. This makes `git log` easier to deal with. – Purplejacket Aug 29 '17 at 23:07

10 A file can't help produce a deterministic install if it doesn't exist. – Alan H. Sep 30 '17 at 6:11

2 Depends on the project. github.com/npm/npm/issues/20603 – Gajus May 13 '18 at 1:21

1 If you really trust npm sure, the purpose is to more explicitly report what the project is using. If you really want predictability ignore this file and instead install your node_modules (see .npmrc and related config in the answers+comment) and use that to track what's actually changing rather than what your package manager states it's doing. Ultimately: wich is more important? Your package manager or the code you're using. – jimmont Nov 30 '18 at 18:15

10 Answers

1417

Yes, `package-lock.json` is intended to be checked into source control. If you're using npm 5, you may see this on the command line: `created a lockfile as package-lock.json. You should commit this file.` According to `npm help package-lock.json`:

> `package-lock.json` is automatically generated for any operations where npm modifies either the `node_modules` tree, or `package.json`. It describes the exact tree that was generated, such that subsequent installs are able to generate identical trees,

✓

regardless of intermediate dependency updates.

This file is intended to be committed into source repositories, and serves various purposes:

- Describe a single representation of a dependency tree such that teammates, deployments, and continuous integration are guaranteed to install exactly the same dependencies.
- Provide a facility for users to "time-travel" to previous states of `node_modules` without having to commit the directory itself.
- To facilitate greater visibility of tree changes through readable source control diffs.
- And optimize the installation process by allowing npm to skip repeated metadata resolutions for previously-installed packages.

One key detail about `package-lock.json` is that it cannot be published, and it will be ignored if found in any place other than the toplevel package. It shares a format with npm-shrinkwrap.json(5), which is essentially the same file, but allows publication. This is not recommended unless deploying a CLI tool or otherwise using the publication process for producing production packages.

If both `package-lock.json` and `npm-shrinkwrap.json` are present in the root of a package, `package-lock.json` will be completely ignored.

edited Apr 24 '18 at 17:10

answered May 26 '17 at 22:16

vine77
15k 1 14 12

62 In what kind of projects is it actually helpful to commit the file? The whole point of semver and package.json is that updated compatible dependencies shouldn't need to be noted. – curiousdannii May 27 '17 at 12:05

39 The key word is "shouldn't need to be" - but in practice people don't follow semver perfectly. That's why you can use package-lock.json and package.json together to make it easy to update packages but still making sure every developer and every deployed application is using the same dependency tree. – Panu Horsmalahti May 31 '17 at 8:51

24 @trusktr: Sindre Sorhus recommends using "Lockfiles for apps, but not for packages." – vine77 Jun 23 '17 at 18:17

18 Another thing is, package-lock.json is ignored for publishing on NPM, so if a developer uses it for a library dev, then they are minimizing the chance that they will catch a regression from an updated dependency version, and therefore will pass that bug onto end users. For this reason, not using a lock file for library dev increases the chance of shipping less bugs. – trusktr Jul 19 '17 at 22:49

111 Personally I've now had to resort to adding `package-lock.json` to my `.gitignore` ... it was causing me far more problems than solving them. It always conflicts when we merge or rebase, and when a merge results in a `package-lock.json` being corrupted on the CI server, it's just a pain to have to stay fixing it. – Stefan Z Camilleri Aug 13 '17 at 18:37

101

Yes, it's intended to be checked in. I want to suggest that it gets its own unique commit. We find that it adds a lot of noise to our diffs.

answered Jun 16 '17 at 21:18
xer0x
10.7k 5 27 24

13 it's fair to debate whether it should be checked into your source code repository, but publishing this file to npm is not really up for debate - you must include either your package-lock.json or your shrinkwrap file into your npm registry. if you do not, your published package will be subject to unpinned changes of dependencies of your 1st generation dependencies. you won't notice this to be a problem until one of those 2nd+ generation dependencies publishes a breaking change, and your published package becomes mysteriously broken. this package-lock.json file was created to solve that problem. – guerillapresident Sep 10 '17 at 15:16

7 @BetoAveiga by noise I mean that the commits with package-lock.json can have so many lines of node package versions, that any other work in that commit becomes hidden. – xer0x Nov 25 '17 at 7:45

6 I usually keep package installations separate from other work. I never need to diff a commit like "Installed chai and mocha", because I already know what changed. – Keith Dec 14 '17 at 17:45

2 Any advice regarding the `package-lock.json` file when working on a SCM system with trunks and branching? I'm making some changes on a branch that need to be merged to trunk... do I now have to (somehow) resolve conflicts between the two `package-lock.json` files? This feels painful. – kmiklas Jan 23 '18 at 21:35

1 @guerillapresident As I understand it, you're partially correct. Publishing this file to npm is not up for debate. You can't publish it. – Tim Gautier May 10 '18 at 15:03

50

Yes, the best practice is to check-in (YES, CHECK-IN)

I agree that it will cause a lot of noise or conflict when seeing the diff. But the benefits are:

1. **guarantee exact same version of every package**. This part is the most important when building in different environments at different times. You may use `^1.2.3` in your `package.json` , but how can u ensure each time `npm install` will pick up the same version in your dev machine and in the build server, especially those indirect dependency packages? Well, `package-lock.json` will ensure that. (With the help of `npm ci` which installs packages based on lock file)
2. it improves the installation process.
3. it helps with new audit feature `npm audit fix` (I think the audit feature is from npm version 6).

edited Aug 19 at 3:27

answered Jun 15 '18 at 3:23
Xin

12.5k 6 52 53

1 As far as I know, never using semver (which npm devs dont understand anyway) should yield the same behavior as having a lockfile at least in 99% of cases. My own experience is that semver fuckups happen mostly with primary packages (direct dependencies, crappy jquery datepickers, etc). My personal experience with npm has been that lock files were noise forever. I hope this wisdom is unchanged with recent versions. – Svend Jun 25 '18 at 8:48

11 +1 for mentioning `npm ci` . People frequently mention that a `package-lock.json` allows a deterministic installation of packages but almost never mention the command that facilitates this behavior! Many people probably incorrectly assume `npm install` installs exactly what's in the lock file... – ahaurat Feb 12 at 0:18

npm ci isn't in npm 5. – dpurrington Apr 9 at 0:03

Thank you! It only makes sense to commit package-lock.json if you are using `npm ci` . Your team/lead developer can decide when to update. If everyone is just arbitrarily committing it, there is no point to it, and it's just creating noise in your repo. NPM documentation should make this more clear. I think most developers are just confused by this feature. – adampasz Aug 17 at 15:52

@adampasz actually each dev can commit the lock file, and once pass the testing and merged, the second branch just renew the lock file if somehow the packages get changed (we do not change the package.json often, we less facing this issue(– Xin Aug 19 at 3:25

34

I don't commit this file in my projects. What's the point ?

1. It's generated
2. It's the cause of a SHA1 code integrity err in gitlab with gitlab-ci.yml builds

Though it's true that I never use ^ in my package.json for libs because I had bad experiences with it.

edited Oct 4 at 8:19
ROMANIA_engineer
40.1k 22 172 156

answered Jul 12 '18 at 14:53
Deunz
804 10 18

9 I wish this could be expounded more from within npm docs - It would be useful to have an outline of what specifically you lose by not committing `package-lock.json` . Some repos may not require the benefits that come from having it, and may prefer to have no auto-generated content in source. – PotatoFarmer Oct 1 '18 at 22:32

I can see how it can be useful for debugging (a diff between two locks for example) to help resolve issues. I guess it can also be used to prevent these sort of things but it can also be a pain having it in a shared repo where it may experience merge conflicts due to it. For starters I want to keep things simple, I will just use package.json on its own until I see there is a real need for the package-lock.json. – radtek Mar 14 at 17:18

4 You may not use ^ at your package.json, but you can be sure your dependencies don't use it? – neiker May 15 at 18:25

34

Yes, you SHOULD:

1. **commit the `package-lock.json`.**
2. **use `npm ci` instead of `npm install`** when building your applications both on your CI and your local development machine

The `npm ci` workflow *requires* the existence of a `package-lock.json`.

A big downside of `npm install` command is its unexpected behavior that it may mutate the `package-lock.json`, whereas `npm ci` only uses the versions specified in the lockfile and produces an error

- if the `package-lock.json` and `package.json` are out of sync
- if a `package-lock.json` is missing.

Hence, running `npm install` locally, esp. in larger teams with multiple developers, may lead to lots of conflicts within the `package-lock.json` and developers to decide to completely delete the `package-lock.json` instead.

Yet there is a strong use-case for being able to trust that the project's dependencies resolve repeatably in a reliable way across different machines.

From a `package-lock.json` you get exactly that: a known-to-work state.

In the past, I had projects without `package-lock.json` / `npm-shrinkwrap.json` / `yarn.lock` files whose build would fail one day because a random dependency got a breaking update.

Those issue are hard to resolve as you sometimes have to guess what the last working version was.

If you want to add a new dependency, you still run `npm install {dependency}`. If you want to upgrade, use either `npm update {dependency}` or `npm install ${dependendency}@{version}` and commit the changed `package-lock.json`.

If an upgrade fails, you can revert to the last known working `package-lock.json`.

To quote npm doc:

> It is highly recommended you commit the generated package lock to source control: this will allow anyone else on your team, your deployments, your CI/continuous integration, and anyone else who runs npm install in your package source to get the exact same

> dependency tree that you were developing on. Additionally, the diffs from these changes are human-readable and will inform you of any changes npm has made to your node_modules, so you can notice if any transitive dependencies were updated, hoisted, etc.

And in regards to the difference between `npm ci` vs `npm install`:

> - The project must have an existing package-lock.json or npm-shrinkwrap.json.
> - If dependencies in the package lock do not match those in package.json, `npm ci` will exit with an error, instead of updating the package lock.
> - `npm ci` can only install entire projects at a time: individual dependencies cannot be added with this command.
> - If a `node_modules` is already present, it will be automatically removed before `npm ci` begins its install.
> - It will never write to `package.json` or any of the package-locks: installs are essentially frozen.

Note: I posted a similar answer here

edited Nov 11 at 11:55 answered May 22 at 10:14

k0pernikus
29.8k 34 136 226

5 This answer deserves more credit, especially using npm ci. Using this mitigates most of the issues people have experienced with package lock. – JamesB Jun 13 at 21:42

31

To the people complaining about the noise when doing git diff:

```
git diff -- . ':(exclude)*package-lock.json' -- . ':(exclude)*yarn.lock'
```

What I did was use an alias:

```
alias gd="git diff --ignore-all-space --ignore-space-at-eol --ignore-space-change --
ignore-blank-lines -- . ':(exclude)*package-lock.json' -- . ':(exclude)*yarn.lock'"
```

To ignore package-lock.json in diffs for the entire repository (everyone using it), you can add this to `.gitattributes`:

```
package-lock.json binary
yarn.lock binary
```

This will result in diffs that show "Binary files a/package-lock.json and b/package-lock.json differ whenever the package lock file was changed. Additionally, some Git services (notably GitLab, but not GitHub) will also exclude these files (no more 10k lines changed!) from the diffs when viewing online when doing this.

edited May 7 at 17:18

answered Jun 22 '18 at 7:04
Raza
1,055 1 12 25

I have `gd() { git diff --color-words $1 $2 -- :!/yarn.lock :!/package-lock.json; }` in my .bashrc instead of an alias. – apostl3pol May 14 at 21:47

15

Yes, you can commit this file. From the npm's official docs:

> `package-lock.json` is automatically generated for any operations where `npm` modifies either the `node_modules` tree, or `package.json`. It describes the exact tree that was generated, such that subsequent installs are able to generate identical trees, regardless of intermediate dependency updates.
>
> This file is intended to be committed into source repositories[.]

edited Jan 11 at 12:33
Martijn Pieters ♦
781k 177 2865 2563

answered Oct 6 '17 at 7:34
Bablu Singh
351 2 11

11 Won't an install always update node_modules, and therefore update package-lock.json? – Tim Gautier May 10 '18 at 15:01

1 No, you can run `npm ci` to install from the package-lock.json – William Hampshire Aug 28 at 15:00

Disable package-lock.json globally

type the following in your terminal:

3

```
npm config set package-lock false
```

this really work for me like magic

answered Oct 26 '18 at 17:01
Balogun Ridwan Ridbay
105 2

1 this creates `~/.npmrc` (at least on my macos) with content `package-lock=false` and the same can be done in any specific project alongside `node_modules/` (eg `echo 'package-lock=false' >> .npmrc` – jimmont Nov 30 '18 at 18:09

3 its sort of funny to me that this would be a negative. the npm community just can't accept that package-lock.json automatic generation was bad community involvement. you shouldn't do stuff that can impact a teams process. it should have been an option to enable, not forced. how many people just do "git add *" and not even notice it and screw up builds. If you have any kind of a merge based flow, I know git flow is like the bible to the people that use it, this won't work. you can't have generation on merge! npm versioning is broken, package : 1.0.0 should be deterministic! – Eric Twilegar Feb 5 at 22:25

1 why is this down-voted? this is clearly a legit way of disabling a feature that does not work. And although it does not answer the question per se, it moots the question. i.e. it no longer needs answering. Thumbs up from me :) – Superole Feb 6 at 16:34

The reason why it's getting downvoted is because you are simply disabling a feature. – Raza May 7 at 17:22

2

Yes, it's a standard practice to commit package-lock.json

The main reason for committing package-lock.json is that everyone in the project is on the same package version.

Pros:-

- Everyone in the project will be on the same package version, all you have to do is

  npm install

- If you follow strict versioning and don't allow updating to major versions automatically to save yourself from backward incompatible changes in third-party packages committing package-lock helps a lot.

Cons:-

- It can make your pull requests look ugly :)'

edited Jun 23 at 14:31 answered May 28 at 10:45

Nikhil Mohadikar
780 8 7

3 The cons would go away if you'd use `npm ci` instead of `npm install`. – k0pernikus Sep 6 at 15:49

1 Scope creeping a little, but here's [more info on that excellent advice from @k0pernikus](). – ruffin Dec 8 at 21:13

1

My use of npm is to generate minified/uglified css/js and to generate the javascript needed in pages served by a django application. In my applications, Javascript runs on the page to create animations, some times perform ajax calls, work within a VUE framework and/or work with the css. If package-lock.json has some overriding control over what is in package.json, then it may be necessary that there is one version of this file. In my experience it either does not effect what is installed by npm install, or if it does, It has not to date adversely affected the applications I deploy to my knowledge. I don't use mongodb or other such applications that are traditionally thin client.

I remove package-lock.json from repo because npm install generates this file, and npm install is part of the deploy process on each server that runs the app. Version control of node and npm are done manually on each server, but I am careful that they are the same.

When `npm install` is run on the server, it changes package-lock.json, and if there are changes to a file that is recorded by the repo on the server, the next deploy WONT allow you to pull new changes from origin. That is you can't deploy because the pull will overwrite the changes that have been made to package-lock.json.

You can't even overwrite a locally generated package-lock.json with what is on the repo (reset hard origin master), as npm will complain when ever you issue a command if the package-lock.json does not reflect what is in node_modules due to npm install, thus breaking the deploy. Now if this indicates that slightly different versions have been installed in node_modules, once again that has never caused me problems.

If node_modules is not on your repo (and it should not be), then package-lock.json should be ignored.

If I am missing something, please correct me in the comments, but the point that versioning is taken from this file makes no sense. The file package.json has version numbers in it, and I assume this file is the one used to build packages when npm install occurs, as when I remove it, npm install complains as follows:

```
jason@localhost:introcart_wagtail$ rm package.json
jason@localhost:introcart_wagtail$ npm install
npm WARN saveError ENOENT: no such file or directory, open
'/home/jason/webapps/introcart_devtools/introcart_wagtail/package.json'
```

and the build fails, however when installing node_modules or applying npm to build js/css, no complaint is made if I remove package-lock.json

```
jason@localhost:introcart_wagtail$ rm package-lock.json
jason@localhost:introcart_wagtail$ npm run dev

> introcart@1.0.0 dev /home/jason/webapps/introcart_devtools/introcart_wagtail
> NODE_ENV=development webpack --progress --colors --watch --mode=development

 10% building 0/1 modules 1 active ...
```

edited Jan 21 at 22:54

answered Jan 9 at 3:15

MagicLAMP
676 6 16