Contents

[Learning npm 2](#_Toc38656637)

[Initializing the package. json file 2](#_Toc38656638)

[Local vs global installation 2](#_Toc38656639)

[Dev dependencies 2](#_Toc38656640)

[Install packages globally 3](#_Toc38656641)

[Install react globally: 3](#_Toc38656642)

[CLI tool for creating react apps 3](#_Toc38656643)

[Install specific version of a package 3](#_Toc38656644)

[Removing a package 3](#_Toc38656645)

[Semantic versioning 3](#_Toc38656646)

[The package-lock .json file 3](#_Toc38656647)

[The npm cache 4](#_Toc38656648)

[npm-scripts 4](#_Toc38656649)

[Introduction to npx 4](#_Toc38656650)

# Learning npm

The current stable version of npm is [here](https://github.com/npm/cli/releases/latest). To upgrade, run: **npm install npm@latest -g**

Run packages without downloading using [npx](https://www.npmjs.com/package/npx).

The npm consists of the three following components:

* the website
* the Command Line Interface (CLI)
* the registry

When you install **node**, ***npm*** is automatically installed.

Search for router or date packages and install thru npm.

These packages can be installed on the machine where the code is exported. No need to bundle these packages in the source code. (with the help of dev dependencies) – look details below in the document.

*package. json* is like *pom.xml* – *NodeJS* is like *JRE* for server-side *JavaScript* – *ExpressJS* is a framework like *Spring boot* – *ReactJS* is a library for front end programming like *Angular* – JavaScript or TypeScript is a language like Java. *npm* manages the dependencies like *Maven*.

Check Node version: cmd: *node -v*

## Initializing the package. json file

*npm-init* is the base command to initialize the *package. json* file.

This creates a new *package.json* file.

### Local vs global installation

If a package is installed locally, it is installed within the project and if the package is installed globally, it is available for other projects on your system as it is installed on the system.

To install the express package:

cmd: *npm install express*

Once express package is installed, dependency is automatically added in *package. json* and a package-lock.json file is created. A node\_modules folder is also created where the express is installed.

## Dev dependencies

This install dependencies under *devDependencies* so they are not shipped with production but are used during development only.

npm install babel-cli babel-preset-stage-0 babel-preset-es2015 --save-dev

## Install packages globally

Add -g or -global to your command.

The modules will be installed in:

\AppData\Roaming\npm\node\_modules

%appData%

### Install react globally:

cmd: *npm install -g react*

### CLI tool for creating react apps

The following is the cli tool to create a new react app:

*npm install create-react-app -g*

## Install specific version of a package

*npm install eslint@5.2.0 -g* (Use @version\_number) in the command to install the specific version of a package.

*npm outdated* (check dependencies for update)

Check globally with -g

To update the packages: Use npm update or npm install

*npm install eslint@5.8.0 -g*

### Removing a package

npm uninstall babel-preset-es2015

*npm install babel-preset-env --save-dev*

## Semantic versioning

major\_release. minor\_release. patches (1.4.2)

^4.17.1

The ^ defines that whenever npm install is used to install the packages, the Major\_release.any\_minor\_latest\_release.any\_latest\_patches will be installed. (4.x.x).

~4.17.1 (Tilde means install 4.1.7.x (any latest patch)

For strict installs, remove the leading character from the package version.

### The package-lock .json file

In response of the package.json file, when the command is run npm install, a package-lock.json file is created which ensures that the accurate packages are installed every time on the exported machine.

Let’s say we have ^ with a package version in the package.json file so it may install the latest minor version or patch and the patch was old so this may create problem. For that, package-lock. Json comes into rescue.

### The npm cache

Clear the cache because npm caches the packages details.

Verify the cache: npm cache verify

To clear the cache, use the --force with the command because otherwise it won’t work.

Cmd: *npm cache clean --force*

npm audit is automatically run when a new package is installed and it checks the packages for issues.

*npm -v (check version)*

*npm audit*

## npm-scripts

<https://docs.npmjs.com/misc/scripts>

Install npm nodemon – it installs a package to restart the node server whenever we make changes in the code.

Add script in the package.json file and then use cmd: *npm start*

 "scripts": {

    "start": "nodemon ./index.js --exec babel-node -e js"

  },

*npm start* on command line is like running this above command in start at command prompt.

We can create out own script as well. Following is an example:

"scripts": {

    "myStartScript": "nodemon ./index.js --exec babel-node -e js"

  },

Run the above script thru: *npm run myStartScript*

## Introduction to npx

In order to use npx, which provides us the option to use packages without installing them globally on the system, we need to have a Node.js version greater than 5.2

Example: *npx -p @angular/cli ng new myTestApp*

*npx mocha (this command is used to run test cases used by mocha)*

*npx cowsay hello*

We can create scripts to run npx commands same as normal scripts and use *npm run your\_script*

**Yarn** is same as npm. It was introduced by Facebook.