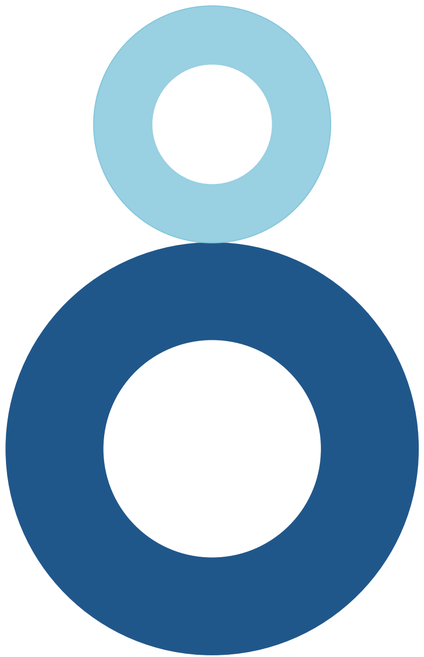
After studying this topic, you should be able to:

- Identify the three components of npm and how to find packages using the npm website.
- Identify the main npm CLI commands and how they can be used.
- Define the structure of the package.json file and any related considerations.
- Identify how to use semantic versioning for updating the version number of a package.

Introduction

Node Package Manager (npm) is the package manager for Node.js. It has three different components, namely, website, CLI, and registry. One can search for packages on the npm website and find information about them.



The npm CLI offers many useful commands for package management. The npm install <package-name> command allows installing a specific package. It can be installed as a development dependency using the --save-dev flag. The -g flag can be added to install it as a global package. The npm update <package-name> command allows updating the specified package. The npm outdated command can be used to check if any installed packages are outdated. The npm uninstall <package-name> command can be used to uninstall the specified package.

The package.json file lists the dependencies and devDependencies of the package. The name and version fields are required in the file. When publishing a new package, the version number should be updated in the file based on the semantic versioning spec.

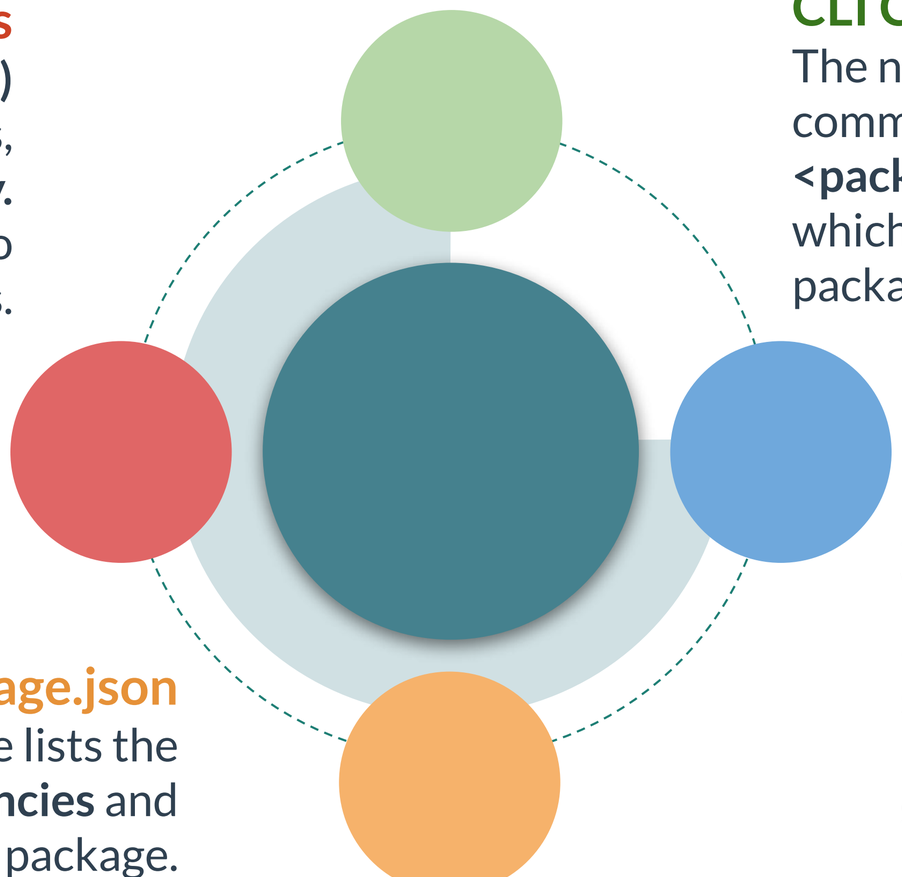
Overview

NPM Components

The **Node Package Manager (npm)** has three distinct components, namely, **website**, **CLI**, and **registry.** One can search on the website to discover npm packages.

CLI Commands

The npm CLI offers many useful commands such as **npm install <package-name>** and **npm update**, which can be used for managing the packages used by an application.



Semantic Versioning

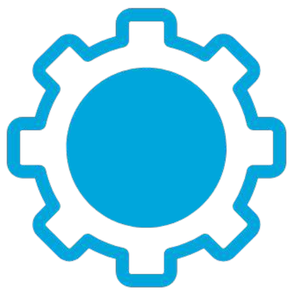
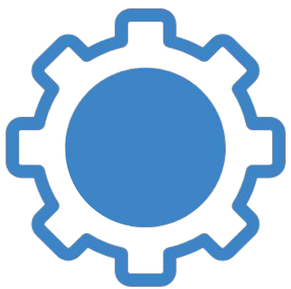
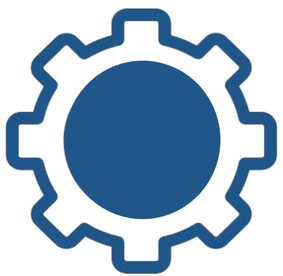
When publishing a new version of a package, the **version number** should be updated in the package.json file based on the **semantic versioning spec**.

package.json

The **package.json** file lists the **dependencies** and **devDependencies** of the package. The **name** and **version** fields are required in the file.

Node Package Manager (npm)

npm is the package manager for Node.js that allows **sharing** and **searching** for packages in an online database. It consists of three distinct components.

		
WEBSITE	**CLI**	**REGISTRY**
The **npm website (npmjs.com)** can be used to find JavaScript packages and information about them, set up profiles and organizations, etc.	The **Command-Line Interface(CLI)** allows interacting with npm using commands in the terminal. For example, it can be used to install an npm package.	The **registry** is the online database of hundreds of thousands of JavaScript packages and their metadata information.

Finding Packages

The search bar of the **npm website (npmjs.com)** can be used to search for packages. Each package is ranked according to four criteria, namely, **popularity**, **quality**, **maintenance**, and **optimal**.

Finding Packages

More information about a package, such as how to install and use it, can be viewed by selecting the package on the search results page.

validator

13.1.17 • Public • Published 2 months ago

Readme | Explore BETA | 0 Dependencies | 4,624 Dependents | 203 Versions

validator.js

npm v13.1.17 | build passing | codecov 100% | downloads 18M/month | backers 2 | sponsors 1

A library of string validators and sanitizers.

Strings only

This library validates and sanitizes strings only.

If you're not sure if your input is a string, coerce it using `input + ''`. Passing anything other than a string is an error.

Installation and Usage

Server-side usage

Install the library with `npm install validator`

No ES6

```
var validator = require('validator');
```

Install

```
> npm i validator
```

Weekly Downloads

4,182,073

Version	License
13.1.17	MIT
Unpacked Size	Total Files
467 kB	196
Issues	Pull Requests
95	32

Homepage

github.com/chriso/validator.js

Repository

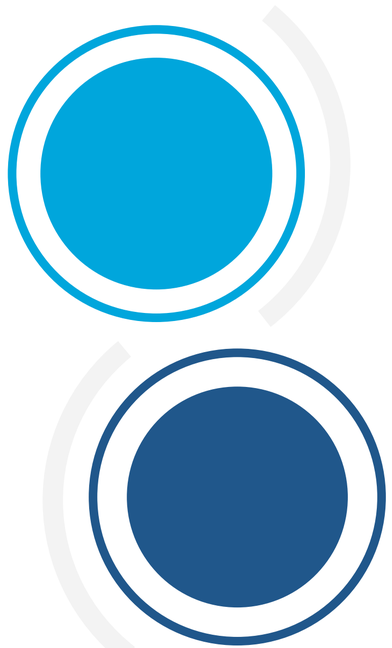
github.com/chriso/validator.js

Installing Packages

Once a suitable package is found, it can be installed either **locally** or **globally**.

LOCAL INSTALLATION

The **npm install <package-name>** command can be used to install a package locally. It should be installed locally in the project directory if only the application depends on it



GLOBAL INSTALLATION

The **npm install -g <package-name>** command can be used to install a package globally. It should be installed globally if the code needs to be used as a set of tools on the local computer.

Installing Packages

Once a suitable package is found, it can be installed either **locally** or **globally**.

TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE

```
PS C:\dev\node-app> npm install validator
+ validator@13.1.17
added 1 package from 2 contributors and audited 184 packages in 2.327s
```

The **npm install <package-name>** command can be used in the terminal to install a package locally. The command must be entered in the root of the project directory.

Installing Packages

Once a suitable package is found, it can be installed either **locally** or **globally**.

The **npm install -g <package-name>** command can be used in the terminal to install a package globally.

```
PS C:\dev\node-app> npm install -g chalk
+ chalk@4.1.0
added 6 packages from 3 contributors in 1.424s
```

Using Packages

Once a package has been installed, it can be used in the code by passing an argument to the **require()** function.

```
// This example shows how to use an npm package in the code by passing an argument to the require() function.

const validator = require('validator');

const value1 = '$999.99';
const value2 = 'Jon';
const isCurrency1 = validator.isCurrency(value1);
const isCurrency2 = validator.isCurrency(value2);
console.log(`${value1} ${isCurrency1 ? 'is' : 'is not'} a valid currency amount.`);
console.log(`${value2} ${isCurrency2 ? 'is' : 'is not'} a valid currency amount.`);
```

Output

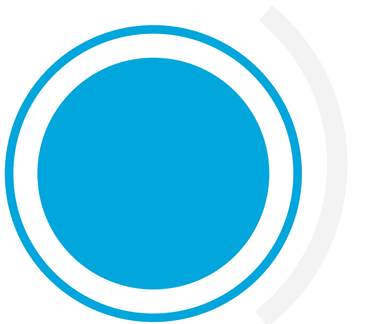
```
$999.99 is a valid currency amount.
Jon is not a valid currency amount.
```

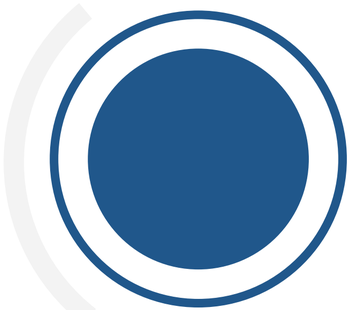
Updating Packages

Local and global packages that have been downloaded from the npm registry can be updated. It is also possible to check which packages are outdated.

UPDATE PACKAGE

The **npm update <package-name>** command can be used in the root directory of the project to update the specified package. The **-g** flag can be used to update a global package.





OUTDATED PACKAGES

The **npm outdated** command can be used to check the npm registry to determine if any installed packages are outdated. A specific package can also be used to check if that package is outdated.

Uninstalling Packages

The **npm uninstall <package-name>** command can be used to uninstall a package.

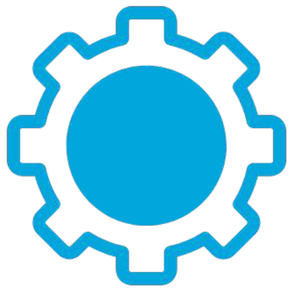
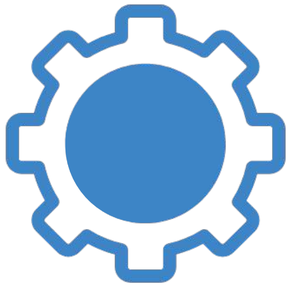
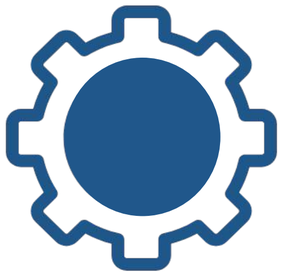
TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE

```
PS C:\dev\node-app> npm uninstall validator
removed 1 package and audited 183 packages in 1.453s
```

The **npm uninstall <package-name>** command can be used in the terminal to uninstall a package. The **-g** flag can be used to uninstall a global package.

The package.json File

The **package.json** file contains information about the package. A package must contain a package.json file if it needs to be published to the npm registry.

		
INFORMATION	**INSTALLATION**	**MODULES**
The package.json file contains information such as **name**, **description**, **version**, etc. It also lists the **package dependencies**, including the **development dependencies**.	The package.json file makes it easier for others to manage and install the package. The **npm install** command allows installing all the dependencies.	**Not** all **Node.js modules** are **packages**. A module is either a JavaScript file or a folder with a package.json file containing a 'main' field.

Structure of the package.json File

The **package.json** file contains various fields. The **name** and **version** fields are required. To publish a package, the name must be **unique** and follow the **npm policy guidelines**. Information such as **author**, **email**, and **website** can be specified.

```
// This example shows a package.json file.
{
  "name": "web-app",
  "description": "",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "John Wayne <jwayne@jwayne.com> (http://www.jwayne.com)",
  "license": "ISC",
  "dependencies": {
    "validator": "^13.1.17"
  }
}
```

Creating a package.json File

A **package.json** file can be created by using the **npm init** command in the root of the project directory. It requires answering the questions in the command line questionnaire. A default file can be created by using the **-y** or **--yes** flag.

```
PS C:\dev\example-app> npm init -y
Wrote to C:\dev\example-app\package.json:

{
  "name": "example-app",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

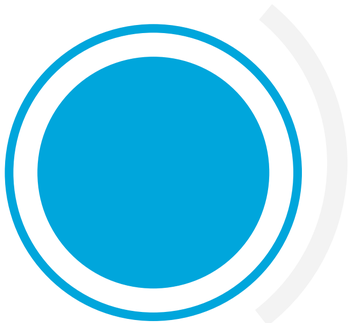
The **npm init -y** command can be used in the root of the project directory to create a default package.json file.

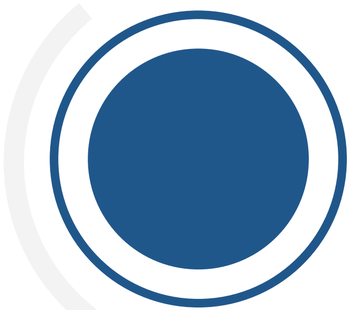
Dependencies

The packages that the project depends on are listed under **dependencies** or **devDependencies** in the package.json file.

DEPENDENCIES

Packages listed under **dependencies** are those that are required by the application in production.





DEVELOPMENT DEPENDENCIES

Packages listed under **devDependencies** are those that are only needed for local development and testing.

Dependencies

To save a package under **dependencies**, the **npm install <package-name>** or **npm install <package-name> --save-prod** command should be used. To save it under **devDependencies**, the **--save-dev** flag should be used instead.

```
// After executing the command below, the 'axios' package is listed under 'dependencies' in the package.json file.
{
  "name": "example-app",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "axios": "^0.21.0"
  }
}
```

```
PS C:\dev\example-app> npm install axios --save-prod
```

Dependencies

To save a package under **dependencies**, the **npm install <package-name>** or **npm install <package-name> --save-prod** command should be used. To save it under **devDependencies**, the **--save-dev** flag should be used instead.

```
// After executing the command below, the 'jest' package is listed under 'devDependencies' in the package.json file.
{
  "name": "example-app",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "devDependencies": {
    "jest": "^26.6.3"
  }
}
```

```
PS C:\dev\example-app> npm install jest --save-dev
```

Semantic Versioning

When a new version of a package is published, it should be updated with an updated **version number** in the **package.json** file that follows the **semantic versioning spec**.

NEW PRODUCT The package version should start at **1.0.0** for the **first release** of a package. It should then be incremented based on the type of the release.	**PATCH RELEASE** In case of a **patch release** which has backward compatible bug fixes, the third digit of the version number should be incremented (**1.0.1**).	**MINOR RELEASE** In case of a **minor release** which has backward compatible new features, the middle digit should be incremented and the last digit should be set to zero (**1.1.0**).	**MAJOR RELEASE** In case of a **major release** which breaks backward compatibility, the first digit should be incremented and middle and last digits should be set to zero (**2.0.0**).

Versioning Operators

A **versioning operator** can be used to specify which **update types** a package can accept from **dependencies** in the package's **package.json** file.

PRIMITIVE The **primitive operators (<, <=, >, >=, =, and -)** can be used to specify version numbers of a package that satisfy **a particular range**.	**TILDE (~)** The **tilde symbol (~)** can be used to specify that updates should be accepted if the version number of a package is **greater** than a particular version in the **same minor range**.	**CARAT (^)** The **carat symbol (^)** can be used to specify that updates should be accepted if the version number of a package is a particular version in the **same major range**.	**X-RANGES** When specifying a range, **X**, **x** or * can be used as a placeholder for the numeric value of a **major**, **minor** or **patch** release in the **version number**.

Versioning Operators

A **versioning operator** can be used to specify which **update types** a package can accept from **dependencies** in the package's **package.json** file.

```
/* This example shows the 'dependencies' in a package.json file. Versioning operators of different types are used to specify the acceptable update types. */

"dependencies": {
  "lodash": ">=2.3.0 <3.2.0", // Any version number from 2.3.0 to 3.2.0 (not including 3.2.0) is acceptable.
  "chalk": "~2.2.0", // Any version number from 2.2.0 to 2.3.0 (not including 2.3.0) is acceptable.
  "express": "^3.16.0", // Any version number from 3.16.0 to 4.0.0 (not including 4.0.0) is acceptable.
  "moment": "1.x" // Any version number that starts with 1 (but not 2, such as 2.0.1) is acceptable.
}
```

Learn More

- npmjs
- Packages and Modules
- npm CLI Commands
- semver
- npm semver calculator

Scenario and Solution

Scenario

A developer who has built a web application using Node.js is using ES2015 features extensively in JavaScript code. These include classes, arrow functions, and the const keyword.

In order to convert the ES2015 code into backwards compatible version of JavaScript in current and older browsers, she would like to use a transpiler called 'Babel'. But it should only be installed for the purpose of development since code transformations are not necessary for running the application in production.

Also, she has installed a lot of other packages which are required for running the application in production. She would first like to check the npm registry to determine if any installed packages are outdated. Then she would like to update the outdated packages.

Lastly, she would like to ensure that the application follows the semantic versioning spec. Its current version is 1.1.4. For the next minor release, she needs to use the correct version number in the package.json file of the application.

Solution

In order to install the transpiler called 'Babel' as a development dependency, the `npm install @babel/core --save-dev` command can be used, as shown below.

```
npm install @babel/core --save-dev
```

To check if there are any installed packages that are outdated, the `npm outdated` command can be used.

```
PS C:\dev\node-app> npm outdated
Package   Current   Wanted   Latest   Location
webpack     5.3.2    5.4.0    5.4.0   node-app
```

The `npm update` command can be used to update the outdated packages.

```
PS C:\dev\node-app> npm update
+ webpack@5.4.0
updated 1 package and audited 184 packages in 3.722s
```

For the next minor release of the application, the version number of the package should be set to `1.2.0`, according to the semantic versioning spec. That's because the middle digit should be incremented and the last digit should be set to zero.