2.3 NODE PACKAGE MANAGER

NPM (Node Package Manager) is a package manager for Node.js modules. It helps developers manage project dependencies, scripts, and third-party libraries. By installing Node.js on your system, NPM is automatically installed, and ready to use.

- It is primarily used to manage packages or modules—these are pre-built pieces of code that extend the functionality of your Node.js application.
- The NPM registry hosts millions of free packages that you can download and use in your project.
- NPM is installed automatically when you install Node.js, so you don't need to set it up manually.

How to Use NPM with Node.js?

To start using NPM in your project, follow these simple steps

Step 1: Install Node.js and NPM

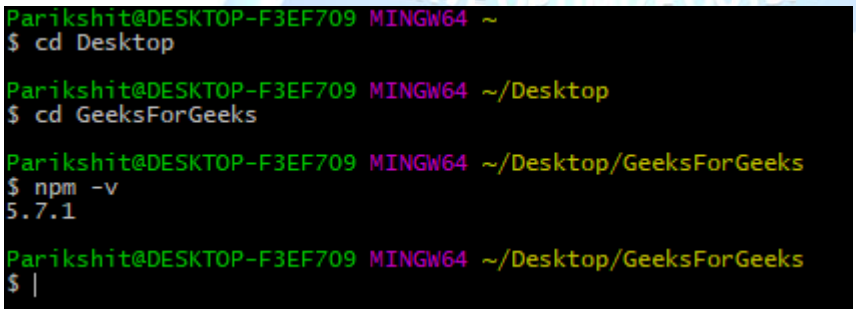
First, you need to install Node.js. NPM is bundled with the Node.js installation. You can follow our article to Install the Node and NPM- How to install Node on your system

Step 2: Verify the Installation

After installation, verify Node.js and NPM are installed by running the following commands in your terminal:

node -v
npm -v

These commands will show the installed versions of Node.js and NPM.



NodeJS NPM Version

Step 3: Initialize a New Node.js Project

In the terminal, navigate to your project directory and run:

npm init -y

This will create a package.json file, which stores metadata about your project, including dependencies and scripts.

Step 4: Install Packages with NPM

To install a package, use the following command

npm install <package-name>

For example, to install the Express.js framework

npm install express

This will add express to the node_modules folder and automatically update the package.json file with the installed package information.

Step 5: Install Packages Globally

To install packages that you want to use across multiple projects, use the -g flag:

npm install -g <package-name>

Step 6: Run Scripts

You can also define custom scripts in the package.json file under the "scripts" section. For example:

```
{
  "scripts": {
    "start": "node app.js"
  }
}
```

Then, run the script with

npm start

Using NPM Package in the project

Create a file named **app.js** in the project directory to use the package

```
//app.js
const express = require('express');//import the required package
const app = express();
app.get('/', (req, res) => {
  res.send('Hello, World!');
```

```
});
app.listen(3000, () => {
   console.log('Server running at http://localhost:3000');
});
```

- **express()** creates an instance of the Express app.
- **app.get()** defines a route handler for HTTP GET requests to the root (/) URL.
- **res.send()** sends the response "Hello, World!" to the client.
- **app.listen(3000)** starts the server on port 3000, and console.log() outputs the server URL.

Now run the application with

node app.js

Visit **http://localhost:3000** in your browser, and you should see the message: **Hello, World!**

Managing Project Dependencies

1. Installing All Dependencies

In a Node.js project, dependencies are stored in a package.json file. To install all dependencies listed in the file, run:

npm install

This will download all required packages and place them in the node_modules folder.

2. Installing a Specific Package

To install a specific package, use:

npm install <package-name>

You can also install a package as a development dependency using:

npm install <package-name> --save-dev

Development dependencies are packages needed only during development, such as testing libraries.

To install a package and simultaneously save it in package.json file (in case using Node.js), add **–save** flag. The **–save** flag is default in npm install command so it is equal to **npm install package_name** command.

Example:

npm install express --save

Usage of Flags:

- **–save:** flag one can control where the packages are to be installed.
- **–save-prod :** Using this packages will appear in Dependencies which is also by default.
- **–save-dev :** Using this packages will get appear in devDependencies and will only be used in the development mode.

Note: If there is a package.json file with all the packages mentioned as dependencies already, just type npm install in terminal

3. Updating Packages

You can easily update packages in your project using the following command

npm update

This will update all packages to their latest compatible versions based on the version constraints in the package.json file.

To update a specific package, run

npm update <package-name>

4. Uninstalling Packages

To uninstall packages using npm, follow the below syntax:

npm uninstall <package-name>

For uninstall Global Packages

npm uninstall package_name -g

Popular NPM Packages

NPM has a massive library of packages. Here are a few popular packages that can enhance your Node.js applications:

- **Express**: A fast, minimal web framework for building APIs and web applications.

- **Mongoose**: A MongoDB object modeling tool for Node.js.
- **Lodash**: A utility library delivering consistency, customization, and performance.
- **Axios**: A promise-based HTTP client for making HTTP requests.
- **React**: A popular front-end library used to build user interfaces