

Node.js

- **Getting Started with Node.js:**

- Understanding Node.js
- Installing Node.js
- Working with Node Packages
- Creating a Node.js Application
- Writing Data to the Console

About Node.Js

- Node.js is an open-source and cross-platform **JavaScript runtime environment**, that runs the **V8 JavaScript engine**, the core of Google Chrome, outside of the browser.
- A Node.js app runs in a single process, without creating a new thread for every request. Node.js provides a set of **asynchronous I/O** primitives in its standard library that prevent JavaScript code from blocking(**i.e., Non-blocking**)
- When Node.js performs an I/O operation, like reading from the network, accessing a database or the filesystem, instead of blocking the thread and wasting CPU cycles waiting, Node.js will resume the operations when the response comes back.
- This allows Node.js to handle thousands of concurrent connections with a single server without introducing the burden of managing thread concurrency, which could be a significant source of bugs.
- For CPU intensive requests ,it is not as good as I/O intensive requests.

Differences Browser vs. node.js(V8 engine)

- In the browser, most of the time is interacting with the DOM, or other Web Platform APIs like Cookies.
- From the perspective of a frontend developer who extensively uses JavaScript, Node.js apps bring with them a huge advantage: the comfort of programming everything - the frontend and the backend - in a single language.

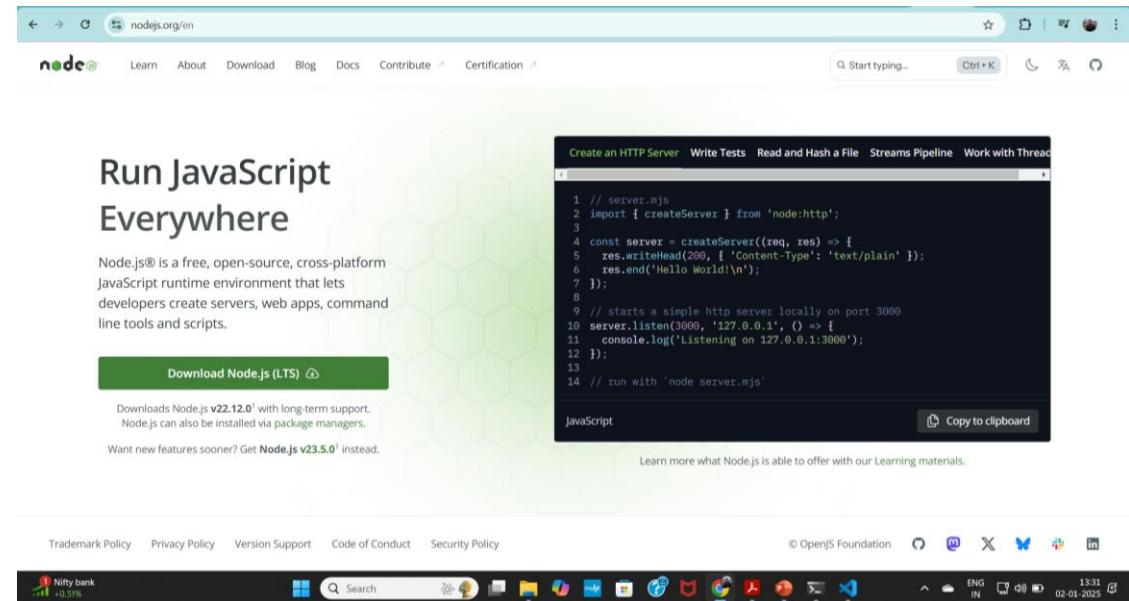
In browser	In node.js
Interact with DOM and APIs like Cookies	No need to modify DOM, and no window and document objects.
Rich set of APIs are not provided.	Provided APIs as modules.
Java script version Compatibility	No transformation . Supports from ES2015+ onwards. Uses babel to transform the code compatible with ES5-compatible
Supports CommonJS and ES modules standard being implemented.	Supports both CommonJS and ES(ECMAScript) modules

V8 Engine

- V8 is the name of the JavaScript engine that powers Google Chrome, which parses and executes javascript code in chrome browser.
- It is independent of the browser in which it is hosted, which becomes the part of node.js in 2009 and its evolution.
- JavaScript is internally compiled by V8 with **just-in-time (JIT) compilation** to speed up the execution.
- V8 is both compiler and interpreter ,it uses **TurboFan compiler** to translate bytecode to machine code and **ignition(interpreter)** to generate bytecode from Abstract Syntax tree.
- Other browsers has the similar javascript engines
 - Firefox has **SpiderMonkey**
 - Safari has **JavaScriptCore**(known as nitro)
 - Edge was originally based on **Chakra** but has more recently been rebuilt using **Chromium** and the **V8** engine.

Installation of Nodejs

Step 1: download latest version of nodejs from nodejs.org and install in your computer

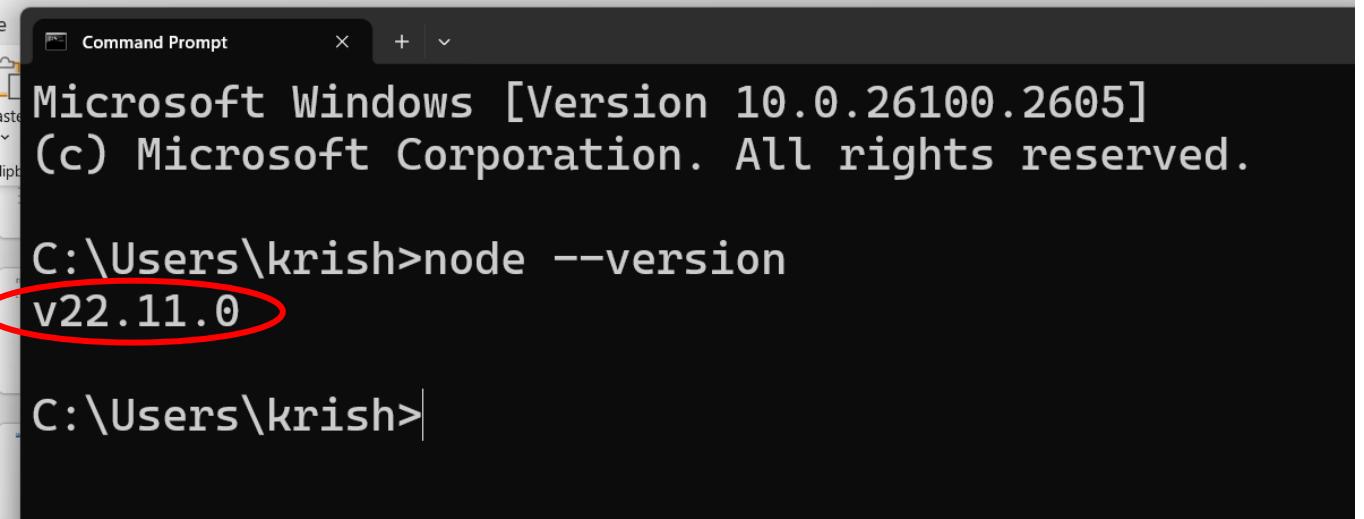


After successful installation, nodejs folder is created in the directory C:\Program Files\nodejs (in my system, it may be differ in your system depends on the drive you selected at the time of installation)

Check, the version of nodejs from command promptc

Check the version of nodejs and npm(node package manager)

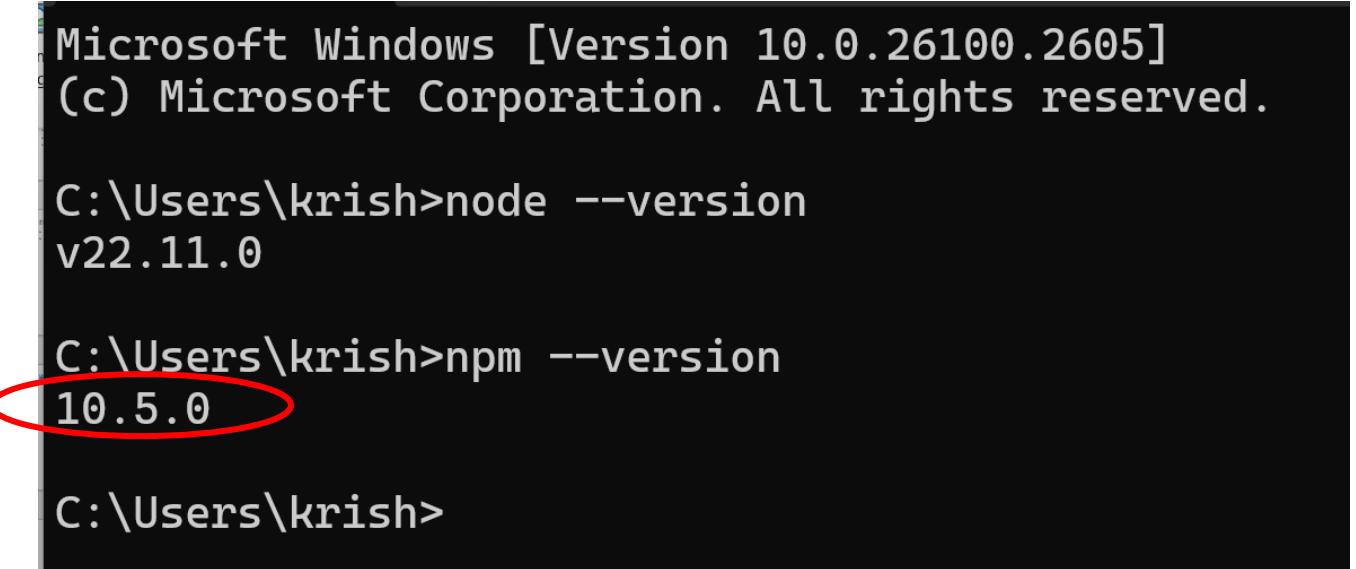
Note: include the path in edit environment variable, if command is not working



```
Microsoft Windows [Version 10.0.26100.2605]
(c) Microsoft Corporation. All rights reserved.

C:\Users\krish>node --version
v22.11.0

C:\Users\krish>
```



```
Microsoft Windows [Version 10.0.26100.2605]
(c) Microsoft Corporation. All rights reserved.

C:\Users\krish>node --version
v22.11.0

C:\Users\krish>npm --version
10.5.0

C:\Users\krish>
```

npm(node package manager)

- npm is the standard javascript package manager for node.js.
- It allows you to find, install, remove, publish, and do everything else related to Node Package Modules. The *npm* provides the link between the Node Package Registry and your development environment.

Option	Description	Example
search	Finds Module packages in the repository	npm search express
install	Installs a package either using a package.json file, from the repository, or a local location	npm install (or) npm install express
pack	Packages the module defined by the package.json file into a .tgz file	npm pack
publish	publishes the module defined by a package.json file to the registry	npm publish
unpublish	Unpublishes a module you have published	npm unpublish
update	Updates the specified package name	npm update or npm update <package-name>

What is Module?

- A module is any file or directory in the node_modules directory that can be loaded by the Node.js require() function.
- To be loaded by the Node.js require() function, a module must be one of the following:
 - A folder with a package.json file containing a "main" field.
 - A JavaScript file.

Built-in modules in Node.js

assert	Provides a set of assertion tests
buffer	To handle binary data
child_process	To run a child process
cluster	To split a single Node process into multiple processes
crypto	To handle OpenSSL cryptographic functions
dgram	Provides implementation of UDP datagram sockets
dns	To do DNS lookups and name resolution functions
domain	Deprecated. To handle unhandled errors
events	To handle events
fs	To handle the file system
http	To make Node.js act as an HTTP server
https	To make Node.js act as an HTTPS server.

Built-in modules

net	To create servers and clients
os	Provides information about the operation system
path	To handle file paths
punycode	Deprecated. A character encoding scheme
querystring	To handle URL query strings
readline	To handle readable streams one line at the time
stream	To handle streaming data
tls	To implement TLS and SSL protocols
tty	Provides classes used by a text terminal
url	To parse URL strings
util	To access utility functions
v8	To access information about V8 (the JavaScript engine)
vm	To compile JavaScript code in a virtual machine
Zlib	To compress or decompress files
string_decoder	To decode buffer objects into strings
timers	To execute a function after a given number of milliseconds

Importing modules using require()

Modules can be of CJS(common) or ESM(ECMAScript Module). By default, Node.js treats all modules as CJS

- Local modules - with in the same package require('module_name')
- Non-local modules
 - core module - comes with along nodejs installation (example: (example : os, fs, path, http, url, etc..))
 - non-core module - install explicitly using npm and import using require (example: express, nodemon, react, MongoDB etc.,)

// importing a module from local file system

```
const myLocalModule = require('./path/myLocalModule');
```

// Importing a module from node_modules or Node.js built-in module:

```
const crypto = require('node:os');
```

// Importing a JSON file:

```
const jsonData = require('./path/filename.json');
```

file1.js

```
let a=[10,20];  
  
let b="hello world";  
  
let c={id:101,name:"xyz"};  
  
let greeting=function(name){  
  
    return `have a good day ${name}`;  
  
}  
  
module.exports={a,b,c,greeting};
```

Build and Import local modules

Exports make the data objects or functions accessible outside the file, in which it is imported.

file2.js

```
const test=require('./file1');  
  
console.log(test.a);  
  
console.log(test.b);  
  
console.log(test.c);  
  
console.log(test.greeting("krishna"));
```



require function is used to import the module

test is the object created in file2 for all the exports in file1 and can be accessed using .(dot).

```
let a=[10,20];  
let b="hello world";  
let c={id:101,name:"xyz"};  
let greeting=function(name){  
    return `have a good day ${name}`;  
}  
module.exports={a,b,c,greeting};
```

Functions can be exported at time of definition

```
exports.greeting=function(name){  
    return `have a good day ${name}`;  
}
```

(or)

exporting in a single group

(or)

```
module.exports.a=a;  
module.exports.b=b;  
module.exports.c=c;  
module.exports.greeting=greeting;
```

exporting individually by giving a name to each object exported.

Build and Import built-in modules

```
const path = require('path');
const filePath="C:\\temp\\myfile.html";
console.log(path.dirname(filePath));
console.log(path.basename(filePath));
console.log(path.extname(filePath));
```

Creating a package.json file

To create a default package.json, go to directory

```
$ npm init --yes
```

```
PS E:\2024-25\Mern\nodejs1> npm init -y
```

```
Wrote to E:\2024-25\Mern\nodejs1\package.json:
```

```
{
  "name": "nodejs1",
  "version": "1.0.0",
  "description": "",
  "main": "sample.js",
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

- name: the current directory name
- version: always 1.0.0
- description: info about the package, or an empty string ""
- main: name of .js file
- scripts: by default creates an empty test script
- keywords: empty
- author: empty
- license: [ISC](#)

Creating NodeJs Application

1. Create a project folder named . . . /Numbers. This is the root of the package.

```
E:\2024-25\Mern>mkdir Numbers  
E:\2024-25\Mern>cd Numbers  
E:\2024-25\Mern\Numbers>
```

Step 2: Create the file that will be loaded when your module is required by another application

```
function isEven(n){  
    return n%2==0;  
}  
function isOdd(n){  
    return n%2==1;  
}  
function isPrime(n){  
    for(var i=2;i<parseInt(Math.sqrt(n));i++)  
    {  
        if(n%i==0)  
            return false;  
    }  
    return true;  
}  
exports.isEven=isEven;  
exports.isOdd=isOdd;  
exports.isPrime=isPrime;
```

Step 3: create package.json in the root folder

E:\2024-25\Mern\Numbers> npm init

Enter the details as prompting

```
{  
  "name": "numbers",  
  "version": "1.0.0",  
  "description": "identifying prime,even,odd, etc...",  
  "main": "numbercategory.js",  
  "scripts": {  
    "test": "echo \\\"Error: no test specified\\\" && exit 1"  
  },  
  "keywords": [  
    "prime",  
    "even",  
    "odd"  
  ],  
  "author": "krishnalikki",  
  "license": "ISC"  
}
```

Step 4: Create a file named `README.md` in the `.../censorify` folder. You can put whatever read me instructions you want in this file.

publish it in npm registry

It can be done in 2 ways

1. Using git (by adding repositories attribute in package.json file)
2. Without using git.

Without using git

Step 5: Create an account in npm registry at <https://npmjs.org/signup>.

Use the `npm adduser` command from a console prompt to add the user you created to the environment.
Type in the username, password, and email that you used to create the account.

Step 6: Publish the module using the following command from the folder in the console:

```
npm publish
```

1. Using git (by adding repositories attribute in package.json file

Add repository attribute to package.json file.

```
{  
  "name": "numbers",  
  "version": "1.0.0",  
  "description": "identifying prime,even,odd, etc...",  
  "main": "numbercategory.js",  
  "scripts": {  
    "test": "echo \\\"Error: no test specified\\\" && exit 1"  
  },  
  "repository": {  
    "type": "git",  
    "url": "git+https://github.com/krishnalikki/folder1.git"  
  },  
  "keywords": [  
    "prime",  
    "even",  
    "odd"  
  ],  
  "author": "krishnalikki",  
  "license": "ISC"  
}
```

Errors

```
npm notice Publishing to https://registry.npmjs.org/ with tag latest and default access
npm ERR! code E403
npm ERR! 403 Forbidden - PUT https://registry.npmjs.org/numbers - You do not have permission to publish "numbers". Are you logged in
as the correct user?
npm ERR! 403 In most cases, you or one of your dependencies are requesting
npm ERR! 403 a package version that is forbidden by your security policy, or
npm ERR! 403 on a server you do not have access to.
```



Resolution:

name in the package.json is not unique. So, give a unique name by prefixing username to the name of the package

```
{
  "name": "numbers",
  "version": "1.0.0",
  "description": "identifying prime,even,odd, etc...",
  "main": "numbercategory.js",
  ▶ Debug
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [
    "prime",
    "even",
    "odd"
  ],
}
```

```
{
  "name": "@krishnalikki/numbers",
  "version": "1.0.0",
  "description": "identifying prime,even,odd, etc...",
  "main": "numbercategory.js",
  ▶ Debug
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [
    "prime",
    "even",
    "odd"
  ],
}
```

```
npm notice Publishing to https://registry.npmjs.org/ with tag latest and default access
npm ERR! code E402
npm ERR! 402 Payment Required - PUT https://registry.npmjs.org/@krishnalikki%2fnumbers - You must sign up for private packages
npm ERR! A complete log of this run can be found in: C:\Users\krish\AppData\Local\npm-cache\_logs\2025-01-03T05_20_08_421Z-debug-0.log
PS E:\2024-25\Mern\Numbers> 
```

Change the access to public, because by default it is a private package

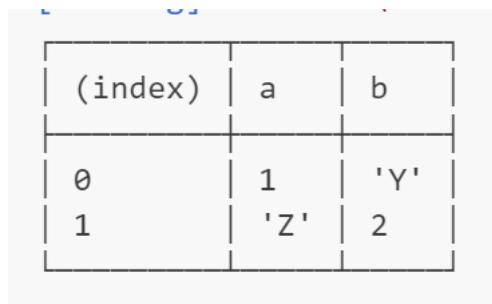
```
E:\2024-25\Mern\Numbers> npm publish --access=public
```

Writing data to console

console is the default module , no need to import using require ()

```
x=10;  
console.log("the value of x is" , x);  
the value of x is 10
```

```
console.table([ { a: 1, b: 'Y' }, { a: 'Z', b: 2 }]);
```

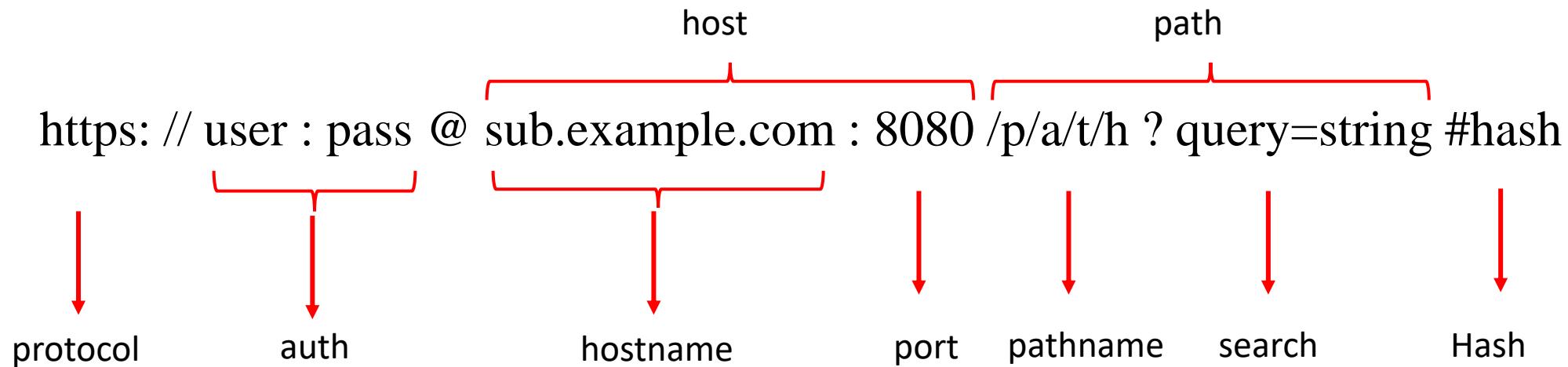


A screenshot of a browser's developer tools console window. It displays the output of the `console.table()` command. The output is a table with three rows and three columns. The first row contains column headers: '(index)' in the first column, 'a' in the second, and 'b' in the third. The second row contains data for index 0: '0' in the first column, '1' in the second, and "'Y'" in the third. The third row contains data for index 1: '1' in the first column, "'Z'" in the second, and '2' in the third.

(index)	a	b
0	1	'Y'
1	'Z'	2

URL(Uniform Resource Locator) module

A URL string is a structured string containing multiple meaningful components. When parsed, a URL object is returned containing properties for each of these components.



Import an url module

address

```
const url = require('url');
```

```
var addr = 'http://localhost:8080/default.htm?year=2024&month=february';
```

Parsing the address

```
//Parse the address:
```

```
var q = url.parse(addr,true);
```

The parse method returns an object containing url properties

```
url {
```

```
  protocol: 'http:',
```

```
  slashes: true,
```

```
  auth: null,
```

```
  host: 'localhost:8080',
```

```
  port: '8080',
```

```
  hostname: 'localhost',
```

```
  hash: null,
```

```
  search: '?year=2024&month=february',
```

```
  query: [Object: null prototype] { year: '2024', month: 'february' },
```

```
  pathname: '/default.htm',
```

```
  path: '/default.htm?year=2024&month=february',
```

```
  href: 'http://localhost:8080/default.htm?year=2024&month=february'
```

```
}
```

Properties of url

Creating a sample http server

```
const { createServer } = require('node:http');

const hostname = '127.0.0.1';
const port = 3000;

const server = createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello World');
});

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

The `createServer()` method of `http` creates a new HTTP server and returns it.

The server is set to listen on the specified port and host name. When the server is ready, the **callback function** is called, in this case informing us that the server is running.

Whenever a new request is received, the [request event](#) is called, providing two objects:

- `request` (an [http.IncomingMessage](#) object) and
- `response` (an [http.ServerResponse](#) object).

npm

- npm is the standard javascript package manager for node.js.
- It installs, updates and manages downloads of dependencies of your project.
Dependencies are pre-built pieces of code, such as libraries and packages, that your Node.js application needs to work.

If the project has package.json file ,

npm install

will install everything the project needs, in the **node_modules** folder, creating it if it's not existing already.

Installing a single package

You can also install a specific package by running

npm install <package-name>

adds **<package-name>** to the package.json file.

- It allows you to find, install, remove, publish, and do everything else related to Node Package Modules.

The Node Package Manager provides the link between the Node Package Registry and your development environment.

Option	Description	Example
search	Finds Module packages in the repository	npm search express
install	Installs a package either using a package.json file, from the repository, or a local location	npm install (or) npm install express
pack	Packages the module defined by the package.json file into a .tgz file	npm pack
publish	publishes the module defined by a package.json file to the registry	npm publish
unpublish	Unpublishes a module you have published	npm unpublish
update	Updates the specified package name	npm update