MDN Web Docs moz://a

Setting up a Node development environment

Now that you know what Express is for, we'll show you how to set up and test a Node/Express development environment on Windows, Linux (Ubuntu), and macOS. Whatever common operating system you are using, this article should give you what you need to be able to start developing Express apps.

Prerequisites:	Know how to open a terminal / command line. Know how to install software packages on your development computer's operating system.
Objective:	To set up a development environment for Express (X.XX) on your computer.

Express development environment overview

Node and *Express* make it very easy to set up your computer in order to start developing web applications. This section provides an overview of what tools are needed, explains some of the simplest methods for installing Node (and Express) on Ubuntu, macOS, and Windows, and shows how you can test your installation.

What is the Express development environment?

The *Express* development environment includes an installation of *Nodejs*, the *NPM package manager*, and (optionally) the *Express Application Generator* on your local computer.

Node and the *NPM* package manager are installed together from prepared binary packages, installers, operating system package managers or from source (as shown in the following sections). *Express* is then installed by NPM as a dependency of your individual *Express* web applications (along with other libraries like template engines, database drivers, authentication middleware, middleware to serve static files, etc.)

NPM can also be used to (globally) install the *Express Application Generator*, a handy tool for creating skeleton *Express* web apps that follow the MVC pattern. The application generator is optional because you don't *need* to use this tool to create apps that use Express, or construct

Express apps that have the same architectural layout or dependencies. We'll be using it though, because it makes getting started a lot easier, and promotes a modular application structure.

> **Note:** Unlike some other web frameworks, the development environment does not include a separate development web server. In *Node*/*Express* a web application creates and runs its own web server!

There are other peripheral tools that are part of a typical development environment, including text editors or IDEs for editing code, and source control management tools like Git for safely managing different versions of your code. We are assuming that you've already got these sorts of tools installed (in particular a text editor).

What operating systems are supported?

Node can be run on Windows, macOS, many "flavours" of Linux, Docker, etc. (there is a full list on the nodejs Downloads page). Almost any personal computer should have the necessary performance to run Node during development. *Express* is run in a *Node* environment, and hence can run on any platform that runs *Node*.

In this article we provide setup instructions for Windows, macOS, and Ubuntu Linux.

What version of Node/Express should you use?

There are many releases of Node — newer releases contain bug fixes, support for more recent versions of ECMAScript (JavaScript) standards, and improvements to the Node APIs.

Generally you should use the most recent *LTS (long-term supported)* release as this will be more stable than the "current" release while still having relatively recent features (and is still being actively maintained). You should use the *Current* release if you need a feature that is not present in the LTS version.

For *Express* you should always use the latest version.

What about databases and other dependencies?

Other dependencies, such as database drivers, template engines, authentication engines, etc. are part of the application, and are imported into the application environment using the NPM package manager. We'll discuss them in later app-specific articles.

Installing Node

In order to use *Express* you will first have to install *Nodejs* and the Node Package Manager (NPM) on your operating system. The following sections explain the easiest way to install the Long Term Supported (LTS) version of Nodejs on Ubuntu Linux 20.04, macOS, and Windows 10.

> **Tip:** The sections below show the easiest way to install *Node* and *NPM* on our target OS platforms. If you're using another OS or just want to see some of the other approaches for the current platforms then see Installing Node.js via package manager (nodejs.org).

macOS and Windows

Installing *Node* and *NPM* on Windows and macOS is straightforward because you can just use the provided installer:

1. Download the required installer:
    1. Go to https://nodejs.org/en/
    2. Select the button to download the LTS build that is "Recommended for most users".
2. Install Node by double-clicking on the downloaded file and following the installation prompts.

Ubuntu 20.04

The easiest way to install the most recent LTS version of Node 12.x is to use the package manager to get it from the Ubuntu *binary distributions* repository. This can be done very by running the following two commands on your terminal:

```
curl -sL https://deb.nodesource.com/setup_12.x | sudo -E bash -
sudo apt-get install -y nodejs
```

> **Warning:** Don't install directly from the normal Ubuntu repositories because they contain very old versions of node.

Testing your Nodejs and NPM installation

The easiest way to test that node is installed is to run the "version" command in your terminal/command prompt and check that a version string is returned:

```
> node -v
v12.18.4
```

The *Nodejs* package manager *NPM* should also have been installed, and can be tested in the same way:

```
> npm -v
6.14.6
```

As a slightly more exciting test let's create a very basic "pure node" server that prints out "Hello World" in the browser when you visit the correct URL in your browser:

1. Copy the following text into a file named **hellonode.js**. This uses pure *Node* features (nothing from Express) and some ES6 syntax:

   ```
   //Load HTTP module
   const http = require("http");
   const hostname = '127.0.0.1';
   const port = 3000;

   //Create HTTP server and listen on port 3000 for requests
   const server = http.createServer((req, res) => {

     //Set the response HTTP header with HTTP status and Content type
     res.statusCode = 200;
     res.setHeader('Content-Type', 'text/plain');
     res.end('Hello World\n');
   });

   //listen for request on port 3000, and as a callback function have the
   server.listen(port, hostname, () => {
     console.log(`Server running at http://${hostname}:${port}/`);
   });
   ```

   The code imports the "http" module and uses it to create a server (`createServer()`) that listens for HTTP requests on port 3000. The script then prints a message to the console about what browser URL you can use to test the server.
   The `createServer()` function takes as an argument a callback function that will be

invoked when an HTTP request is received — this returns a response with an HTTP status code of 200 ("OK") and the plain text "Hello World".

> **Note:** Don't worry if you don't understand exactly what this code is doing yet! We'll explain our code in greater detail once we start using Express!

2. Start the server by navigating into the same directory as your `hellonode.js` file in your command prompt, and calling `node` along with the script name, like so:

```
>node hellonode.js
Server running at http://127.0.0.1:3000/
```

3. Navigate to the URL http://127.0.0.1:3000 . If everything is working, the browser should display the string "Hello World".

Using NPM

Next to *Node* itself, NPM is the most important tool for working with *Node* applications. NPM is used to fetch any packages (JavaScript libraries) that an application needs for development, testing, and/or production, and may also be used to run tests and tools used in the development process.

> **Note:** From Node's perspective, *Express* is just another package that you need to install using NPM and then require in your own code.

You can manually use NPM to separately fetch each needed package. Typically we instead manage dependencies using a plain-text definition file named package.json . This file lists all the dependencies for a specific JavaScript "package", including the package's name, version, description, initial file to execute, production dependencies, development dependencies, versions of *Node* it can work with, etc. The **package.json** file should contain everything NPM needs to fetch and run your application (if you were writing a reusable library you could use this definition to upload your package to the npm repository and make it available for other users).

Adding dependencies

The following steps show how you can use NPM to download a package, save it into the project dependencies, and then require it in a Node application.

> **Note:** Here we show the instructions to fetch and install the *Express* package. Later on we'll show how this package, and others, are already specified for us using the *Express*
>
> *Application Generator*. This section is provided because it is useful to understand how NPM works and what is being created by the application generator.

1. First create a directory for your new application and navigate into it:
   ```
   mkdir myapp
   cd myapp
   ```
2. Use the npm `init` command to create a **package.json** file for your application. This command prompts you for a number of things, including the name and version of your application and the name of the initial entry point file (by default this is **index.js**). For now, just accept the defaults:
   ```
   npm init
   ```
   If you display the **package.json** file (`cat package.json`), you will see the defaults that you accepted, ending with the license.
   ```
   {
     "name": "myapp",
     "version": "1.0.0",
     "description": "",
     "main": "index.js",
     "scripts": {
       "test": "echo \"Error: no test specified\" && exit 1"
     },
     "author": "",
     "license": "ISC"
   }
   ```
3. Now install Express in the `myapp` directory and save it in the dependencies list of your **package.json** file

4.
   ```
   npm install express
   ```
   The dependencies section of your **package.json** will now appear at the end of the **package.json** file and will include *Express*.
   ```
   {
     "name": "myapp"
   ```

```
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.17.1"
  }
}
```

5. To use the Express library you call the `require()` function in your index.js file to include it in your application. Create this file now, in the root of the "myapp" application directory, and give it the following contents:

```
const express = require('express')
const app = express();
const port = 8000;

app.get('/', (req, res) => {
  res.send('Hello World!')
});

app.listen(port, () => {
  console.log(`Example app listening on port ${port}!`)
});
```

This code shows a minimal "HelloWorld" Express web application. This imports the "express" module using `require()` and uses it to create a server (`app`) that listens for HTTP requests on port 8000 and prints a message to the console explaining what browser URL you can use to test the server. The `app.get()` function only responds to HTTP `GET` requests with the specified URL path ('/'), in this case by calling a function to send our *Hello World!* message.

> **JavaScript Note:** The backticks in the `` `Example app listening on port ${port}!` `` let us interpolate the value of `$port` into the string.

6. You can start the server by calling node with the script in your command prompt:

```
>node index.js
Example app listening on port 8000
```

7. Navigate to the URL (http://127.0.0.1:8000/). If everything is working, the browser should display the string "Hello World!".

Development dependencies

If a dependency is only used during development, you should instead save it as a "development dependency" (so that your package users don't have to install it in production). For example, to use the popular JavaScript Linting tool eslint you would call NPM as shown:

```
npm install eslint --save-dev
```

The following entry would then be added to your application's **package.json**:

```
  "devDependencies": {
    "eslint": "^7.10.0"
  }
```

Note: "Linters " are tools that perform static analysis on software in order to recognize and report adherence/non-adherence to some set of coding best practice.

Running tasks

In addition to defining and fetching dependencies you can also define *named* scripts in your **package.json** files and call NPM to execute them with the run-script command. This approach is commonly used to automate running tests and parts of the development or build toolchain (e.g., running tools to minify JavaScript, shrink images, LINT/analyze your code, etc).

Note: Task runners like Gulp and Grunt can also be used to run tests and other external tools.

For example, to define a script to run the *eslint* development dependency that we specified in the previous section we might add the following script block to our **package.json** file (assuming that our application source is in a folder /src/js):

```
"scripts": {
  ...
  "lint": "eslint src/js"
  ...
}
```

To explain a little further, `eslint src/js` is a command that we could enter in our terminal/command line to run `eslint` on JavaScript files contained in the `src/js` directory

inside our app directory. Including the above inside our app's package.json file provides a shortcut for this command — `lint`.

We would then be able to run *eslint* using NPM by calling:

```
npm run-script lint
# OR (using the alias)
npm run lint
```

This example may not look any shorter than the original command, but you can include much bigger commands inside your npm scripts, including chains of multiple commands. You could identify a single npm script that runs all your tests at once.

Installing the Express Application Generator

The Express Application Generator tool generates an Express application "skeleton". Install the generator using NPM as shown:

```
npm install express-generator -g
```

> **Note:** You may need to prefix this line with `sudo` on Ubuntu or macOS. The `-g` flag installs the tool globally so that you can call it from anywhere.

To create an *Express* app named "helloworld" with the default settings, navigate to where you want to create it and run the app as shown:

```
express helloworld
```

> **Note:** You can also specify the template library to use and a number of other settings. Use the `help` command to see all the options:
>
> ```
> express --help
> ```

NPM will create the new Express app in a sub folder of your current location, displaying build progress on the console. On completion, the tool will display the commands you need to enter to install the Node dependencies and start the app.

> The new app will have a **package.json** file in its root directory. You can open this to see what dependencies are installed, including Express and the template library Jade:
>
> ```
> {
>   "name": "helloworld",
>   "version": "0.0.0",
>   "private": true,
>   "scripts": {
>     "start": "node ./bin/www"
>   },
>   "dependencies": {
>     "cookie-parser": "~1.4.3",
>     "debug": "~2.6.9",
>     "express": "~4.16.0",
>     "http-errors": "~1.6.2",
>     "jade": "~1.11.0",
>     "morgan": "~1.9.0"
>   }
> }
> ```

Install all the dependencies for the helloworld app using NPM as shown:

```
cd helloworld
npm install
```

Then run the app (the commands are slightly different for Windows and Linux/macOS), as shown below:

```
# Run helloworld on Windows with Command Prompt
SET DEBUG=helloworld:* & npm start

# Run helloworld on Windows with PowerShell
SET DEBUG=helloworld:* | npm start

# Run helloworld on Linux/macOS
DEBUG=helloworld:* npm start
```

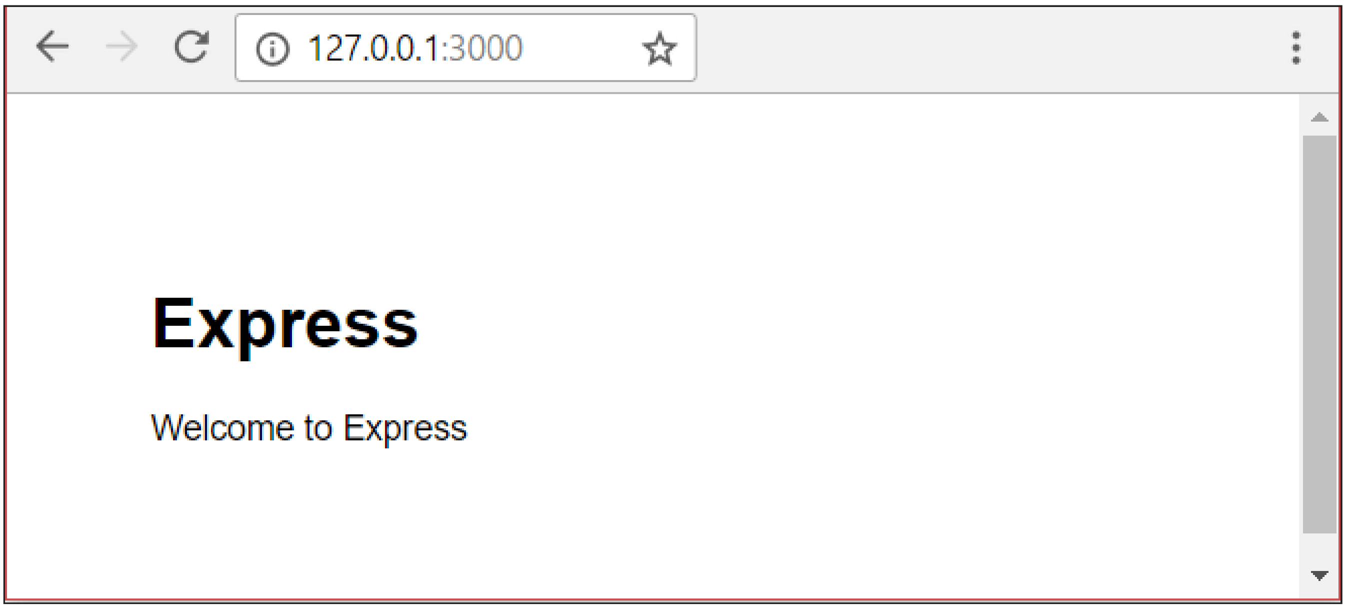
The DEBUG command creates useful logging, resulting in an output like that shown below.

```
>SET DEBUG=helloworld:* & npm start

> helloworld@0.0.0 start D:\Github\expresstests\helloworld
> node ./bin/www

  helloworld:server Listening on port 3000 +0ms
```

Open a browser and navigate to http://127.0.0.1:3000/ to see the default Express welcome page.



Express

Welcome to Express

We'll talk more about the generated app when we get to the article on generating a skeleton application.

Summary

You now have a Node development environment up and running on your computer that can be used for creating Express web applications. You've also seen how NPM can be used to import Express into an application, and also how you can create applications using the Express Application Generator tool and then run them.

In the next article we start working through a tutorial to build a complete web application using this environment and associated tools.

See also

- [Downloads](#) page (nodejs.org)
- [Installing Node.js via package manager](#) (nodejs.org)

- Installing Express (expressjs.com)
- Express Application Generator (expressjs.com)
- Using Node.js with Windows subsystem for Linux (docs.microsoft.com)

In this module

Last modified: Feb 24, 2021, by MDN contributors

Change your language

English (US) Change language