NodeJS Tutorial: Part 1

Posted by Alejandro G. Carlstein Ramos Mejia on April 6, 2017 April 6, 2017 About Programming / NodeJS

Installation

Let's begin by installing NodeJS. Go to the following [link], download the installation package recommended for most users, and while installing, select the default settings provided to you.

After installing, we must ensure that we can run NodeJS. We can do that by opening a command prompt (or terminal) and run the command line:

```
C:\>node --version
v.6.10.2
```

Create a folder where to begin working. In my case, I created the following folder:

```
C:\nodejs\projects\test>
```

> In Windows, you may have to ensure that the environment variable PATH contains the folder where NodeJS was installed.

Building The Foundations

To begin working, we are going to use NPM which is installed with NodeJS. NPM is short for Node Package Manager which is in charge to for the publishing of open-source NodeJS projects; as well as to be our command-line utility which aids us in dependency management, package installation, and version management. This will be our tool to create and maintain our project.

Let's create a package for our project. Run this command line and accept the default values by using the key [Enter] in our keyboard:

```
C:\nodejs\projects\test>npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help json` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg> --save` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
name: (test)
version: (1.0.0)
description:
entry point: (index.js)
test command:
git repository:
keywords:
author:
license: (ISC)
About to write to C:\nodejs\projects\test\package.json:

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


Is this ok? (yes) y
```

Now that we have our package.json file created for us, let's install a dependency called express. This dependency is code available for us to use in our project. Run the command line:

```
C:\nodejs\projects\test>npm install express --save
test@1.0.0 C:\nodejs\projects\test
`-- express@4.15.2
  +-- accepts@1.3.3
  | +-- mime-types@2.1.15
  | | `-- mime-db@1.27.0
  | `-- negotiator@0.6.1
  +-- array-flatten@1.1.1
  +-- content-disposition@0.5.2
  +-- content-type@1.0.2
  +-- cookie@0.3.1
  +-- cookie-signature@1.0.6
  +-- debug@2.6.1
  | `-- ms@0.7.2
  +-- depd@1.1.0
  +-- encodeurl@1.0.1
  +-- escape-html@1.0.3
  +-- etag@1.8.0
  +-- finalhandler@1.0.1
  | +-- debug@2.6.3
  | `-- unpipe@1.0.0
  +-- fresh@0.5.0
  +-- merge-descriptors@1.0.1
  +-- methods@1.1.2
  +-- on-finished@2.3.0
  | `-- ee-first@1.1.1
  +-- parseurl@1.3.1
  +-- path-to-regexp@0.1.7
  +-- proxy-addr@1.1.4
  | +-- forwarded@0.1.0
  | `-- ipaddr.js@1.3.0
  +-- qs@6.4.0
  +-- range-parser@1.2.0
  +-- send@0.15.1
  | +-- destroy@1.0.4
  | +-- http-errors@1.6.1
  | | `-- inherits@2.0.3
  | `-- mime@1.3.4
  +-- serve-static@1.12.1
  +-- setprototypeof@1.0.3
  +-- statuses@1.3.1
  +-- type-is@1.6.15
  | `-- media-typer@0.3.0
  +-- utils-merge@1.0.0
  `-- vary@1.1.1

npm WARN test@1.0.0 No description
npm WARN test@1.0.0 No repository field.
```

This will do two things. Install the package which are going to reside on the folder node_modules and update our package.json file.

Here is the content of the updated package.json file:

```
{
  "name": "test",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.15.2"
  }
}
```

You may notice that under dependencies, the versioning of express starts with a caret **^** symbol. This symbol have a meaning; therefore, let's go over package versioning before moving forwards.

Package Versioning

Versioning consists of one or more sequences of numbers or letters. In this case, it goes like this:

<Major Release>.<Minor Release>.<Patch Release>

Major releases will break backwards compatibility. This means that the new features may not work with the old features.

Minor releases are new features which don't break the existing features.

Patch releases are bug fixes and other minor changes.

The caret **^** symbol means that NPM will install any version of this package that is the same major version. In this case, **^4.15.2** could be seen as **^4.xx.x**. This means that if **4.16** would come up, and we execute npm install (or update), then npm will install/update to the new version **4.16.**

The tilde **~** symbol is another option which tells NPM to install/update based on the minor version. This means that if we have **4.15.2** and a new version is **4.15.3** then this last new version will be install (or update). However, it will not install for example **4.16**.

You can choose to not use the caret (**^**) or tilde (**~**) symbol. Which means you don't wish to take any upgrades. So writing **4.15.2** will tell NPM that you only want that version and no updates are needed.

There are other options and symbols that can be use with NPM; however, I am not going to go over on this tutorial. Therefore, I will just leave these two links where you can go deeper if you wish to: [link], [link]