



## *A technical discussion by TRAVCO*



*What is all the hype about??? I mean, it is JavaScript on the server-side right?? JavaScript, YUCK!  
I'll stick to REAL server-side programming languages!*

## HISTORY



*Express is the most popular JavaScript library that developers use with node to build server-side web applications.*

Node.js was originally designed to be an asynchronous, event-driven framework to build large, scalable network applications. The goals behind its creation included:

- Make it very light-weight
- Make it provide near real-time responses
- Include modules that are designed to reduce the complexity of writing server applications (callbacks are used in place of complex threading code)
- Make it familiar to JavaScript programmers – most of its modules are written in JavaScript

## OKAY, SO WHAT EXACTLY IS NODE.JS??



*Because Node.js does not run within a browser, it does not include browser global variables such as window and document (i.e. window.document)!*

*Chrome's V8 Engine compiles JavaScript source code to native machine code instead of interpreting it in real time.*

In essence, Node.js is a cross-platform JavaScript runtime engine built upon Chrome's V8 engine that you can run from the command-line. It provides an event-driven architecture and a non-blocking I/O API. It runs in a single thread so there is no need to worry about designing code with thread concurrency or dead-locks in mind. Even though it runs in a single thread, it does provide an API to handle multiple CPU cores and share sockets if that is a desire.



You can run Node.js in interpreter/shell mode:

```
[tjjenk2:~]$ node
> for (i = 1; i < 3; i++) {
... console.log(i);
... }
1
2
undefined
> .exit
```

Or as a process to execute JavaScript files: (this includes the ability to launch a network service, such as TCP or HTTP)

```
tjjenk2:~]$ node someJavaScriptFile.js
```

Examples of applications that would benefit from Node.js when used as a web server are: Communication Applications, Web-Based Gaming, and any other application that would require near real-time responses.

## So, is it a JavaScript framework??



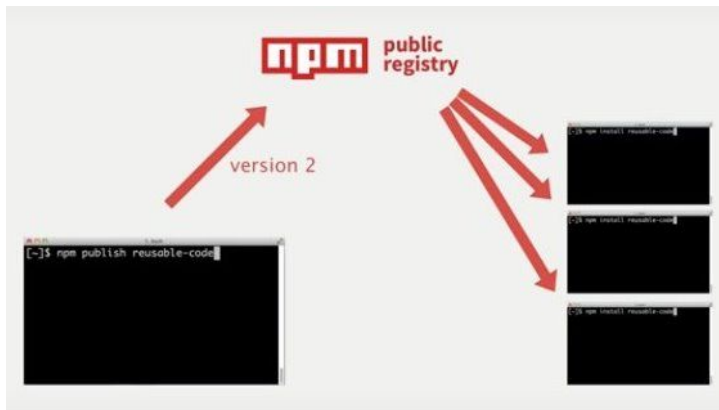
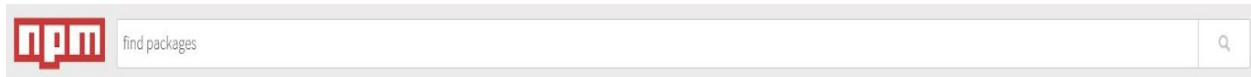
*Node.js was designed by Ryan Dahl in 2009 and the initial release only supported Linux. The inspiration came from a file upload using Flickr!!!*

Node.js is **not** a JavaScript framework, such as Backbone or ExtJS, but much of its supporting code was written in JavaScript and its applications are written in JavaScript. This makes Node.js well suited for the foundation of a web library or framework.



*Npm is the largest ecosystem of open source libraries in the world!*

## NPM??? I THOUGHT WE WERE TALKING ABOUT NODE!



In 2011, Node.js started shipping with a package manager called npm. Npm makes it easy for JavaScript developers to share their code. That is, developers can now build re-usable JavaScript packages/modules that can be used by other developers. Furthermore, npm

makes it easy to check if any **updates** have occurred. The “npm, Inc.” maintains a registry called the “npm Registry” and it is a public collection of packages of open-source JavaScript code. Although most of these packages were designed to run in the node.js engine, some of them were not. You can browse available npm packages via <https://www.npmjs.com/>.

In its simplest form, a package is just a directory with one or more files in it, to include a file called **package.json**. This file represents the meta-data about the package and is similar to Maven’s pom.xml. Similar to pom.xml, it can also contain references to other dependent packages.

There are different kinds of npm packages, to include:

- command-line utilities (e.g. grunt, gulp, bower, yeoman, SASS, Karma)
- server-side node modules – packages that network applications depend upon (e.g. express) and that are loaded with `require()` in a Node.js program.
- client-side node modules – packages that client applications depend on and that are loaded with `require()` in a Node.js program.
- developmental work flow packages/modules (e.g. uglify, jshint, autoprefixer)



- packages that can be used in an application that will run in the browser (e.g. angular or bootstrap) or locally as a client application (e.g. browserify, JSONPath)

Thus, the “npm Registry” behaves similarly to the Maven Central Repository in that it manages versions of JavaScript packages that were designed around node.js and/or npm.

```
[tjjenk2:~]$ sudo npm install npm -g
```

Once you install Node.js on your local file system and add its `bin` directory to the `$PATH`, the `npm` client will be available from the command line. However, the `npm` client gets updated more frequently than Node so you may want to update it periodically by doing:

(-g implies global install)

An internet connection is required for npm to interact with the remote repository. However, it is possible to setup a private npm repository. I believe this can be done by replicating the couch database.

## OKAY, SO HOW DO I USE NPM TO MANAGE MY PACKAGES?

*[!] Largely due to npm, there has been an explosion in the community of active Node.js users.*

There are two ways to install npm packages:

- Locally
  - o If you depend on a package from your own module using Node.js `require()`
  - o If you want to use it in a web application
  - o If you want to use it in your developmental work flow
  - o All packages will be installed in the current working directory. Node modules will go in `./node_modules`.
  - o You can install a local package in three ways:



1. package dependency – this will modify the package.json file and add the dependency and associated version to dependencies.
2. package development-dependency – this will modify the package.json file and add the dependency and associated version to devDependencies. These types of packages are meant to be used in the build process and not as runtime dependencies.
3. local node module or package – the package.json file will be left unchanged.

- Globally

- o If you want to use it as a command-line utility or on your shell
- o npm uses an internal variable called prefix to track the global location
- o You need to ensure you add the global prefix directory to your \$PATH so that all command-line utilities are visible to the shell

```
> # view the npm global directory
[tjjenk2:~]$ npm config get prefix
/usr/local

# set the npm global directory
[tjjenk2:~]$ npm config set prefix ~/apps/nodenpm

# install a global command-line utility
[tjjenk2:~]$ npm install -g grunt-cli

# creates a generic package.json file in current directory
[tjjenk2:project]$ npm init

# install a local node module (package.json left alone)
[tjjenk2:project]$ npm install requirejs

# install a dependent package (modifies package.json)
[tjjenk2:project]$ npm install requirejs --save

# install a dev-dependent package (modifies package.json)
[tjjenk2:project]$ npm install grunt-contrib-uglify
--save-dev

# to see what packages are outdated
[tjjenk2:project]$ npm outdated
```



```
# to update packages to latest versions (this does all)
[tjjenk2:project]$ npm update
```

If you use the package.json file to maintain the list of dependencies, you alleviate the need to share dependency packages with coworkers. Coworkers will only need the package.json file and can get all the dependencies by simply doing a `npm install` in the project directory that contains the package.json file.

## WAIT THERE'S MORE?



Well, hello there “Developmental Workflow!” Now that you understand npm, you can use it to download utilities to simplify the complexity involved in maintaining web application builds, such as the ability to use SASS/Compass for CSS, autoprefixer CSS, Typescript compile, concat, minify, unit test, etc.




Bower is an alternative package manager to npm. If you do not intend on building applications that will run in the Node.js and instead want to use a package manager for web application packages, then you can use Bower in place of npm. One difference is that bower requires **GIT**.

Bower is optimized for the front-end because if more than one package depends upon another package, it will only download a single copy. Unlike npm, which downloads its packages to the `./node_modules` directory, to include all the metadata files and




dependencies files, bower simply downloads the necessary web application files and stores them in a single directory called `./bower_components`.

You can override the default directory by creating a file called `.bowerrc` that has contents similar to the following:

```
 { "directory": "thirdparty/bin" }
```

`.bowerrc`


Then in the HTML file, you can reference a script by doing something similar to the following:

```
 <script  
  src="thirdparty/bin/jquery/jquery.min.js"></script>
```

`index.html`

Bower behaves nearly the same as npm. Similar to npm's `package.json` metadata file, it has a **`bower.json`** metadata file. And instead of using `npm install`, you use `bower install`. Bower also has its own repository that can be searched via <http://bower.io/search/>.

Below are some sample bower commands:

```
 # install bower as a command line tool (only do this once)  
[tjjenk2:~]$ npm install -g bower  
  
# to create a default bower.json file  
[tjjenk2:~]$ bower init  
  
# get help on bower command-line utility  
[tjjenk2:~]$ bower -help  
  
# search bower registry  
[tjjenk2:project]$ bower search <package>  
  
# list installed packages  
[tjjenk2:project]$ bower list  
  
# check for updates  
[tjjenk2:project]$ bower check-new  
  
# update  
[tjjenk2:project]$ bower update
```



```
# download bootstrap for use in your web application
[tjjenk2:project]$ bower install bootstrap

# download angular, but modify the bower.json dependencies
[tjjenk2:project]$ bower install angular -save

# use bower.json to download all dependencies
[tjjenk2:project]$ bower install
```

Those are just a few of the commands, but you should be able to notice that Bower behaves similarly to npm.

## DEVELOPMENTAL WORKFLOW – WHAT’S A TASK RUNNER?

*[!] The task runners mentioned below are currently the most popular, but others may arrive in the future.*

Similar to Apache ANT, there are three npm-based “task runners” or build systems that can help manage the workflow of your application. As web developers, we often need to perform multiple tasks before we can deploy our web code. An example workflow may include the following:




1. We may need to download and install multiple Javascript APIs, such as Angular, Bootstrap, Underscore, q (promises), JQuery, etc. and we often end up managing them on our own. We often check them into our version/control system.
2. We may use SASS/Compass to build our applications CSS and we may want to also use an autoprefixer to support CSS3 working in multiple browsers. Furthermore, we may end up having multiple files and we’ll want to concatenate them and minify the result.
3. We may want to compress our images.
4. We may want to write our JavaScript code in multiple files and use something like Typescript to do it. You also may want to use JSHint as well. Like CSS, we may want to concatenate all those files together and minify the result.
5. You may want to include JavaScript unit tests.





The above is a common scenario and trying to do this in Maven could be cumbersome. This is where Node.js task runners can help “Save the Day!” However, there are many to choose from and each has its own syntaxes, quirks, and gotchas that you need to learn.

Which one you choose may depend upon the following chart:

Task Runner Type	Description
<b>Grunt</b> 	<ul style="list-style-type: none"><li>• It was one of the first and has a healthy community around it, even on windows (this one probably has the widest community support)</li><li>• It is easy to use because tasks are configured declaratively; just pick plugins and then declare them</li><li>• It can get too verbose if you have a large build flow and even simple-tasks can involve over-configuration</li><li>• Since tasks are configured declaratively, you can have a hard time figuring out the order in which tasks get executed</li><li>• Because it needs to complete a task before going onto the next, performance can be slow when transforming files for example</li><li>• It usually requires additional grunt-specific CLI commands and contrib packages</li></ul>
<b>Gulp</b> 	<ul style="list-style-type: none"><li>• It is similar to grunt in that it relies on plugins and it's cross-platform</li><li>• It is a code-driven build tool that uses Node.js streams (via piping), in contrast with Grunt's declarative approach of task definitions</li><li>• Because of its streaming approach, it can be much faster than Grunt when performing tasks that transforms files through many stages</li><li>• Because it is code-driven (i.e. pipe tasks), it requires much less configuration than Grunt</li><li>• It does not yet have as large of a community following as Grunt</li><li>• It usually requires additional gulp-specific CLI commands and contrib packages</li></ul>
<b>npm-scripting</b> 	<ul style="list-style-type: none"><li>• It is built into npm and can be declared in the package.json file in the scripts section</li><li>• If a plugin does not exist for a specific tool (e.g. Grunt or Gulp), you're out of luck (unless you write it). However, since this is native npm, you can take advantage of all of npm's packages. You won't need any additional CLI tooling or files</li><li>• It supports bash shell scripting and you can even use bash pipes</li><li>• You can schedule tasks as background jobs</li><li>• It does not work well with Windows</li></ul>
<b>Others</b>	Broccoli and Bunch (but I did not read about them)



## OKAY, SETTING UP DEV-WORK FLOW SEEMS CUMBERSOME!

 *The community has usually thought of everything you may need to do. So search the repos first!*

Managing build files and determining which tasks you may need seems cumbersome. Often, you will end up copying old project metadata files because they will include everything you probably need. The community has already addressed this need by providing various ways (i.e. scaffolding tools) to automate this task.

Scaffolding tools are used to automate project creation! A scaffolding tool, such as grunt-init or Yeoman, can be used to quickly initialize a project with all your needed files. If you are working with grunt, you can install grunt-init as a command-line tool and then download the various templates (or create your own) from the repositories. Then, you'd use grunt-init in a project directory to quickly create your Grunt and NPM metadata files as well as preparing it for third-party packages, such as angular or bootstrap.



Likewise, you can use the Yeoman scaffolding workflow to quickly create all the initial files you may need for your new project. Yeoman integrates well with Grunt and Bower.