

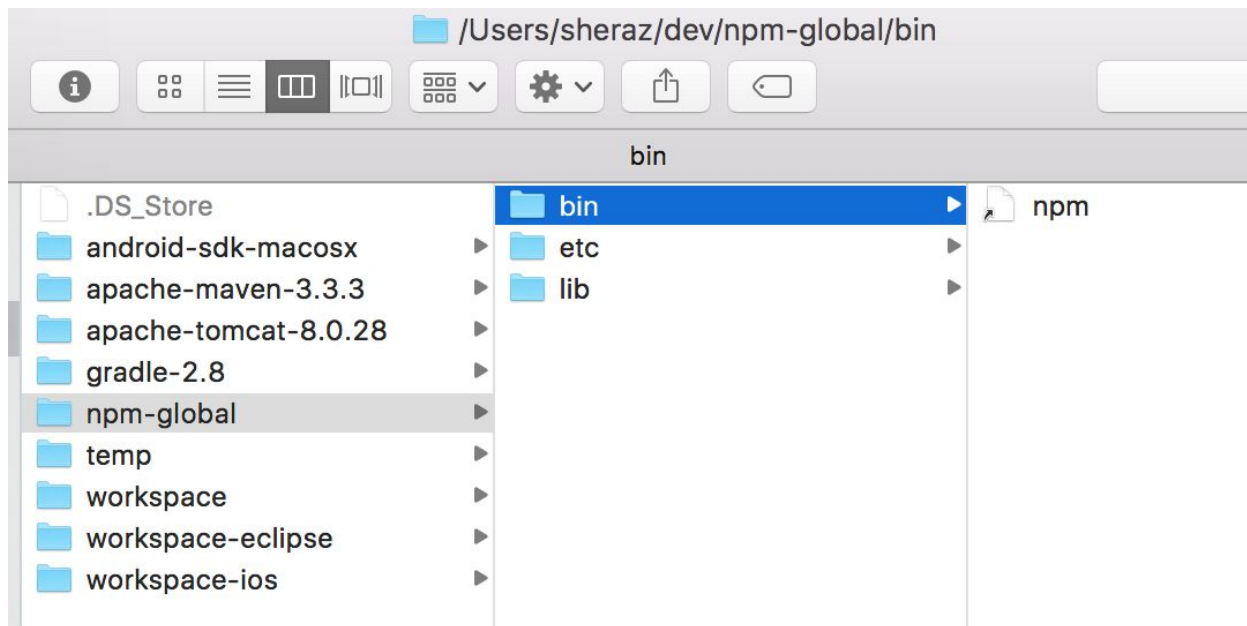
NPM

Change Default Global directory Or Change npm prefix

Default npm global directory is "/usr/local"

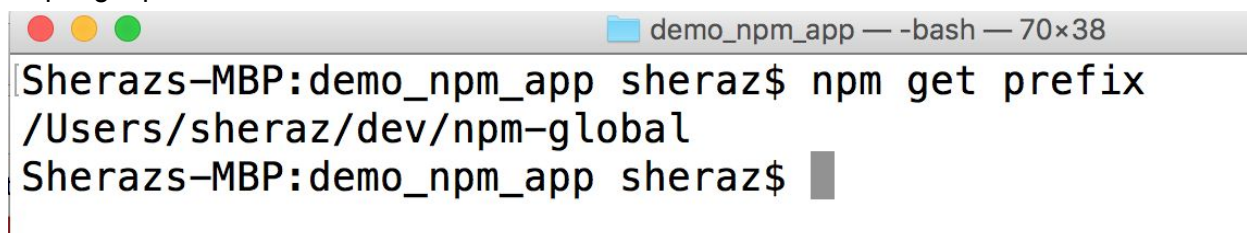
To change it give this command:

```
$ npm set prefix /Users/sheraz/dev/npm-global
```



View configured prefix

```
$ npm get prefix
```



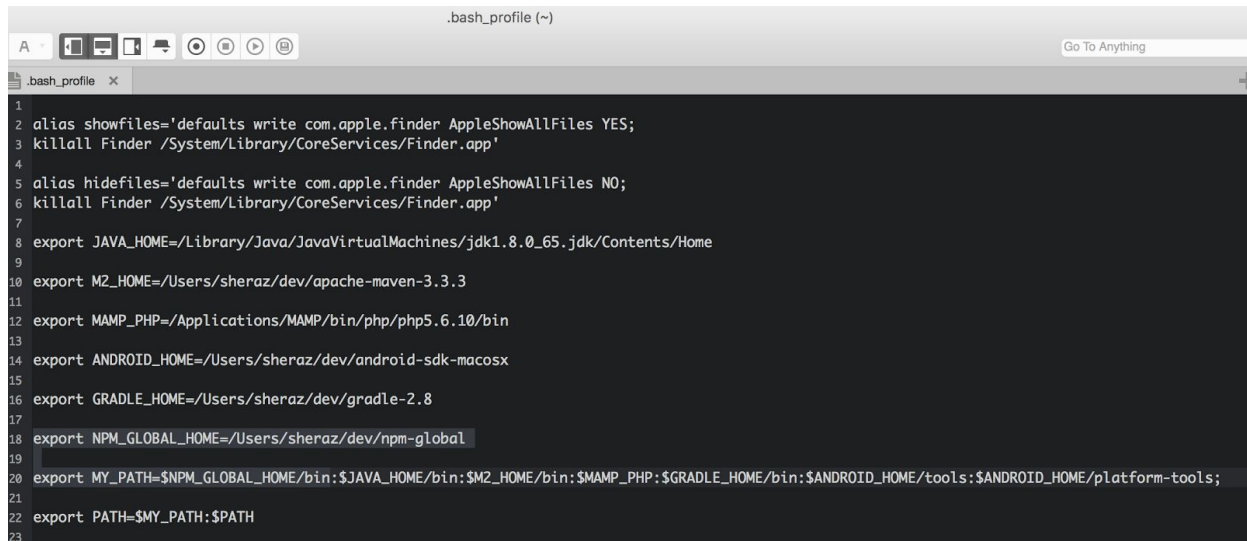
~/.npmrc

<https://docs.npmjs.com/files/npmrc>

.npmrc file in user home store all default user settings. prefix is also stored in it.

Add new global home to path

Now add new created global directory's bin in the path by modifying ~/.profile or ~/.bash_profile



```
.bash_profile (~)
1 alias showfiles='defaults write com.apple.finder AppleShowAllFiles YES;
2 killall Finder /System/Library/CoreServices/Finder.app'
3
4 alias hidefiles='defaults write com.apple.finder AppleShowAllFiles NO;
5 killall Finder /System/Library/CoreServices/Finder.app'
6
7 export JAVA_HOME=/Library/Java/JavaVirtualMachines/jdk1.8.0_65.jdk/Contents/Home
8
9 export M2_HOME=/Users/sheraz/dev/apache-maven-3.3.3
10
11 export MAMP_PHP=/Applications/MAMP/bin/php/php5.6.10/bin
12
13 export ANDROID_HOME=/Users/sheraz/dev/android-sdk-macosx
14
15 export GRADLE_HOME=/Users/sheraz/dev/gradle-2.8
16
17 export NPM_GLOBAL_HOME=/Users/sheraz/dev/npm-global
18
19 export MY_PATH=$NPM_GLOBAL_HOME/bin:$JAVA_HOME/bin:$M2_HOME/bin:$MAMP_PHP:$GRADLE_HOME/bin:$ANDROID_HOME/tools:$ANDROID_HOME/platform-tools;
20
21 export PATH=$MY_PATH:$PATH
22
23
```

Now restart computer or give command

~ \$ source ~/.bash_profile

Update npm

This will install updated version of npm globally

\$ npm install npm -g

Create New Project/node module

Configuration and create package.json

\$ npm init

View all remote package versions available

\$ npm view lodash versions

view command prints package.json values.

Install packages/modules locally in project

Install/download dependency in node_modules folder

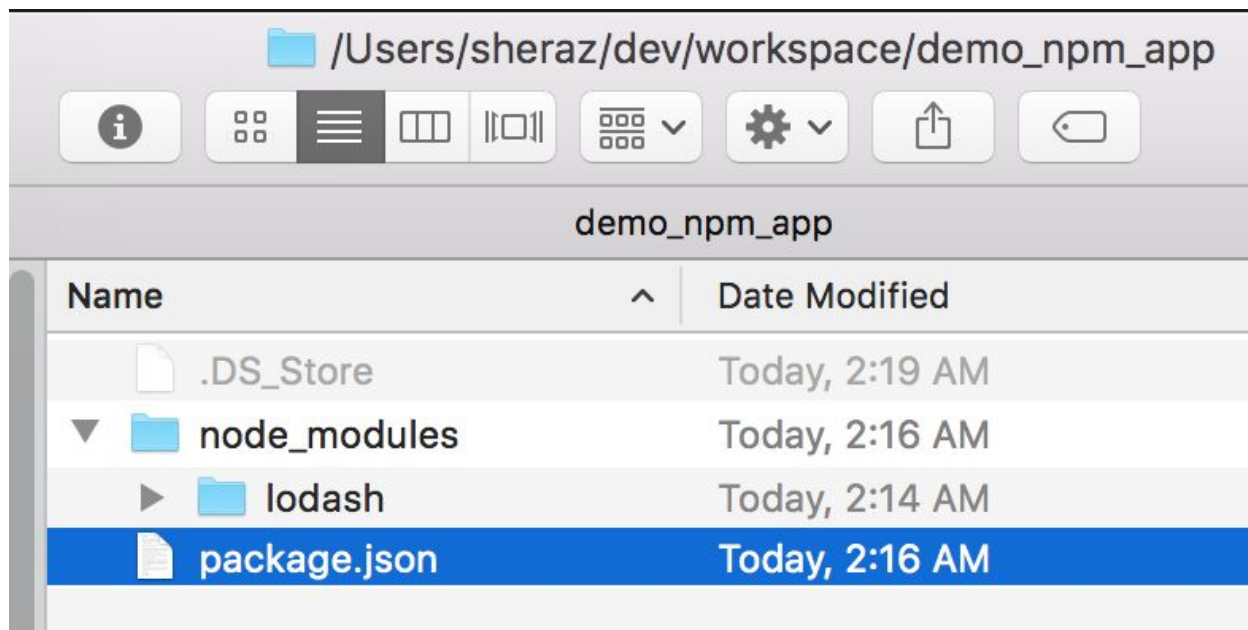
```
$ npm install lodash
```

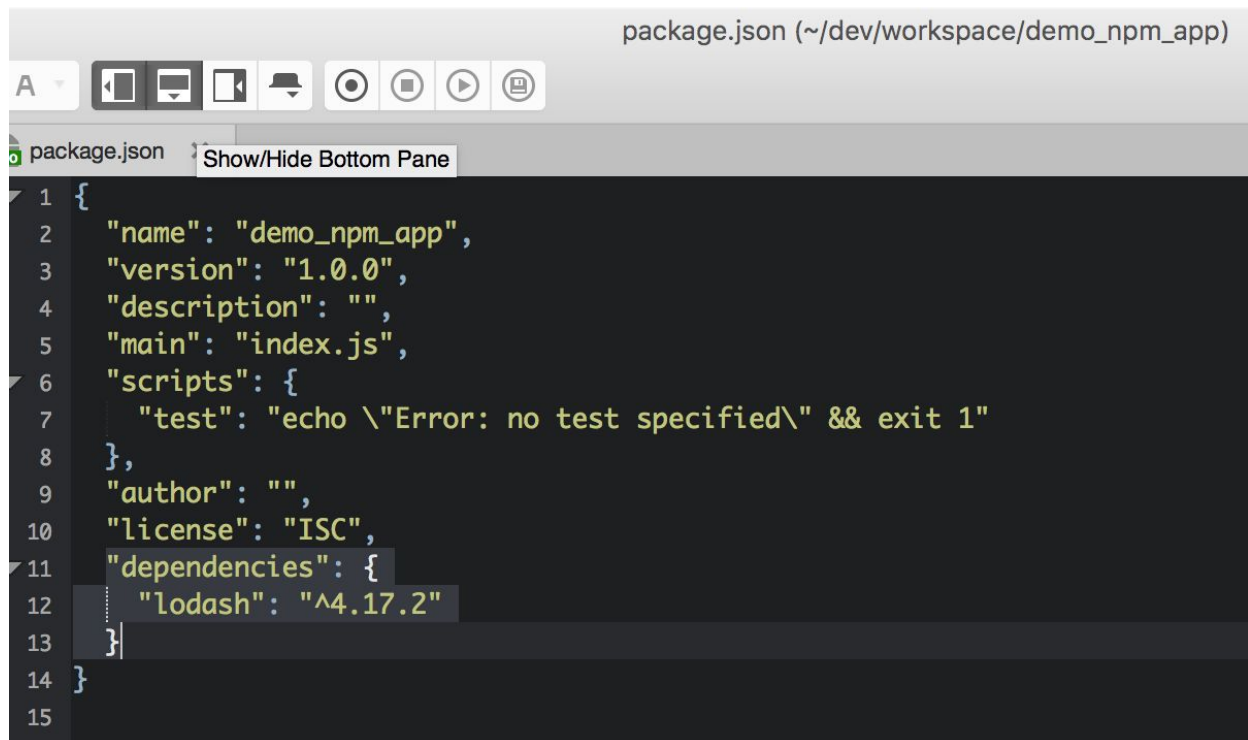
Update package.json and install/download dependency in node_modules folder

```
$ npm install lodash --save
```

Install a particular version of dependency. Version do not have to be a full minor versions.

```
npm install lodash@4.16 --save
```





```
package.json (~/dev/workspace/demo_npm_app)

1 {
2   "name": "demo_npm_app",
3   "version": "1.0.0",
4   "description": "",
5   "main": "index.js",
6   "scripts": {
7     "test": "echo \"Error: no test specified\" && exit 1"
8   },
9   "author": "",
10  "license": "ISC",
11  "dependencies": {
12    "lodash": "^4.17.2"
13  }
14 }
15
```

e.g.

```
"dependencies": {
  "lodash": "^4.17.2",
  "tap": "*"
}
```

Installing latest non Production, alpha or beta version

\$ npm install --save react-router@next
Instead of using version we use "@next"

Semantic versioning

Npm version number are of 3 parts

<Major releases>.<Minor releases>.<Patch releases>

Major releases: Major overhaul of package.

Minor releases: Feature changes.

Patch releases: Bug fixes.

To get latest **Patch releases** specify version like: 1.0 or 1.0.x or ~1.0.4

To get latest **Minor releases** specify version like: 1 or 1.x or ^1.0.4

To get latest **Major releases** specify version like: * or x

Upgrade current app/module version

Upgrade commands upgrade version number in package.json

Check current app/module version

In node project directory run:

```
$ npm version
```

Upgrade patch version

```
$ npm version patch
```

Upgrade minor version

```
$ npm version minor
```

Upgrade major version

```
$ npm version major
```

Listing Packages

Listing local packages

```
$ npm ls --depth=0
```

List global packages

```
$ npm ls -g --depth=0
```

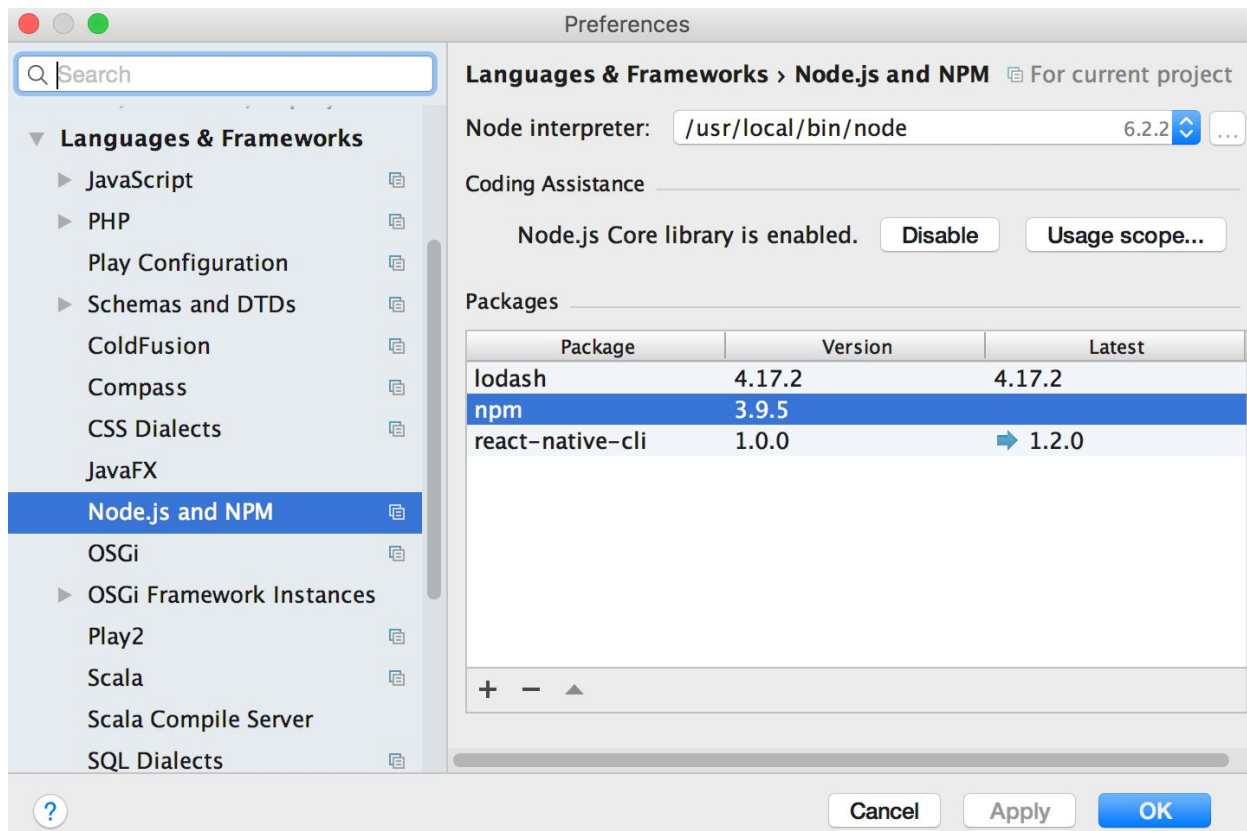
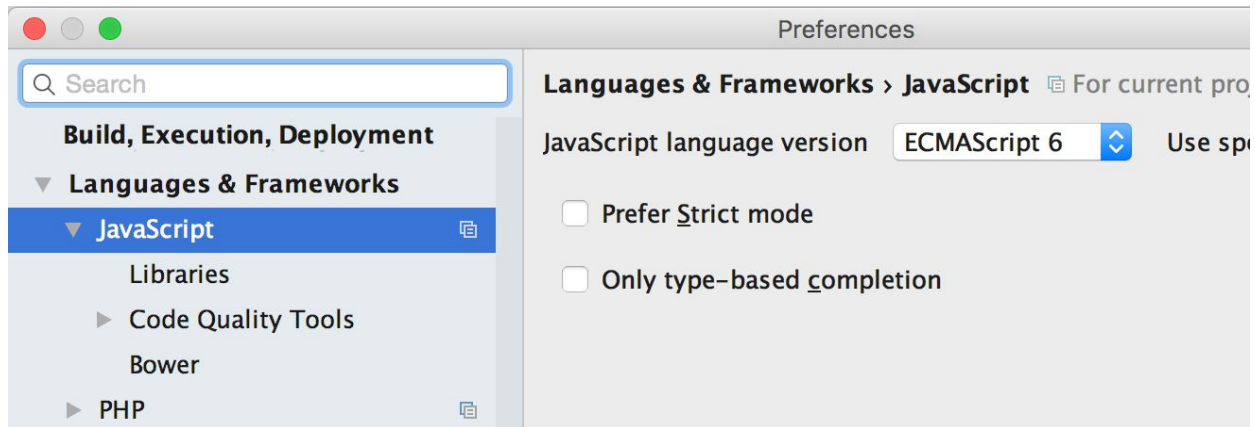
--depth=0 only prints first level hierarchy.

Reinstalling dependencies

If any dependency "node_modules" directories is missing. Then run this command to re-install them

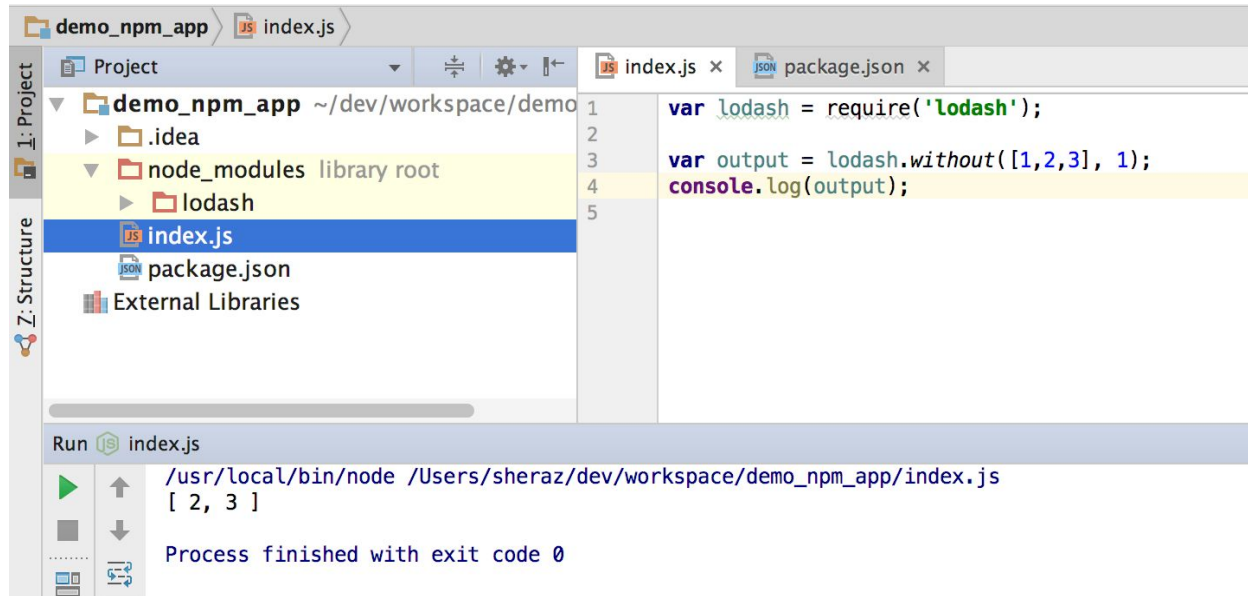
```
$ npm install
```

IntelliJ - add support for ES6 and Node code assistance



Example of using dependency

Run in IntelliJ



Or Run in console

```
[Sherazs-MBP:demo_npm_app sheraz$ ls
index.js      node_modules  package.json
[Sherazs-MBP:demo_npm_app sheraz$ node index.js
[ 2, 3 ]
```


View/Check latest and wanted version and updating installed dependencies

```
demo_npm_app — -bash — 70x38
Sherazs-MBP:demo_npm_app sheraz$ npm ls
demo_npm_app@1.0.0 /Users/sheraz/dev/workspace/demo_npm_app
└─ UNMET DEPENDENCY lodash@2.4.0

npm ERR! missing: lodash@2.4.0, required by demo_npm_app@1.0.0
Sherazs-MBP:demo_npm_app sheraz$ npm install
demo_npm_app@1.0.0 /Users/sheraz/dev/workspace/demo_npm_app
└─ lodash@2.4.0

npm WARN demo_npm_app@1.0.0 No description
npm WARN demo_npm_app@1.0.0 No repository field.
Sherazs-MBP:demo_npm_app sheraz$ npm ls
demo_npm_app@1.0.0 /Users/sheraz/dev/workspace/demo_npm_app
└─ lodash@2.4.0

Sherazs-MBP:demo_npm_app sheraz$ npm outdated
Package   Current  Wanted  Latest  Location
lodash    2.4.0    2.4.0   4.17.2  demo_npm_app
Sherazs-MBP:demo_npm_app sheraz$ npm update
```

Uninstall Local Packages

This will delete package from "node_modules"

```
$ npm uninstall lodash
```

This will update package.json and delete package from "node_modules"

```
$ npm uninstall lodash --save
```

If dependency is removed from package.json but is in node_modules. Then use prune command to cleanup


```

demo_npm_app — -bash — 70x38
Sherazs-MBP:demo_npm_app sheraz$ npm ls
demo_npm_app@1.0.0 /Users/sheraz/dev/workspace/demo_npm_app
├─ lodash@4.16.6 extraneous

npm ERR! extraneous: lodash@4.16.6 /Users/sheraz/dev/workspace/demo_npm_app/node_modules/lodash
Sherazs-MBP:demo_npm_app sheraz$ npm prune
npm WARN package.json demo_npm_app@1.0.0 No description
npm WARN package.json demo_npm_app@1.0.0 No repository field.
npm WARN package.json demo_npm_app@1.0.0 No README data
unbuild lodash@4.16.6
Sherazs-MBP:demo_npm_app sheraz$

```

Install packages globally

Usually we install packages globally that are cli (command line interface). E.g. jshint, grunt

Install

\$ npm install jshint -g

```

demo_npm_app — -bash — 80x39
[
Sherazs-MBP:demo_npm_app sheraz$ npm install jshint -g
[ /Users/sheraz/dev/npm-global/bin/jshint -> /Users/sheraz/dev/npm-global/lib/node_modules/jshint/bin/jshint
[ /Users/sheraz/dev/npm-global/lib
[ └─ jshint@2.9.4
]

```

Uninstall

\$ npm uninstall jshint -g

Update

\$ npm update jshint -g

Using jshint

Jshint CLI shows error in a js file

\$ jshint file_contain_error.js

Add NPM user to publish module on npmjs.com

<https://docs.npmjs.com/cli/adduser>

```
demo_npm_app — -bash — 80x4
[ Sherazs-MBP:demo_npm_app sheraz$ npm adduser
[ Username: sheraz
[ Password:
[ Email: (this IS public) stariqch@gmail.com
[ Logged in as sheraz on https://registry.npmjs.org/.
[ Sherazs-MBP:demo_npm_app sheraz$ █
```

NPM user
stariqch@gmail.com
sheraz/password123

After successfully adding user. You can lookup user created at:
<https://www.npmjs.com/~sheraz>
Click login to edit personal data and view all published packages

View npm configuration

\$ npm config ls
Or
\$ npm get

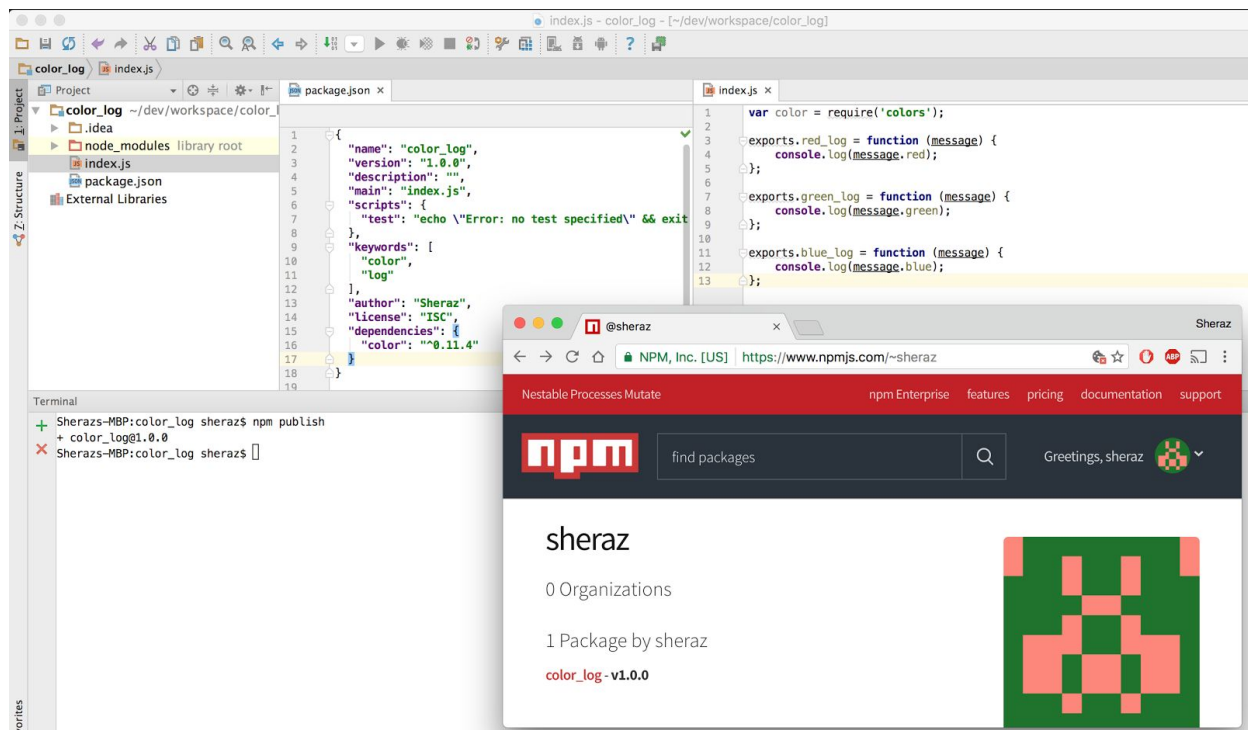
Publish package

Creating a publishing package

Create a node project. Add functionalities to it and run the command below to show up on npm user registry.

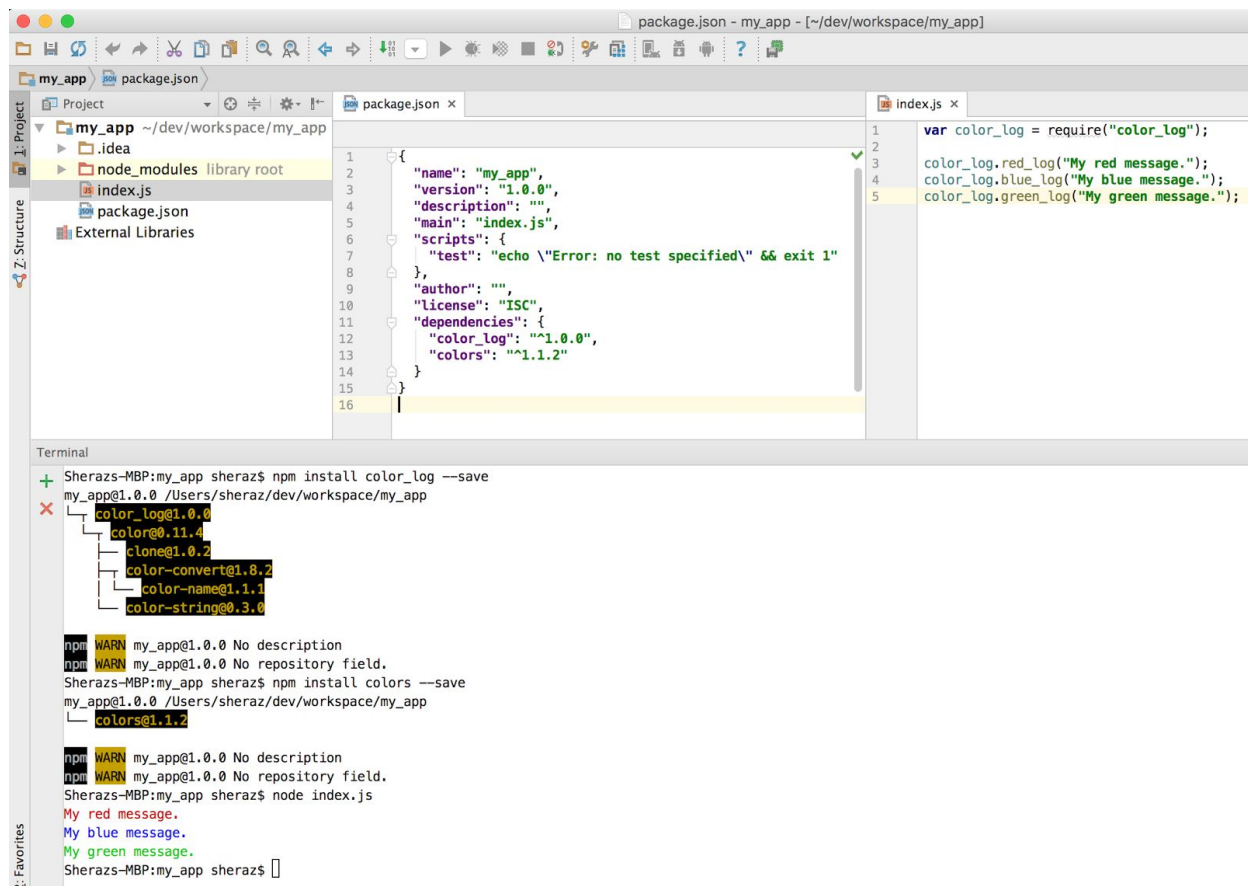
<https://www.npmjs.com/~sheraz>

\$ npm publish



Using published package

- Create new node project
- Install dependencies.
- require() dependency
- Run application.



Scoped Packages - Namespaces

<https://docs.npmjs.com/getting-started/scoped-packages>

To avoid name conflicts we can create Scoped Packages or in other words give them namespaces.

To create scoped Package

```
$ npm init --scope=my_org_name
```

To install scoped Package

```
$ npm install @angular/core
```

To use scoped Package

```
import { Component } from '@angular/core';
```