NPM

Introduction

Node Package Manager (NPM) is a package manager for Javascript packages. This includes both packages for the web clients (e.g. jquery) and packages for programs that run on nodejs (e.g. redis-commander).

A npm package is a directory with one or more modules inside of it and a package.json file that contains metadata describing the package. The metadata includes things like who the author of the package is, what the version of the package is, what other packages does the package depend on, etc..

> **NOTE**: Remember that there's a difference between a package and a module. A module is a single Javascript file. A package can have many modules (javascript files).

> **NOTE**: Unlike maven, packages in npm's registry isn't limited to development packages -- they contain packages for users as well. For example, there's a package called

Setup

To set up NPM and nodejs, you can use the instructions at https://nodejs.org/en/download/package-manager.

For Linux Mint, this was…

```
curl -sL https://deb.nodesource.com/setup_8.x | sudo -E bash -
sudo apt-get install -y nodejs
```

Package Creation

To create a package, all you really need is a package.json file. If you don't want to write it out by hand, you can use the npm init. This command will ask you questions via the command-line and generate a package.json based on your answers.

> **NOTE**: If you're creating a lot of packages, you can use npm set to set defaults for a lot of the fields that npm init asks for. For example, you can use npm set init-author-name 'Firstname Lastname' to set the default author.

```
~/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>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (test)
version: (1.0.0)
description:
entry point: (index.js)
test command:
git repository:
keywords:
```

```
author:
license: (ISC)
About to write to /home/user/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)
```

Package Dependencies

Packages can have code dependencies and dev dependencies. Code dependencies are used by your code, while dev dependencies are used by your build process.

Add Dependencies

Code Dependencies

To add a dependency in your package, you you can run npm install packagename --save. This will download the dependency (along with any dependencies it depends on) and add the dependencies field of your package.json file.

By default, the latest version of the package is used. Downloaded dependencies should end up in a subdirectory called node_modules.

> **NOTE**: -S can be used as shorthand for --save.

```
~/test $ npm install lodash --save
+ lodash@4.17.4
added 1 package in 0.578s
~/test $ cat package.json
```

```
{
  "name": "test",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "lodash": "^4.17.4"
  }
}
```

Dev Dependencies

In addition to code dependencies, npm allows you to have development dependencies. Development dependencies are dependencies that your build process may need but your code doesn't. For example, you may have tests that depend on the Javascript testing framework Mocha -- the code for your application won't need Mocha, but the tests for your application will.

To add a development dependency, you can run npm install packagename --save-dev. This will download the dependency (along with any dependencies it depends on) and add the devDependencies field of your package.json file.

By default, the latest version of the package is used. Downloaded dependencies should end up in a subdirectory called node_modules.

> **NOTE**: -D can be used as shorthand for --save-dev.

```
user@user-VirtualBox ~/test $ npm install mocha --save-dev
+ mocha@4.0.1
added 24 packages in 1.057s
user@user-VirtualBox ~/test $ cat package.json
{
  "name": "test",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
```

```
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "lodash": "^4.17.4"
  },
  "devDependencies": {
    "mocha": "^4.0.1"
  }
}
```

Remove Dependencies

To remove dependencies, you can use npm uninstall packagename. It doesn't matter if it's a development dependency or a code dependency -- it gets removed from the appropriate section of the package.json file as well as deleted from the node_modules subdirectory.

```
~/test $ npm uninstall mocha
removed 24 packages in 0.174s
```

> **NOTE**: It looks like in older versions of npm, the uninstall command just deleted the package from the node_modules subdirectory. If you wanted it out of package.json you needed to provide a --save flag (for code dependencies) or a --save-dev flag (for development dependencies). It doesn't look like this is the case now (using npm 5.5.1).

Sometimes you may remove dependencies by directly removing them from your package.json file. If you do this, you should also run npm prune afterwards to remove the dependencies from your node_modules subdirectory.

List Dependencies

To list the dependency tree of your package, you can list it using npm list.

> **NOTE**: To limit the depth, you can use the --depth argument.

```
~/test $ npm list
test@1.0.0 /home/user/test
├─── lodash@4.17.4
└─┬ mocha@4.0.1
  ├─── browser-stdout@1.3.0
  ├─── commander@2.11.0
```

```
├─┬─ debug@3.1.0
│ └──── ms@2.0.0
├──── diff@3.3.1
├──── escape-string-regexp@1.0.5
├─┬─ glob@7.1.2
│ ├──── fs.realpath@1.0.0
│ ├─┬─ inflight@1.0.6
│ │ ├──── once@1.4.0 deduped
│ │ └──── wrappy@1.0.2
│ ├──── inherits@2.0.3
│ ├─┬─ minimatch@3.0.4
│ │ └─┬─ brace-expansion@1.1.8
│ │   ├──── balanced-match@1.0.0
│ │   └──── concat-map@0.0.1
│ ├─┬─ once@1.4.0
│ │ └──── wrappy@1.0.2 deduped
│ └──── path-is-absolute@1.0.1
├──── growl@1.10.3
├──── he@1.1.1
├─┬─ mkdirp@0.5.1
│ └──── minimist@0.0.8
└─┬─ supports-color@4.4.0
  └──── has-flag@2.0.0
```

Manage Dependencies

Dependency Versions

Packages in npm use semantic versioning (https://semver.org). When you use npm install but don't specify a version number, npm will…

- download the latest version that package that's currently in the npm registry
- setup your package.json to use the latest MINOR version of what was downloaded

For example, if you were to npm install lodash --save, npm will look through the registry and see that version 4.17.4 is the latest version of lodash. It'll download that version into your node_modules subdirectory, and setup your package.json to use the newest version of lodash 4.x.x.

> **NOTE**: The carrot (^) before the version means always use the latest minor version, so the latest version so long as it's version 4.x.x.

```
"dependencies": {
  "lodash": "^4.17.4"
}
```

Exact Versions

To download a specific version of the dependency rather than the latest, add the version after the package name when you run npm install, separated by a @ sign. In addition to that, if you want your package.json to target that exact version as well, add the --save-exact flag to npm install.

For example, npm install lodash@4.1.0 --save --save-exact will download version 4.1.0 and setup your package.json to target that exact version of lodash.

> **NOTE**: Notice how there's no carrot (^) or tilde (~) before the version number here. Noy having those tells npm to use this exact version.

```
"dependencies": {
  "lodash": "4.1.0"
}
```

Version Ranges

By default, when you add a dependency but don't use the --save-exact flag, the version that goes into your package.json has a carrot (^) prepended to it.

```
"dependencies": {
  "lodash": "^4.17.4"
}
```

The carrot (^) and tilde(^) have a special meaning when they're prepended onto a version number. When npm sees the …

- ~, it uses the most recent minor version → e.g. ~3.5.5 will use the latest 3.5.x version
- ^, it uses the most recent major version → e.g. ^3.5.5 will use the latest 3.x.x version

Manage Dependency Locations

Sometimes, packages may not be in the npm registry. In those cases, you can either download the project to a local directory or point to URL when you do npm install: npm install <location> --save.

For example, npm install https://github.com/palantir/eclipse-typescript --save results in…

```
"dependencies": {
  "eclipse-typescript":
"git+https://github.com/palantir/eclipse-typescript.git"
}
```

Updating Dependencies

Depending on how your dependency versions are specified in your package.json (read the section on versions for more info on this), you can update dependencies using npm update.

Prior to an npm update…

```
  "dependencies": {
    "lodash": "^4.1.0"
  }
```

After the npm update...

```
  "dependencies": {
    "lodash": "^4.17.4"
  }
```

Package Scripts

The package.json file may define one or more scripts that you can run to perform some task. These scripts are basically just CLI commands that get piped to an OS shell -- they aren't unique to npm or Javascript.

NPM has several built-in script names that you can run from just by calling npm scriptname: https://docs.npmjs.com/misc/scripts. For example, you can run the "test" script by simply running npm test.

```
  "scripts": {
    "test": "node runscripts"
    "start": "node startwebserverapp"
  }
```

If your script name isn't part of the list of built-in script names, you can still run it but you have to use npm run scriptname. For example, npm run mycustomscriptnamehere.

The most common scripts included in package.json are…

- test → runs tests for your project

- start → starts your project (e.g. start a webserver and point it to your files)

Cloned Packages

Sometimes, packages may not be in the npm registry. In those cases, you can either download the project to a local directory and perform npm install, or you can point to the URL directly and do npm install <url>.

```
~/node-github $ npm install
npm notice created a lockfile as package-lock.json. You should commit this
file.
added 927 packages in 19.567s
```

After installing, you should notice a new subdirectory called node_modules. This is where all the dependencies, along with whatever other dependencies those dependencies may have down the line, have been downloaded to.

End-user Packages

To install a end-user package, you typically need to include the -g flag when you install: npm install -g packagename. The -g flag means that the package will be installed globally.

For example, there's a package called redis-commander that provides users with a nice web-based GUI to interface with a Redis server. You install it by using npm install -g redis-commander.

The installation process will set up your system with all the scripts/aliases needed to run the package. In redis-commander's case, your system will be updated such that if you type redis-commander in a shell, it'll start up the redis-commander UI.

> **NOTE**: Since system-level changes may be happening (e.g. setting up new global aliases), you may need to be running the installation as root.
>
> **NOTE**: To list all packages installed globally, you can use npm list -g. To uninstall a global package, you can use npm uninstall -g packagename.
>
> **NOTE**: If you don't use the -g flag, the scripts/aliases/binaries will go in the .bin subdirectory. If you're running npm scripts (see npm scripts section), this .bin subdirectory is automatically added to the paths environment variable prior to the script running, so you'll automatically have access to the scripts/aliases/binaries from your scripts.

```
~ $ sudo npm install -g redis-commander
+ redis-commander@0.4.5
added 475 packages in 8.966s
~ $ redis-commander
{ Error: ENOENT: no such file or directory, open
'/home/user/.redis-commander'
  errno: -2,
  code: 'ENOENT',
  syscall: 'open',
  path: '/home/user/.redis-commander' }
No config found or was invalid.
Using default configuration.
No Save: true
listening on  0.0.0.0 : 8081
```

Searching for Packages

Searching for packages is straight forward, you can either use npm search searchterm, or you can go to the npm registry itself and search for search terms: https://www.npmjs.com/.

```
~/test $ npm search test
NAME                      | DESCRIPTION            | AUTHOR
test                      | (Un)CommonJS test...   | =gozala
react-test-renderer       | React package for...   | =clemmy...
chai                      | BDD/TDD assertion...   | =chaijs
invariant                 | invariant              | =cpojer...
selenium-webdriver        | The official...        | =jmleyba
enzyme                    | JavaScript Testing...  | =gdborton...
mocha                     | simple, flexible,...   | =scottfreecode
clean-css                 | A well-tested CSS...   | =goalsmashers.
jasmine-core              | Official packaging...  | =dwfrank...
test-exclude              | test for inclusion...  | =jakxz =bcoe
balanced-match            | Match balanced...      | =juliangruber
protractor                | Webdriver E2E test...  | =juliemr...
jasmine                   | Command line jasmine   | =slackersoft...
nyc                       | the Istanbul...        | =isaacs =bcoe
kind-of                   | Get the native type... | =doowb...
is                        | the definitive...      | =ljharb...
is-number                 | Returns true if the... | =realityking..
is-glob                   | Returns `true` if...   | =doowb...
```

```
deep-eql                  | Improved deep...       | =chaijs
should                    | test framework...      | =gjohnson...
```

Upgrading NPM

NPM itself is a package in the NPM repository, and you can use NPM to upgrade NPM.

To upgrade NPM to the latest release, perform the following command: sudo npm install npm@latest -g.

```
~/test $ sudo npm install npm@latest -g
[sudo] password for user:
/usr/bin/npm -> /usr/lib/node_modules/npm/bin/npm-cli.js
/usr/bin/npx -> /usr/lib/node_modules/npm/bin/npx-cli.js
+ npm@5.5.1
updated 1 package in 4.628s
```

> **NOTE**: Remember that -g means install globally (see end-user packages section). Since we have -g, we need to use sudo. The @latest after npm is what we use to define the version of npm that we want (see dependency versions section).