

NPM Cheat Sheet

Installing NPM

```
curl http://npmjs.org/install.sh | sh
```

Update NPM

There are several ways you can update NPM.

```
curl http://npmjs.org/install.sh | sh
```

or

```
npm install npm -g
```

Search for NPM packages

```
npm search hook.io
```

View details of a NPM package

```
npm view hook.io
```

Installing a NPM package locally

For the purpose of this demo, we will use `http-server`, a package we've written which provides an easy to use wrapper around NodeJS's core `http.Server` class.

This module makes for a good example, since it's API provides both a CLI binary and a requirable NodeJS module.

```
npm install http-server
```

This performs a local install of `http-server` in our current working directory.

You may also notice a new `node_modules/` folder. You can ignore this for now.

Installing a NPM package into an application

```
mkdir mynewapp/  
cd mynewapp  
npm install http-server  
touch test.js  
run script  
node test.js
```

Notice how we: `require('http-server')` - what kind of **wizardry** is *this*?

`http-server` is not the name of a native NodeJS module. It's the name of the package we just installed from NPM. NodeJS and NPM are smart enough to automatically load modules from our local

node_modules/ folder.

Understanding Global versus Local installs in NPM

By default, NPM will install all packages into the *local* directory you are working in. This is a good thing. It can however, be slightly confusing if you have worked with inferior package management systems in the past.

For example, if we:

```
mkdir anotherapp/  
cd anotherapp/  
touch test.js  
test.js
```

```
var HTTPServer = require('http-server');
```

and then run the script...

```
node test.js
```

we'll get this error:

```
node.js:134 throw e; // process.nextTick error, or 'error' event on first tick  
Error: Cannot find module 'http-server'  
    at Function._resolveFilename (module.js:326:11)  
    at Function._load (module.js:271:25)  
    at require (module.js:355:19)  
    at Object.<anonymous>  
    (/Users/maraksquires/dev/nodeapps/anotherapp/test.js:1:80)  
    at Module._compile (module.js:411:26)  
    at Object..js (module.js:417:10)  
    at Module.load (module.js:343:31)  
    at Function._load (module.js:302:12)  
    at Array.<anonymous> (module.js:430:10)  
    at EventEmitter._tickCallback (node.js:126:26)
```

This is logical, we installed `http-server` *locally* into `mynewapp/`, **not in** `anotherapp/`.

There are two direct solutions to fix this:

1. Install the package again, but locally into our new application

```
cd anotherapp/  
npm install http-server
```

2. Install the package globally

```
npm install http-server -g
```

Global Package Installation

If you want to have a package available globally use:

```
npm install http-server -g
```

The `-g` flag will indicate that `http-server` should be installed *globally*, and be available for all NodeJS scripts to `require()`.

Now, we can `require('http-server')` in any NodeJS script on our system.

In addition, since the `http-server` package has specified a `bin` property, it will also install a binary script called `http-server` *globally*.

Now you can simply run the command:

```
http-server
```

Uninstalling a package locally

```
cd mynewapp/  
npm uninstall http-server
```

Uninstalling a package globally

```
npm uninstall http-server -g
```

Installing a specific version of a package

```
cd mynewapp/  
npm install http-server@0.3.0
```

Cloning a module from Github

This is important. In some cases, there will be patches, forks, or branches that we will want to use for our module, but have not yet been published to NPM. Thankfully, the source code for most NPM modules is also available on Github.com

```
git clone git://github.com/nodeapps/http-server.git  
cd http-server/  
npm link
```

Our *cloned* version of `http-server` is now **linked locally**

Linking any NPM package locally

If you have a local directory containing an NPM package, you can link this package locally. This is good for development purposes and for situations when we do not want to publish our package to the public NPM repository.

```
cd http-server/  
npm link
```

Our *local* version of `http-server` is **linked** on our *local* machine

Linking local NPM packages to multiple applications

As we've seen before, NPM will install packages into the local directory by default. `npm link` works pretty much the same way.

```
mkdir newapp/  
cd newapp/  
npm link http-server
```

This indicates that we've now **linked** `http-server` into our new application `newapp`. If we had not run `npm link http-server` we would have gotten a missing module **error**.

Unlinking a NPM package from an application

```
cd newapp/  
npm unlink http-server
```

Unlinking a NPM package from your system

```
cd http-server/  
npm unlink
```

Create a new NPM package

```
mkdir mypackage/  
cd mypackage/  
npm init
```

Creating a new user account on NPM

```
npm adduser
```

Publishing a NPM package

```
cd mypackage/  
npm publish
```

Unpublishing a NPM package

```
npm unpublish http-server
```

Managing owners of packages

If you want multiple users to be able to publish to the same package:

```
npm owner add marak http-server  
npm owner rm marak http-server  
npm owner ls http-server
```