# NPM as a Build Tool

by Jennifer Bland

## Who am I?

http://www.jenniferbland.com
https://github.com/ratracegrad/
ratracegrad@gmail.com

# What is a Build Tool?

Pipeline converts application to deployable unit

Build Tool handles

| | |
|---|---|
| compile | bundle |
| test | build |
| minify | deployment |

YEOMAN

BOWER.IO

GRUNT

GULP

## Why use NPM?

Already using npm in your application
Keep all build scripts within your package.json
No additional configuration needed
Don’t have to rely on plugins

## Getting Started

```
npm init
```

## How to run npm Scripts

npm scripts are in the package.json

Execute by running either

npm run-script test

npm run test

## Well Known Scripts

npm built shortcuts for well-known Scripts

```
npm run test
npm test
npm tst
npm t
```

## Other Well Known Commands

```
npm start
npm stop
npm restart
```

## Pre and Post Hooks

A hook is a script that has the same name as another script but is prefixed with either pre or post.

“pretest”: “Echo “this is the pretest script\””,
“posttest”: “Echo “this is the posttest script\””

Use the -s flag to ‘silent’ the script

1-one

## Order of Hooks Irrelevant

You can put the scripts in any order in the package.json file. Npm will run them in the correct order.

## Build a Test

One of the most commonly used test libraries is Mocha.

npm install mocha should superset —save-dev

## Available Commands

```
ls node_modules/.bin
```

## Command to run Mocha

mocha

Default it looks for a test directory

Uses the reporter ‘spec’

2-two

## Linting Code

JSHINT is a tool that helps to detect errors and potential problems in your JavaScript code.

## Install JSHINT

```
npm install jshint —save-dev

jshint *.js **/*.js
```

## Building Lint Script

“lint”: “jshint *.js **/*.js

## Lint as pretest Script

Lint is a script that you should run frequently. This script is a good example of a script that should be run before your tests in a pre-hook script.

“pretest”: “npm run lint”

3-three

## Front End Scripts

compile less code to css
bundle and minify all scripts
compile coffeeScript and typeScript

From my example I will be  using Browserify to do the bundling and minify.

## Install Less

```
npm install less —save-dev
```

## Compiling Less to CSS

To compile less code you tell it where is your source less code and then where you want to output the CSS code that is generated

```
lessc client/less/demo.less public/css/site.css
```

## Create less Script

“build:less”: “lessc client/less/demo.less
	public/css/site.css”

## Browserify

```
npm install browserify —save-dev
```

## Running Browserify

Browserify is similar to Less. you tell it where is your sources code and then where you want to output bundled code that is generated.

```
browersify ./client/js/app.js -o ./public/js/bundle.js
```

## Create Browserify Script

```
“build:bundle”: “browersify ./client/js/app.js
    -o ./public/js/bundle.js”
```

## Minify

Minify is an npm module that will minify the bundle created by Browserify.

```
npm install uglify —save-dev
```

## Minify Script

```
uglify ./client/js/app.js —mc -o ./public/js/out.min.js
```

```
“build:uglify”: “uglify ./client/js/app.js —mc
    -o ./public/js/out.min.js”
```

## Refactoring Uglify

We don’t want to send the app.js file into uglify. We want to send the file generated by browserify into uglify. So let’s refactor our script to combine these.

```
“build:bundle”: “browserify ./client/js/app.js |
    uglifyjs =mc > ./public/js/bundle.js”
```

## Explaining the Script

The pipe command tells to take the output of the first command - browserify - and use that as the input to the next command - uglify.

The -mc parameter tells us to mangle and compress the files.

The greater than - > - command tells the output to be redirected to ./public/js/bundle.js

## Clean Code

Every time we run bundle we want to make sure that the old code is deleted before the new code is created. We can create a clean script to do this.

```
npm install rimraf —save-dev
```

```
“build:clean”: “rimraf public/css/*, public/js/*”
```

## Running the Clean Script

We want to run the clean script right before we run the bundle and minify scripts. We can accomplish this by creating a build script and then running the clean script in a “prebuild” script.

## Build Script

“build”: “npm run build:less && npm run build:bundle”
“prebuild”: “npm run build:clean”

4-four

## What scripts are available?

So far we have created almost a dozen scripts to handle linting, test, compiling less, bundle, minify and building.
How do you know what scripts are available?

```
npm run
```

## Server Side Scripts

During development process you need to have scripts that are watching for certain events to occur. Compiling Less code into CSS is an example of script that you can convert into a watch script. Every time there is a change to the less code then it compiles into CSS. All you need to do is refresh your browser to see the results of your code change.

## Server Side Watching

Some of the npm modules we have installed, have flags that will allow you to watch for changes. Mocha has this feature.

“mocha test -u bdd -R min -w”

I am using the min Reporter. The -w flag is the watch.

## Creating Test Watch Script

“watch:test”: “mocha test -R min -w”

## Refactor Watch Test

We can improve our watch script since it is using the same command as our test script.

“watch:test”: “npm run test — -w -R min”

The dash dash space allows us to pass through arguments to the underlying command which is mocha.

## Watching without Builtin Option

If an npm module does not have a watch option then you can use an npm module “watch” that can watch for them.

```
npm install watch —save-dev
```

# Watching Lint

We want a watch script that checks for any change to the code and then runs lint to make sure our code passes.

## Watch Lint Script

“watch:lint” “watch ‘npm run lint’ .”

This will watch for any changes to files in the current directory and if there is a change then run the lint script.

## Versioning

Usually when you do a deploy you want to change the version of your software. Most versioning uses semantic versioning.

# Semantic Versioning

Semantic versioning consists of MAJOR.MINOR.PATCH numbering system. You increment the

1. MAJOR when you make incompatible API changes
2. MINOR when you add functionality in a backwards compatible manner
3. PATCH when you make backwards compatible bug fixes

## Version Scripts

“version:major”: “npm version major”
“version:minor”: “npm version minor”
“version:patch”: “npm version patch”

## Pushing to Github

When you make changes to your code you want to push those changes to Github.

```
git push origin master
```

## Pushing to Github

“push:origin”: “git push —tags origin master”

## Deploying to Heroku

First download the heroku tool belt from
https://toolbelt.heroku.com

You create an application on heroic using command
heroku create appname

## Heroku Push Script

Creating a heroic application will create a new remote for you. You can verify that with the command

```
npm remote -v
```

# Heroku Push Script

To push code to production on Heroku you would use the command

```
git push heroku master
```

```
“push:heroku”: “git push heroku master”
```

## Push Script

When we deploy our code we want to push the latest version to github and to heroic
We can combine our two push scripts into one

“push”: “npm run push:origin && npm run push:heroku

# Final Deploy Script

Our Final Deploy Script should do the following:

1. Link, compile and test server side JavaScript
2. Bundle and minify client side JavaScript
3. Compile LESS to CSS
4. New version
5. Push to Github
6. Deploy to Heroku

## Deploy Script

```
“deploy:prod”: “npm run test -s &&
  npm run build -s &&
  npm run version:patch &&
  npm run push
```

5-five

## Limitations of using npm

No comments

Commands can be different on Mac vs Windows

Harder to understand

## Slide and Code

github.com/ratracegrad/npm-as-a-build-tool

## NPM Config Variables

npm has a config directive which allows you to set values that can be picked up as environment variables in your scripts

## NPM Config Variables

```
“config”: {
    “reporter”: “landing”
},
“scripts”: {
    “test”: “mocha -r $npm_package_config_reporter”,
    “test:dev”: “mocha”
}
```