Gulp

NPM Packages

Gulp Tutorials (multi-part)

MahsupJS’s Gulp

# NPM Packages

## Commands Cheat Sheet

* Find outdated modules
  + npm outdated -–depth=0
  + npm outdated –-json -–depth=0
* Updating a package
  + npm install grunt-contrib-uglify@\* --save-dev

## PowerShell (primer)

Windows users can use either the Command Prompt or PowerShell.

PowerShell is pre-installed on Windows 8 or newer installations.

From Start: *Search programs and files* type “powershell”. Select “powershell.exe”.



### Syntax

Command Syntax: **application action –flags arguments**

For help with any application add the **–h** or **–help** flags for additional instructions.

The **tab** key autocompletes your statement.

### Adding and removing files

To create a new item use the **ni** command. This might not seem useful with Visual Studio 2013 because any file added must also be added to your project file. Visual Studio 2015 does nto have a project file needing updates. Instead a Glob pattern is used to determine what files should and should not be included in the project. That being the case, suddenly, **ni** makes more sense.

Example: ni newjsfile.js -type file

Or : new-item newjsfile.js –type file

Removing files: ri newjsfile.js or remove-item newjsfile.js

## Installing NodeJS and NPM Packages

Install NodeJS from:

<https://nodejs.org/>

<https://iojs.org/en/index.html>

Install NPM packages with the following syntax

npm install [global option –g] [package-name] [options]

Exampe: npm install gulp --save-dev

More information can be found in the Gulp Tutorial: Setting up Gulp

<https://docs.npmjs.com/getting-started/installing-npm-packages-locally>

## Version updates

**How to configure for version updates**

There are multiple options for keeping NPM packages up to date. The approach you choose might depend on your development workflow and automated testing solution. IE: If you have good automated testing it might be safe to allow the latest versions. If not then you might want to choose a more deliberate approach to npm versioning.

## Version updates: Option 1 – Using node tools

**Check to see which npm packages are out of date**

Display colored rows

npm outdated -–depth=0

Display in json which includes current, wanted, latest version numbers

npm outdated –-json -–depth=0



Note: Not all your packages will be displayed. Only the outdated packages.

Note: If you modify the command to include “-g” then you’ll get a list of your outdated global packages.

**To update packages one at a time**

npm install [package-name]@\* [save?]

npm install grunt-contrib-uglify@\* --save-dev

## Version updates: Option 2 – Using npm-check-updates

Using the npm-check-updates package you can keep all your packages updated.

<https://www.npmjs.com/package/npm-check-updates>

npm install -g npm-check-updates

Then execute the following command to see what packages can be updated.

npm-check-updates



To upgrade all your packages

npm-check-updates –u [-g option for global packages]

Now your package.json is updated.

Then execute an npm install to update the package installations.

npm install [-g option for global packages]

## npm versioning semantics

<https://docs.npmjs.com/misc/semver>

<http://semver.org/>

# Gulp Tutorials

## GULP Tutorial Part 1 – Reasons for Build Tools like Gulp

### Productivity

**The reason for a build system is always productivity. Otherwise we wouldn’t invest time into it.**

**Build systems perform house cleaning work allowing you to focus on code. Before build systems, if you were lucky, you could right click and select “minify” in your IDE. As lucky as this might have been minification might not have been worth the additional development effort required. Build systems solve this.**

**Build systems perform tasks with a level of precision humans are incapable of. In order for Continuous Integration and Continuous Delivery to work a build system must be used to keep the human element out. Continuous Delivery requires automation at all levels including testing to mitigate common deployment defects.**

**There are thousands of plugins to perform just about any task imaginable. Here are a few.**

**Performance/Optimization**

* **Minification of JavaScript files**
* **Minification of CSS files**
* **Slimming down CSS classes that are not used**
* **Concatenating many JavaScript files to reduce get requests**
* **MAP files created for debugging at run-time**

**Deployment**

* **Files can be optimized then copied to a folder to isolate deployment from development**
* **A zip file can be generated for deployment**
* **Automated tests can be executed**
* **Create deployments with a particular purpose; IE: an app can be built for mobile**

**Static Analysis**

* **Linters can be executed against your code producing advice**
* **Cyclomatic complexity and other measures can be generated.**

**Documentation**

* **Documentation can be generated from code into readable formats.**
* **HTML documents can be generated from Markdown, a popular text format.**

**Additional resources**

<https://www.youtube.com/watch?v=XJ5F-Auhato>

## GULP Tutorial Part 2 – Gulp Setup

Download and install NodeJS or IoJS. IoJS is a fork of NodeJS by many in the community who wanted more responsive Node advancement. Theoretically you should be able to use either for the Mashup.

<https://nodejs.org/>

<https://iojs.org/en/index.html>

npm packages are defined in the package.json file.

To create a package.json file

npm init

**Installing NPM Packages**

At this point Gulp can be installed with the following command. Notice the “-g” command. This causes the npm package to be deployed globally.

Installing Gulp (add “-g” to install globally)

npm install -g gulp

For more information on installing npm packages  
<https://docs.npmjs.com/getting-started/installing-npm-packages-locally>

Create a basic gulpfile.json with the following content, in the root of your project. Add the following scaffolding to the new gulpfile.json file.

var gulp = require('gulp');

gulp.task('default', function() {

// place code for your default task here

});

## GULP Tutorial Part 3 – Adding Plugins

Plugins provide function to the task runner.

TODO: How to search for plugins. Consider the number of downloads and activity on github when trying to determine which plugin to choose.

Install a few useful plugins.

npm install gulp --save-dev

npm install gulp-uglify --save-dev

npm install gulp-rename --save-dev

npm install gulp-rename --save-dev

npm install gulp-sourcemaps --save-dev

To add these plugins to your gulp implementation at this to the beginning of your gulpfile.json.

var gulp = require('gulp')

, uglify = require('gulp-uglify')

, rename = require('gulp-rename')

, sourcemaps = require('gulp-sourcemaps')

, concat = require('gulp-concat');

Add a variable for your destination.

var dist = '/dist';

Syntax for creating a task

Gulp.task([task-name], function() {

Return gulp.src([glob-array]

.pipe([your-plugin])

.pipe([another-plugin])

.pipe(gulp.dest(dist));

});

## GULP Tutorial Part 4 – Creating an example application

TODO: Use a Yo or Yoemen Generator to create the MashupJS sample project.

TODO: For now use the steps outlined in the main ReadMe.md

## GULP Tutorial Part 5 – Optimizing JavaScript

NOTE: Copy to dist, Concat, uglify (compress and minify), rename, maps, JSHint, consider series/parallel

TODO: Add all client scripts to **bower** so that we aren’t copying unnecessary files to the **dist** directory.

In this part of the tutorial we are setting up our first gulp task.

### Clean out ‘dist’

First, let’s clean out our ‘dist’ directory so we’re starting fresh. Execute the following from PowerShell.

npm install gulp-clean --save-dev

Add the following code to your gulpfile.js

var gulp = require('gulp')

, clean = require('gulp-clean');

var distFolder = 'dist';

gulp.task('clean', function () {

return gulp.src(distFolder, { read: false })

.pipe(clean());

});

gulp.task('default', ['clean']);

Now, from the command line you can execute the default task which includes cleaning out the ‘dist’ directory. If you don’t have any files in your ‘dist’ directory or even the directory itself you can simply create the ‘dist’ directory and a few files for testing.

mkdir dist

ni dist/newjsfile.js -type file

ni dist/newjsfile2.js -type file

Now run the gulp command and watch the ‘dist’ directory disappear.

### Copy all \*.js, \*.css, and \*.html, images, and lib directory to ‘dist’

We will keep our source (‘scr’) code separate from our distribution code so we don’t pollute our development environment. When executing the application we’ll set the index.html file of the ‘dist’ directory as the startup. This approach might seem to introduce challenges because when debugging you’ll need the ability to read the compressed and minified versions of JavaScript and CSS. Gulp will give us that ability.

Now that we’ve cleaned out the ‘dist’ directory in preparation for new files lets go ahead and copy all our source code to ‘dist’. Once we’ve copied the source code we can begin running tasks to optimize our code.

No new npm modules or gulp dependencies are required for the copy operation. Built in features, gulp.src and gulp.dest are enough.

Add the following new task to your gulpfile.js

gulp.task('copy', ['clean'], function () {

return gulp.src(srcFolder + '/\*\*/\*')

.pipe(gulp.dest(distFolder))

});

Notice the [‘clean’] value in the second parameter. This is defining a dependency. Because Gulp defaults to the highest performance option, concurrency, the task run in parallel. In this case the copy cannot occur until the clean operation is complete. Otherwise we’d be deleting files as we add them and that not very useful. The copy task will not begin until the clean task completes.

Execute the Gulp command and the dist directory will be deleted and rebuilt.]

gulp

### Annotation

Before we concatenate and minify let’s make sure our Angular code is in good shape. To make a long story short… Angular is based, largely, on the ability to directly inject dependencies. This ability is made possible because the name of the injected dependency is interpreted. As soon as a file is minified that name is change to ‘a’ or ‘b’ or whatever the next available small variable name is available. This breaks dependency injection.

Passing the dependency name as a string corrects this problem because static string names are not minified. You can do this yourself of let Gulp run a task to do this for you. Even if you decided to handle this while writing code it’s a good idea to run an annotation task anyway. Just in case.

From the command line install the gulp-ng-annotate plugin

npm install gulp-ng-annotate --save-dev

Add this module to your gulp modules list

, annotate = require('gulp-ng-annotate')

Add the annotate task to your gulp file that will update the source code with good uri links.

gulp.task('annotate', function () {

return gulp.src([srcFolder + '/index.controller.js', srcFolder + '/core/\*\*/\*.js', srcFolder + '/apps/\*\*/\*.js'])

.pipe(ngAnnotate())

.pipe(gulp.dest(srcFolder));

});

More information on gulp-ng-annotate

<http://christian.fei.ninja/DRY-dependency-injection-in-Angular-with-gulp-ng-annotate/>

### Concatination

Before we minify our JavaScript files let’s see if there are any files we wish to combine. We could simply minify all JavaScript files then concatenate them together but then we would lose the ability to map minified code to source code for debugging purposes. So we will concatenate together any code we desire and then execute a general minification task.

First install a concatenation plugin

npm install gulp-concat --save-dev

npm install gulp-rename --save-dev

npm install gulp-uglifyjs --save-dev

npm install gulp-sourcemaps --save-dev

Add the new plugins to your gulp required list

var gulp = require('gulp')

, clean = require('gulp-clean')

, concat = require('gulp-concat')

, uglify = require('gulp-uglifyjs')

, rename = require('gulp-rename')

, sourcemaps = require('gulp-sourcemaps');

Create a task that combines all the **core/common files** into one file then compresses and minifies the new file and creates a map file.

gulp.task('coreservices', ['copy'], function () {

return gulp.src(srcFolder + '/core/common/\*\*/\*')

.pipe(uglify('core.services.js', {

mangle: false,

output: {

beautify: true

}

}))

.pipe(gulp.dest(distFolder))

.pipe(uglify('core.services.min.js', {

outSourceMap: true

}))

.pipe(gulp.dest(distFolder))

});

Create a task that combines all the **route.config**.js files together.

NOTE: This capability to combine the route config is what makes the drop in application style of MashupJS work.

gulp.task('routeconfig', ['copy'], function () {

return gulp.src([srcFolder + '/core/config/route.config.js', srcFolder + '/apps/\*\*/route.config.js'])

.pipe(uglify('route.config.js', {

mangle: false,

output: {

beautify: true

}

}))

.pipe(gulp.dest(distFolder))

.pipe(uglify('route.config.min.js', {

outSourceMap: true

}))

.pipe(gulp.dest(distFolder))

});

Now combine all your **bower libraries** together.

gulp.task('libs', function () {

gulp.src(['bower\_components/\*\*/\*.js'])

.pipe(uglify('libs.js', {

mangle: false,

output: {

beautify: true

}

}))

.pipe(gulp.dest(distFolder + '/libs'))

});

For additional documentation on gulp-uglifyjs: <https://www.npmjs.com/package/gulp-uglifyjs>

### Compress and Minify JavaScript

It’s finally time to minify all the JavaScript files.

## GULP Tutorial Part 6 – Optimizing CSS

NOTE: Copy to dist, Concat, uglify (compress and minify), rename, maps, linter for css, remove unused classes, consider series/parallel

## GULP Tutorial Part 7 – Optimizing HTML

NOTE: copy to dist, uglify (compress and minify), rename, maps, linter for html, consider series/parallel

## GULP Tutorial Part 8 – JSON

NOTE: Merge multiple JSON docs. Make note of how duplication is handled. Experiment with data to see when duplication is allowed and when it is handled. Review documentation to see what things people are doing with it. Explain how it manages our dynamic menu system and configurations (webapi uri(s)).

## GULP Tutorial Part 9 – Images

NOTE: copy to dist, optimize, rename, and consider series/parallel

## GULP Tutorial Part 10 – Watch

NOTE: watch newer only, make not of the performance over Grunt, and consider series/parallel

## GULP Tutorial Part 11 – TypeScript

## GULP Tutorial Part 12 – SASS

## GULP Tutorial Part 13 – ES6

## GULP Tutorial Part 14 – Initiate tests

## GULP Tutorial Part 15 – Useful Gulp Commands

**Installing gulp**

npm install --save-dev gulp-install

**Retrieve gulp Version**

Grunt version

**Installing plugins**

The syntax for Grunt plugins is

install [plugin-name] --save-dev

For example if you want to minify and concatenate your JavaScript for performance you would install two plugins.

Perform a quick google search and you’ll find this site

<https://github.com/gruntjs/grunt-contrib-uglify>

npm install grunt-contrib-uglify --save-dev

Perform a quick google search and you’ll find this site

<https://github.com/gruntjs/grunt-contrib-concat>

npm install grunt-contrib-concat --save-dev

## GULP Tutorial Part 16 – gulp Tips

**Retrieve Gulp version**

Gulp --v

**Every Gulp file needs a default task. To execute Gulp’s default task**

grunt

**It’s useful to run specific tasks that you have configured**

grunt [task-name]

**Get a list of grunt commands**

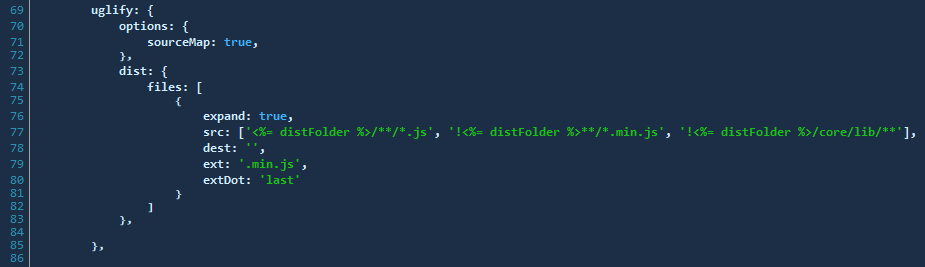
Grunt –help

**~~Last dot~~**

~~When creating new files and adding a new extension Grunt gets confused about which dot is the last dot designating the extension. If your file has multiple dots then you won’t get the result you’re looking for when adding “min.js” for example, to minified JavaScript files.~~

~~my.JavaScript.js would be renamed to my.js.~~

~~Add to your configuration: extDot: ‘last’~~

~~~~

## GULP Tutorial Part 17 – Glob Tips

“dir/\*” – includes all files

“dir/\*\*” – includes all files and directories

# MashupJS gulp Implementation

The Mashup will maintain both Grunt and Gulp. You can use whichever works best for your workflow.

Grunt is a task runner used in the development and the build processes. All files are processed and distributed to the “/dist” directory. Everything needed for deployment should be found here.

To make this work there are a few things you need to do. If you are using VS2015 then some of this will happen auto-magically.

This document will walk you through the basic setup and usage of Grunt, how Grunt is configured for the Mashup, and how to use Grunt as part of the Mashup workflow.

TODO: Explain Mashup’s Grunt implementation