Reproducible Configs

A lot of the projects that I work with use a fairly standard setup of tools and scripts. Rather than reinstall them every time and have to fix paths and install dependencies outside the Node ecosystem I've created a script that will perform the following actions:

1. Check if `package.json` exists. If it doesn't then create one with default values
2. Install packages explaining what it's doing when installing each group
3. Run an additional script that will create configuration files
4. Remove the .git directory and recreate it for the current directory

This way I can initialize a new project using 2 files that will create a reproducible starting point yet retaining a degree of flexibility.

That's why we don't force an ESLint configuration. If I'm writing React I'll use Airbnb preset as it is geared towards working with React. For most other projects I'll use Google's preset; it is what I became comfortable with when developing PWA content and I think it's still the one that makes most sense.

There are two directories and a file that I've included to use for my type of projects. I create a `src` directory with the structure of files that I will use in the Gulpfile.

The first thing we do is check if there is a `package.json` file in the directory we're running the script in. If there is one we exit. I'd rather not overwrite the file that is there.

If there isn't one we initialize a default package.json file that we'll install packages to.

```bash
#!/usr/bin/env bash

echo "Checking if package.json exists"
if [ -f package.json ]
  then
    echo "package.json exists."
    exit 1
  else
    echo "package.json doesn't exists... creating"
    npm init --yes
fi
```

The rest of this script installs different sets of packages. I will only highlight any specifics that I think are important.

```
echo "installing NPM packages "

echo "Installing Gulp 4.0 and load-plugins"
npm i -D gulp@next gulp-load-plugins
```

The first big change I've made is to fully embrace Gulp 4.0. I will modify the file to take into account the new methods it uses.

```
echo "Installing PostCSS and related plugins"
npm i -D postcss autoprefixer cssnano gulp-postcss

echo "Installing Babel Core and Preset Env"
npm i -D @babel/core @babel/preset-env

echo "Installing Browser Sync"
npm i -D browser-sync connect-history-api-fallback

echo "Installing ESLint"
npm i -D eslint gulp-eslint
```

It's important to notice that we're only installing eslint. We'll handle the preset installation later

```
echo "Installing SASS and related plugins"
npm i -D gulp-sass gulp-sourcemaps

echo "Installing critical and uncss"
```

```
npm i -D critical uncss

echo "Installing Remarkable and related plugins.  I use Remarkable and ass
npm i -D remarkable gulp-remarkable gulp-wrap
```

I use Markdown as my authoring tool. I then use `Remarkable` and `gulp-remarkable` to convert it to HTML and `gulp-wrap` to put the content inside a template that has links to the CSS and Javascript that I use for particular projects. All we need to do to use different assets is to change the links in the template.

```
echo "Installing miscelaneous plugins"
npm i -D  gulp-concat gulp-debug gulp-exec gulp-run gulp-size

echo "The following plugins are optional. "

echo "Installs Google Cloud utilities to upload content to google cloud st
npm i -D gulp-gcloud-publish gulp-gzip
```

The original project created PDF assets that were the uploaded to a Google Cloud storage bucket. I haven't decided if I want to continue using GCloud Storage that way or if I want to move to Firebase hosting instead.

```
echo "Utility Opentype adds classes for using Opentype font options"
npm i -D utility-opentype

echo "Cheerio provides the means to query HTML and XML structures from a C
npm i -D cheerio

echo "Axe is an accessibility evaluation tool."
npm i -D gulp-axe-webdriver

echo "Evaluates your node_modules file for vulnerabilities.  This is the t
npm i -D gulp-snyk snyk

echo "External Dependecies Required"
echo "SASSDoc and SCSS Lint depend on external Ruby dependencies."
```

```
npm i -D sassdoc scss-lint gulp-scss-lint gulp-scss-lint-stylish

echo "Running configuration script"
sh configure.sh
```

Rather than make the script longer than it needs to be I call a second script to do configuration. This script is more complex and uses some Bash tricks.

```
#!/usr/bin/env bash

# Configure ESLint
echo "ESLint configuration"
echo "Checking i"ESLint configuration"s.  We're assuming it was written as
if [ -f .eslintrc.js ]
  then
    echo ".eslintrc.json exists."
    exit 1
  else
    echo ".eslintrc.json not found. Creating... "
    echo "Please answer the questions below:"
    npx eslint --init
fi
```

If the `.eslintrc.js` file exists then we tell the user and exit. If it doesn't exist then we run npx `eslint --init` to configure eslint based on our preferences and the needs of the project.

```
# Creating .babelrc file
if [ -f .babelrc ]
  then
    echo ".babelrc exists"
    ex".babelrc exists"cho "creating .babelrc"
fi

touch .babelrc
cat > .babelrc <<- "creating .babelrc""EOF"
```

```
{
  "presets": [
    ["env", {
      "targets": {
        "browsers": ["last 2 versions", "safari >= 7"]
      }
    }]
  ]
}
EOF
```

Configuring babel is slightly different. We test like we did with ESLint. When it doesn’t then we create the file using the `touch` command and then append the content of the file.

```
# Create and populate .editorconfig
if [ -f .editorconfig ]
  then
    echo ".editorconfig exists"
    exit 1
 ".editorconfig exists"rconfig doesn't exist"
    echo "creating..."
fi

touch .editorconfig
cat > .editorconfig <<- "EOF"
# http://editorconfig.org
root = true

[*]
ind't exist"
    echo "creating..."
fi

touch .editorconfig
cat > .editorconfig <<- "EOF"
# http://editorconfig.org
```

```
root = true

[*]
indent_style = space
indent_size = 2
end_of_line = lf
charset = utf-8
trim_trailing_whitespace = true
insert_final_newline = true

# Use 4 spaces for the Python files
[*.py]
indent_size = 4
max_line_length = 80

# The JSON files contain newlines inconsistently
[*.json]
insert_final_newline = ignore

# Minified JavaScript files shouldn'indent_style = tab

# Batch files use tabs for indentation
[*.bat]
indent_style = tab

[*.md]
trim_trailing_whitespace = false
EOF
```

editorconfig provides a standard way to tell your editor how to behave. The (defacto) standard is supported by most modern text editors.

The file we're adding to the project is a basic default that we can use for most projects. Check the website for additional information.

```
# Git configuration has to be the last step to make sure
# that we get all the files we want are committed to the repo
```

```
echo "Reseting GIT configuration.Because I downloaded this as a Git Repo 
echo "If"Reseting GIT configuration.Because I downloaded this as a Git Rep
rm -rf .git
echo "creating empty git repository"
git init
echo "creating .gitignore"
touch .gitignore
echo "adding node_modules and shell files to .gitignore"
"
rm -rf .git
echo "# >> .gitignore
echo "install.sh" >> .gitignore
echo "configure.sh" >> .gitignore
#
echo "adding files"install.sh"#
echo "adding files to repository"
git add .
echo "commit initial files"
git commit -am "initial commit"
```

Working with Git in this setup is the last step in the process. It is important to note that if you're working on an existing project this will blow up your project's local history.

The first step of the process is to remove the existing `.git` directory. I choose to do it tto make sure that the next steps are done on what, for Git, is a clean repository.

Next we create a `.gitignore` file and add the files we don't want to upload to Git. In this case I've chosen not to upload the shell (`install.sh` and `configure.sh` files and the `node_modules` directory.

Then we add the files to the repository and commit them. All that is left is to push them to your Git or Github repo.

In a later post I will discuss the `gulpfile.js` build file and some alternatives for how to run build tasks without Gulp.