# Setting Up And Linting

In a previous post I documented how to typecheck Javascript using Typescript tools. The next step in migrating a codebase to Typescrript is to install Typescript and the Typescript linter.

Since Palantir deprecated TSLint there are multiple ways to compile and lint Typescript code. This post will illustrate two of them.

## ESLint

The recommended way to lint Typescript code is ESLint and the Typescript ESLint Plugin.

To install the tools, run the following command to install the necessary packages.

```
npm i --save-dev eslint typescript @typescript-eslint/parser @typescript-e
```

Next, we need to create a configuration file to use the plugins and parsers that we just installed. I chose to write the configuration as a Javascript file.

The `parser` attribute configures the parser that we want to use. In this case we use the Typescript parsers.

Same thing with the `plugins` section of the configuration tells ESLint to use the `typescript-eslint` plugin.

The final portion of the configuration tells ESLint what rules to extend and use: The default ESLint recommended rules and the Typescript recommended rules.

```
module.exports = {
  root: true,
  parser: '@typescript-eslint/parser',
  plugins: [
    '@typescript-eslint',
  ],
```

```
  extends: [
    'eslint:recommended',
    'plugin:@typescript-eslint/recommended',
  ],
};
```

The last item to consider is to create an `.eslintignore` file to tell ESLint to ignore files and directories.

We don't want to lint `node_modules` as they won't be a part of our application.

We also don't want to lint the resulting code in `dist` and any information in `coverage`. You can add other directories as needed.

```
node_modules
dist
coverage
```

We could also run `tsc --init` to initialize a basic Typescript configuration if we haven't done so already.

# GTS

The Node.js team at Google makes GTS available as an opinionated toolkit to work with Typescript. This is an alternative to using and configuring ESLint on its own that compilation, linting and formatting.

The first thing to do is to run the following npx command to initialize the project:

```
npx gts init
```

This will generate the followiing files.

```
.
├── .eslintignore
├── .eslintrc.json
├── .prettierrc.js
├── node_modules/
├── package-lock.json
├── package.json
├── src/
|   └── index.ts
└── tsconfig.json
```

Note that, under the hood, GTS will install Typescript to compile, Prettier to format and ESLint to lint Typescript code.

> It makes the following commands available:

- **npm run lint** lints the code
- **npm run clean** cleans up directories
- **npm run compile** compiles Typescript using `tsc`
- **npm run fix** fixes code using `gts fix`
- **npm run prepare** and **npm run pretest** are aliases for `npm run compile`
- **npm run posttest** is an alias for `npm run test`

GTS is a set of tools with opinionated defaults. If you need to change or extend them you can do so by working on the top level `.prettierrc.js` and `eslintrc.json` configuration files.