Welcome to the webpack test suite!!!!

Every pull request that you submit to webpack (besides README and spelling corrections in comments) requires tests that are created.

But don't give up hope!!! Although our tests may appear complex and overwhelming, once you become familiar with the test suite and structure, adding and creating tests will be fun and beneficial as you work inside the codebase! ♥

tl;dr

Run all tests (this automatically runs the setup):

```
yarn test
```

Run an individual suite:

```
yarn jest ConfigTestCases
```

Watch mode:

```
yarn jest --watch ConfigTestCases
```

See also: Jest CLI docs

Test suite overview

We use Jest for our tests. For more information on Jest you can visit their homepage!

Class Tests

All test files can be found in *.test.js. There are many tests that simply test APIs of a specific class/file (such as `Compiler`, `Errors`, Integration, `Parser`, `RuleSet`, Validation). If the feature you are contributing involves one of those classes, then best to start there to understand the structure.

xCases

In addition to Class specific tests, there are also directories that end in "Cases". The suites for these cases also have corresponding *.test.js files.

cases (`TestCases.test.js`) [1]

Cases are a set of general purpose tests that will run against a variety of permutations of webpack configurations. When you are making a general purpose change that doesn't require you to have a special configuration, you would likely add your tests here. Inside of the `./test/cases` directory you will find tests are broken into thematic sub directories. Take a moment to explore the different options.

To add a new case, create a new directory inside of the top level test groups, and then add an `index.js` file (and any other supporting files).

By default this file will be the entry point for the test suite and you can add your `it()` 's there. This will also become bundled so that node env support happens as well.

configCases (`ConfigTestCases.basictest.js`) [1]

If you are trying to solve a bug which is reproducible when x and y properties are used together in a config, then configCases is the place to be!!!!

In addition to an `index.js`, these configCases require a `webpack.config.js` is located inside of your test suite. This will run this specific config through `webpack` just as you were building individually. They will use the same loading/bundling technique of your `it()` tests, however you now have a more specific config use cases that you can write even before you start coding.

statsCases (`StatsTestCases.basictest.js`)

Stats cases are similar to configCases except specifically focusing on the `expected` output of your stats. Instead of writing to the console, however the output of stats will be written to disk.

By default, the "expected" outcome is a pain to write by hand so instead when statsCases are run, runner is checking output using jest's awesome snapshot functionality.

Basically you don't need to write any expected behaviors yourself. The assumption is that the stats output from your test code is what you expect.

Please follow the approach described below:

- write your test code in `statsCases/` folder by creating a separate folder for it, for example `statsCases/some-file-import-stats/index.js`

```
import("./someModule");
```

- don't forget the `webpack.config.js`
- run the test
- jest will automatically add the output from your test code to `StatsTestCases.test.js.snap` and you can always check your results there
- Next time test will run -> runner will compare results against your output written to snapshot previously

You can read more about SnapShot testing right here

Questions? Comments?

If you are still nervous or don't quite understand, please submit an issue and tag us in it, and provide a relevant PR while working on!

Footnotes

[1] webpack's parser supports the use of ES2015 features like arrow functions, harmony exports, etc. However as a library we follow Node.js' timeline for dropping older versions of node. Because of this we expect your tests on GitHub Actions to pass all the way back to NodeJS v10; Therefore if you would like specific tests that use these features to be ignored if they are not supported, then you should add a `test.filter.js` file. This allows you to import the syntax needed for that test, meanwhile ignoring it on node versions (during CI) that don't support it. webpack has a variety of helpful examples you can refer to if you are just starting out. See the `./helpers` folder to find a list of the versions.