COMP1531

✅ Correctness - Linting

Lecture 3.4

Author(s): Hayden Smith

(Download as PDF)

In This Lecture

- **Why? 🤔**
  - You can't manually fix your style forever - we need automated approaches
- **What? 📰**
  - Linting
  - eslint

😎 Coding Style

"Programs must be written for people to read, and only incidentally for machines to execute" — *Abelson & Sussman, "Structure and Interpretation of Computer Programs"*

😎 Coding Style

- Why do we care about style?
  - It's easier to follow the flow of code with consistent whitespace.
  - It's easier to visually glance at code with similar patterns.
  - It can be easier to detect bugs.
    - E.G. If you force constants to be uppercase named, it's easy to spot a mutated uppercase variable.

😎 Coding Style

Bad

```
/* eslint-disable */
function a(b,c){
 let d = '';
  for
   (let i = 0; i < b; i++)
d += c;
 return d;
}
console.log(a(5, 'hello '));
```

style_bad.js

Good

```
function manyString(repeat, str) {
  let outString = '';
  for (let i = 0; i < repeat; i++) {
    outString += str;
  }
  return outString;
}
console.log(manyString(5, 'hello '));
```

style_good.js

🧐 Well Who Decides On Style??

Typically when a group picks a style, they:

1. Choose a style guide set out by a large organisation (E.G. Google, Facebook, Microsoft, etc).
2. (Optionally) modify specific style rules to satisfy any clear subjective opinions (e.g. if most of an organisation strongly believes in spaces over tabs).

In COMP1531, the staff use standard style guides and provides them to students.

Linters: Enforcing Style

In early computing courses you're given a style guide and told to manually make sure your code complies to style.

In proper software engineering projects we tend to "lint" code by using software that statically analyses your code and automatically makes adjustments.

We call programs that lint code "linters". You can loosely consider them another form of **static verification**.

🤩 What Does A Linter Do?

Recap: Static verification is the processing of analysing as much of your program as you can before running it.

E.G. C compilation contains elements of static analysis (e.g. type checking , unused variables, etc)

A linter does static analysis to identify:

- Style issues (whitespace, indentation)
- Semantic issues (bad logic, potential bugs)

🤩 What Does A Linter Do?

Linting does not help with poorly named variables. That's still up to humans.

Because JS is interpreted (no compile step) linting helps bridge the gap of some things missed out by compilers.

🧠 eslint

- A popular external tool for statically analysing javascript code
- Can detect errors, warn of potential errors and check against conventions
- Can also automatically fix issues with your style
- Can be configured to be as strict or lenient as desired.

🧠 eslint

Installation

```
npm install --save-dev eslint
```

Once again, we use `--save-dev` since the library is only used in development and not for production code.

🧠 eslint

Configuration

`eslint` determines what is and isn't OK by looking at a `.eslintrc` file. In our case we're using a file named `.eslintrc.js`.

There are tools that will help you create your own! But for COMP1531 we are providing one.

🧠 eslint

Using `eslint`

You can run `eslint` on a file by the following command:

```
node_modules/.bin/eslint 3.4_style_bad.js
```

If nothing prints on the terminal, your file is all good!

Semantic issues need to be fixed manually (e.g. undefined variable).

Style issues can be fixed manually...

🧠 eslint

Automatically Fixing Style Issues

If we add a `--fix` flag to our eslint command, eslint will be able to fix style issues for us.

```
node_modules/.bin/eslint --fix 3.4_style_bad.js
```

This **overwrites** the file.

🧠 eslint

Ignoring One-Off Issues

If you want to disable `eslint` rules for one-off instances, you can disable certain rules with comments as per the eslint disabling rules guide.

```
1 const a=5;
2 // eslint-disable-next-line
3 const b=5;
```

In COMP1531 we only allow the use of a **few** `eslint-disable-next-line` directives per project, but they NEED to be checked with your tutor first.

Using It In Your Project

You will want to add the following to your `package.json` scripts.

```
{
  "scripts": {
    "lint": "eslint \"**/**.js\"",
    "lint-fix": "eslint --fix \"**/**.js\""
  }
}
```

This allows us to run `npm run lint` to check our code, and `npm run lint-fix` to check and fix (where it can).

The reason you can't lint-fix all the time is because it is slower.

Using It In Your Project

You can then add the following to your `.gitlab-ci.yml` file (including the tests).

```yaml
image: comp1531/basic:latest

cache:
  paths:
    - node_modules

stages:
  - checks

testing:
  stage: checks
  script:
    - npm run test

typecheck:
  stage: checks
  script:
    - npm run tsc

linting:
  stage: checks
  script:
    - npm run lint
```

gitlab-ci_lint.yml

 Linting In COMP1531

`eslint` will be part of your labs from week 4. We will give you both the configuration file and add a lint command to your `package.json` scripts as well as your `.gitlab-ci.yml`.

`eslint` will be part of the project from iteration 2. We will give you the configuration file but **expect you** to add a lint command to your `package.json` scripts and to your `.gitlab-ci.yml`.

🛠️ Finishing The Eslint Setup

All of the examples in this lecture were relatively simple.

We sadly need to do slightly more than `npm install --save-dev eslint` to get things working with `jest` and `typescript`.

🛠 Eslint With `jest`

1. Run `npm install --save-dev eslint-plugin-jest`
2. Add the following to `.eslintrc.js`:

```
{
  "plugins": ["jest"],
  "env": { "jest": true }
}
```

🛠️ Eslint With `typescript`

1. Run `npm install --save-dev eslint typescript @typescript-eslint/parser @typescript-eslint/eslint-plugin`
2. Add the following to `.eslintrc.js`:

```
{
  parser: '@typescript-eslint/parser',
  plugins: [
    '@typescript-eslint'
  ],
  extends: [
    'eslint:recommended',
    'plugin:@typescript-eslint/recommended'
  ]
}
```

3. You might then also want to modify scripts `lint` and `lint-fix` that just look for `.ts` files instead of `.js` files to your `package.json` scripts.

🛠 Eslint With `typescript`

This now means it's possible to `eslint` your `.ts` files very easily!

Let's try it out.

🤗 Don't Stress!

All of the environmental setup or changes you've seen in this lecture will either be **done for you** or will be given to you with clear unambiguous instructions.

We don't expect you to all be experts in tweaking these environments.

🦻 Feedback

Or go to the form here.