npm and Linting Basics

Learning Objectives

- [ ] What npm is and how it is used
- [ ] What are packages and how does the npm ecosystem work
- [ ] The basic anatomy of a npm package
- [ ] How does semantic versioning work
- [ ] What are linters and how can we use them
- [ ] Error messages are your friend

---

npm

It's a package registry that works like an app store for your project.

`package.json`

Packages that are installed into your project are called dependencies. They are kept inside a `package.json` file in your project root. The `package.json` file also contains information about your project.

A `package.json` may look something like this:

```
{
  "name": "my-app",
  "version": "1.0.0",
  "description": "A description of my app",
  "scripts": {
    "test": "npm run …"
  },
  "author": "Alex Newfish",
  "license": "UNLICENSED",
  "dependencies": {
    "my-dependency": "^10.4.1",
    "my-other-dependency": "^2.0.0"
  },
  "devDependencies": {
    "my-dev-dependency": "^8.0.105",
    "my-other-dev-dependency": "^0.1.6"
  }
}
```

- `dependencies` are packages that your application source code directly depends on, like libraries or frameworks.

- `devDependencies` are packages that help you while developing your application, like linters or build tools.

Installing dependencies

`dependencies` and `devDependencies` inside the `package.json` can be installed by running `npm install` (or just `npm i` for short).

> 💡 Do not forget to run `npm install` after cloning a new repository that has a `package.json` file.

When installing, npm creates a `node_modules` folder and a `package-lock.json` file.

- `node_modules` must always be in your `.gitignore` and must not be committed to your repository!
- `package-lock.json` should be commited to your repository.

Semantic Versioning

A semantic version is updated whenever a package changes and a new version is published.

It follows this schema: **`Major.Minor.Patch`** (e.g. `1.2.3`)

- **`Major`** → major version, changes when the public api of a package changes (breaking change)
- **`Minor`** → minor version, changes when new features are added
- **`Patch`** → patch version, changes when bugs are fixed

When defining dependencies in `package.json` npm uses version ranges to define which version of package should be installed. npm always installs the newest version of package that still matches the range description.

- `^` (e.g. `"^10.4.1"`) → Newer minor updates and patches can be installed, but major updates cannot.
  (Here version `10.5.6` would be installed but not `11.0.0`)

- `~` (e.g. `"~10.4.1"`) → Newer patches can be installed, minor and major updates cannot.
  (Here version `10.4.8` would be installed but not `10.5.0`)

- `>` (e.g. `">10.4.1"`) → Any newer version will be installed.
  (Here any version newer than `10.4.1` would be installed)

Version ranges described with `^` are by far the most commmon choice because they are usally safe and won't break your application.

Linters

Linters are tools which analyze your code and show syntax errors, oversights like undeclared variables, bugs and stylistic errors. Some important linters are Prettier (Code formatter), HTMLHint (HTML), and ESLint (JavaScript).

To run these linters, we can define a script inside of the package.json as in the following example:

```
    "scripts": {
        "test": "npm run htmlhint && npm run prettier:check && npm run
eslint",
        "fix": "npm run htmlhint && npm run prettier:write && npm run
eslint",
        "htmlhint": "npx htmlhint \"**/*.html\"",
        "prettier:check": "npx prettier --check .",
        "prettier:write": "npx prettier --write .",
        "eslint": "npx eslint \"**/*.js\""
    }
```

Prettier

Prettier makes sure that your code / the code of your team is formatted in the exact same way. There are two important ways to use it:

- `npx prettier --check .` (checks for stylistic errors)
- `npx prettier --write .` (fixes stylistic errors)

The command `npx` (x = execute) starts prettier; the dot `.` at the end tells prettier to go through all files. You can also choose to check only specific files or folders.

The flags `--check` and `--write` decide whether to only check for errors or immediately fixing them.

We can also use the scripts called "prettier:check" and "prettier:write" in the package.json above via `npm run prettier:check` or `npm run prettier:write`. It will do the exact same thing as `npx prettier [...]`.

HTMLHint

HTMLHint analyzes your HTML. You can use

- `npx htmlhint index.html` (analyzes the index.html file) or the script
- `npm run htmlhint`.

Note that, according to the above package.json, the script will run `npx htmlhint \"**/*.html\"` and thus analyze all files ending with `.html`.

ESLint

ESLint analyzes your JavaScript and highlights errors. You can use

- `npx eslint index.js` (analyzes the index.js file) or the script
- `npm run eslint`.

Note that, according to the above package.json, the script will run `npx eslint \"**/*.js\"` and thus analyze all files ending with `.js`.

Combining Scripts

We can write a script using several other scripts and thus running all linters at once. See the above mentioned scripts `npm run test` and `npm run fix` which will run htmlhint, prettier and eslint.

The `&&` means that the script will run one after another. The next script only is executed if the one before found no issues.

Setup Files for Linters

All linters come with a built-in ruleset, but we can configure these rules. We do this with files at the root of our project called `.eslintrc.js`, `.htmlhintrc.json`, or `.prettierrc`. You can recognize them by the "rc", but the file ending might differ.

We can also say which files the linter will ignore. This is done inside of `.eslintignore` or `.prettierignore`.

Error messages are your friends

> 💡 Error messages are your friends, that kindly point you towards errors. Learning to correctly read error messages is one of the most important skills you'll pick up as a developer.
>
> If you come across an error message, take your time to fully understand what it is saying. Then navigate to the place in code where it found the issue. This way you know exactly what to fix and where.

Resources

About npm:

- npm website
- package.json specification
- npm install documentation
- Semantic Versioning specification

Linters:

- HTMLHint
- Prettier
- ESLint

VSCode Plugins for Linters:

- HTMLHint
- Prettier
- ESLint