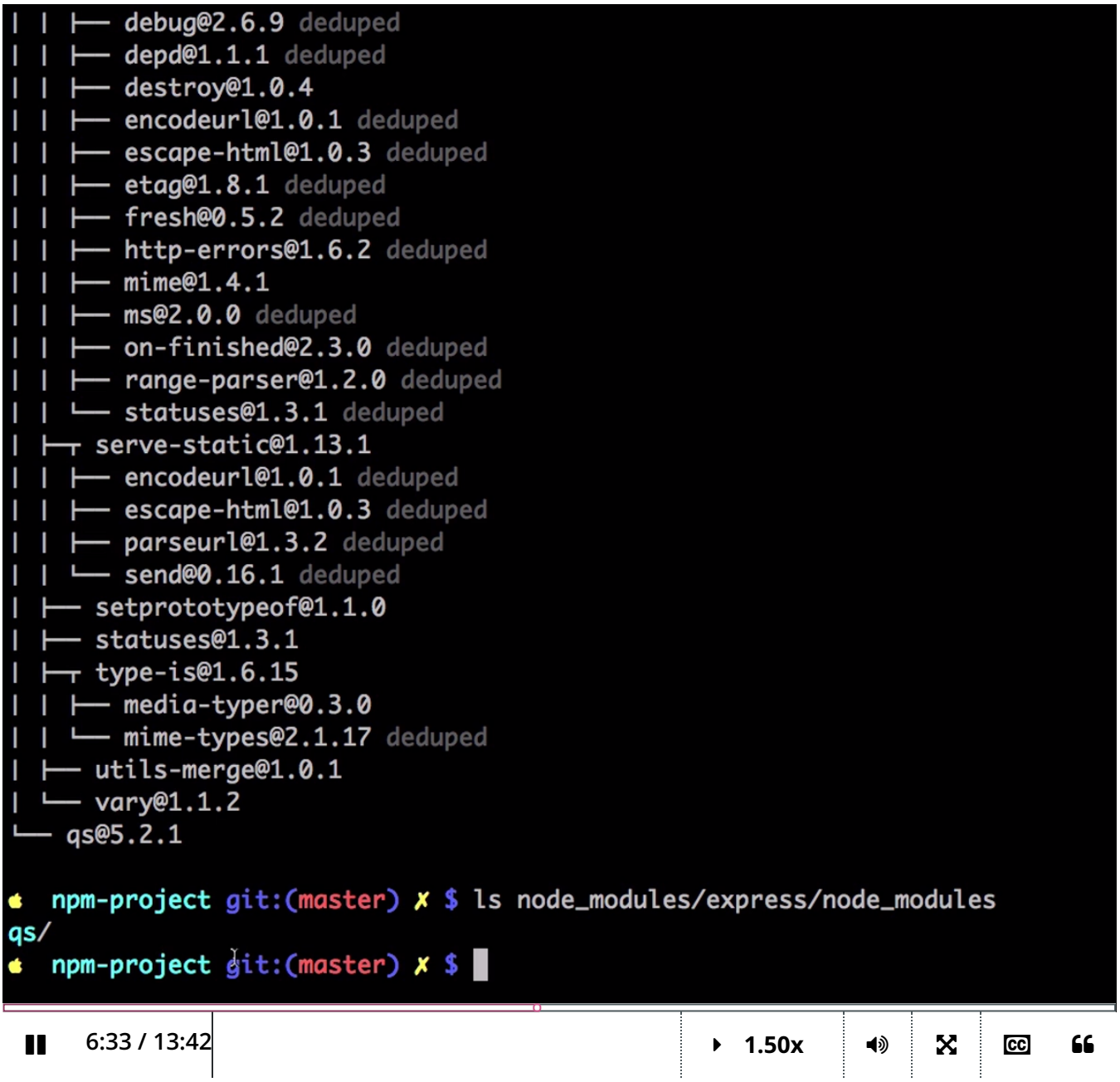

Video

Download video file

Transcripts

Download SubRip (.srt) file

Download Text (.txt) file

---

npm Basics

What is npm?

npm comes with the Node.js platform and allows for seamless Node.js package management. npm (all lower case) consists of three main components:

- Website: https://www.npmjs.com
- Command-line tool: `npm`
- Registries: public and private

The npm command-line tool is the package manager for Node and it is bundled with the Node platform (you don't have to install it separately from Node).

The registries are where the packages (a.k.a. modules) are stored. Developers download packages from the npm registry and publish their packages to the registry.

The website provides a web-based graphical interface to search for modules and find their meta information (how many downloads there were, who is the author or what is the documentation).

Installing Node.js Modules with npm

The way `npm install` works is similar to Git in the way that it traverses the working tree to find a current project. For starters, keep in mind that we need either the `package.json` file or the `node_modules` folder to install modules locally with `npm install name`.

For example, to install a package called `superagent`:

1. In existing project, skip this step but in a new project, create a `package.json` first by running `npm init -y` to initialize the project.
2. In the project root folder (main folder), run `npm install superagent`
3. In a file where you want to use `superagent`, import it: `const superagent = require('superagent')`.

The best thing about npm is that it keeps all the dependencies local, so if module A uses module B v1.3, and module C uses module B v2.0 (with breaking changes compared with v1.3), both A and C will have their own localized copies of different versions of B. This proves to be a more superior strategy than that of Ruby and other platforms that use global installations by default.

The best practice is *not to include* a `node_modules` folder in the Git repository when the project is a module that is supposed to be used in other applications. However, it's recommended *to include* `node_modules` for deployable applications to prevent breakage caused by unfortunate dependency updates.

Introduction to npm

There are two ways to install a module:

1) Locally: most of your projects' dependencies which you import with `require()`, e.g., `express`, `request`, `hapi`. They go into the `node_modules` directory of your local project

```
npm install module-name
npm i module-name
```

2) Globally: command-line tools only (mostly), e.g., `mocha`, `grunt`, `slc`. They go into /usr/local

```
npm install --global module-name
npm i -g module-name
```

The `i` is just an alias to `install`. There's no difference. Use `i` to save time typing.

Some frameworks offer CLI, but most of them belong to the local category. Don't try to install `express` with `-g`!

The `node_modules` folder is where dependencies are stored. It's a local folder which must be in the root (first level sub-folder) of your project. `node_modules` is your friend because it allows for almost no conflicts between different versions of the same dependencies unlike Java, Ruby, Python which prefer global installation over Node's local. Node reduces conflicts because each conflicting dependency will be nested and this will avoid conflicts between different versions of the same dependencies.

Installing Packages

Here are the valid ways in which a Node developer can install an npm module.

Basic installation:

```
npm install express
```

Exact version installation:

```
npm install express@4.2.0
```

Latest version installation, which can be useful when you already have this module but want to upgrade to the latest module:

```
npm install express@latest
```

Explicit save into into package.json dependencies (`--save` or `-S`) or devDependencies (`--save-dev` or `-D`):

```
npm install express --save
npm install express -S
npm install mocha --save-dev
npm install mocha -D
```

In npm version 5, npm will automatically save so `npm i express` will be the same as `npm i -S express`. We recommend using the default behavior of npm version 5 which is to save package information into package.json.

By default, npm will add `^` to the version when you use npm v5 or `--save`. The `^` symbol is dangerous for applications because it means go get the latest version if there's one. It's best to avoid `^`. Using the exact flag will do just that:

```
npm install express --exact
npm install express -E
```

You can combine flags and install more than one dependency in one command:

```
npm i react react-dom babel babel-core -ED
```

Lastly, when you will need to install a tool like npm itself (or upgrade it) you will use the global installation:

```
npm i -g npm@latest
npm install grunt --global
```

If you see an error about permissions, you'll need to change the system folder which npm uses to the appropriate permissions or just use root access with sudo:

```
sudo npm install grunt -g
```

Semantic versioning consists of using three digits which have certain meaning. For example, in semver 4.2.0, 4 is major, 2 is minor and 0 is patch. Major is for major releases which most often break existing code. Minor are for small releases which can break some code but most often are okay. Patch is for small fixes which should not change the main interface and *should* not break your applications.

The key word here is *should* because semantic versioning is not enforced. It's purely a human convention and not all modules and projects in the FOSS follow it.

Listing and Removing Modules

To list what modules are installed, run `npm ls` from your root project location (where you have package.json and node_modules). It will display a tree of dependencies of this current project.

To list all globally installed modules, run `npm ls -g`.

To remove an npm module use the `rm` command:

```
npm rm mysql
```

To remove a global module, apply the global flag:

```
npm rm mysql -g
```