COURSEWARE

- Professional Skills
- Agile Fundamentals
- Jira
- Git
- Databases Introduction
- Java Beginner
- Maven
- Testing (Foundation)
- Java Intermediate
- HTML
- CSS
- Javascript
- Spring Boot
- Selenium
- Sonarqube
- Advanced Testing (Theory)
- Cucumber
- MongoDB
- Express
- NodeJS
  - Intro
  - Modules
  - NPM
  - REST applications with Node
- React
- Express-Testing
- Networking
- Security
- Cloud Fundamentals
- AWS Foundations

NPM

Contents

- Overview
- Creating a Node project
- Configuring a Node project
  - Managing dependencies
  - npm run-script
  - npx
- Tutorial
- Exercises

Overview

NPM is the Node Package Manager.
A 'package' in software development is a single application/library/etc that has been 'built' (made ready for production - usually involves converting source code into a more optimised format) and 'packaged' together with its dependencies (the *other* packages the app needs for it to work).
Npm helps us manage our packages by providing a massive online repository for every node app under the sun - available at www.npmjs.com

From npmjs.com we can see a ton of useful information about any packages we may want to use, such as: documentation (README.md), current version number, version history, dependencies, dependents, licensing info, file size and much more.

Creating a Node project

Before we can package up our app we need to initialise it.
This can be done using

```
npm init
```

This will turn the current folder into a new node project, but first it will ask us for a load of information about the package like the name, version, description, entry point, git repo, author, license, etc.

If you're happy to use all the default values you can skip the initial set-up with:

```
npm init -y
```

Configuring a Node project

Most of the configuration for a Node project comes from the package.json.
On top of storing all the info we discussed previously it also keeps track of a project's dependencies.

Managing dependencies

Dependencies can be added using

```
npm install <PACKAGE_NAME>
```

Dependencies added in this way will be added into the 'dependencies' array in the package.json file like so:

AWS Intermediate
Linux
DevOps
Jenkins Introduction
Jenkins Pipeline
Markdown
IDE Cheatsheet

```
  "name": "demo",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "lodash": "^4.17.15"
  }
}
```

The installed dependencies will be pulled into the **node_modules** folder in your package, it is **very important** to not push this folder into your VCS as it can get quite large and will slow down any git operations significantly.
Similar to the package.json is the package-lock.json file; this file keeps a record of **every** dependency your package needs **including** the *dependencies* of your *dependencies*.
For example, if we were to install chalk.js(a library for styling console.output) the package.json would look like this:

▶ package.json

and the package-lock.json would look like *this*:

▶ package-lock.json

Removing dependencies can be done simply by using the **remove** command

```
npm remove lodash
```

npm run-script

When managing a package there are certain actions which we may want to perform a lot.
Npm makes this convenient by allowing us to save **scripts** into our package.json.

For example, a simple greeting:

```
{
  "name": "demo",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "hello": "echo 'Hello there'"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "chalk": "^2.3.0"
  }
}
```

The 'hello' script specified in this **package-json** simply writes 'Hello there' to the terminal and can be executed using

```
npm run-script hello
```

or just

```
npm run hello
```

Npm can run some scripts by default, allowing us to skip the **run** command. Some of these commands are:

```
npm restart
npm start
npm stop
npm test
```

npx

npx is a package that allows us to execute package binaries from the command line.
Typically this is used to run commands available in packages from npm *without* having to install them.
One simple example is the **cowsay** library - a very simple library for generating ASCII cows saying things:

```
npx cowsay moo
```

This command will execute the **cowsay** binary, passing in 'moo' as an argument.
Node allows us to execute binaries from any by installing packages *globally* using the **-g** flag.

```
npm install cowsay -g
```

Tutorial

There is no tutorial for this module.

Exercises

1. Create a new folder and initialise it as an npm package - call the package

npm-demo and fill in any other fields you want to.

▶ Click here for solution

2. Install the cowsay library to this new project.

▶ Click here for solution

3. Add a script to your package.json called **mycow** - set your own cow type and message (docs can be found at npmjs.com - search for cowsay)

▶ Click here for solution

4. Try out your new script.

▶ Click here for solution

5. When you're happy your script works remove the cowsay package from your project.

▶ Click here for solution