Thinksheet

Node and NPM

Node v8.x

A platform for building applications written JavaScript. Node runs on top of Chrome's JavaScript engine called "V8".

Interactive REPL and Running Script Files

Interactive REPL (Read-Eval-Print Loop) - Type: `^ + C` **to exit**

```
$ node
```

Run server.js

```
$ node server.js
```

Environment Variables

Export env var and run server.js (Mac Terminal and Win Git Bash)

```
$ export DATABASE='YOUR-CONNECTION-STRING'
$ node server.js
```

Node Inspector and Debugging

Run in debug mode

```
$ node --inspect server.js
```

Or enter URL in Chrome "Open dedicated DevTools for Node"

```
chrome://inspect
```

NPM's `package.json` file

Provides documentation for your application, including name, author, description, repo and license. Specifies dependencies using semver.

Initialize `package.json` (`--yes` and accept defaults)

```
$ npm init --yes
```

Dependencies Types:

- **"dependencies"**: required in production
- **"devDependencies"**: only needed for development and testing

Semantic Versioning

Semver (semantic versioning) - 1.2.3 = `major.minor.patch`

- `major` - Changes which break backwards compatibility
- `minor` - New features which don't break existing features
- `patch` - Bug fixes and other minor changes

Dependencies Version Ranges:

Range Types	Example	Allows	D/N Allow
Tilde Ranges: ~1.2.3 ~1.2 • Allows patch-level changes if a minor version is specified; • Allows minor-level changes if not.	~1.2.3	1.2.5 1.2.9	1.2.1 1.3.0
	~1.2	1.2.1 1.2.9	1.1.0 1.3.0
Caret Ranges Allows patch and minor updates	^1.2.3	1.2.5 1.2.9 1.3.0	1.2.1 2.1.1
X-Ranges: X is a wildcard	1.2.x	1.2.0 1.2.5 1.2.9	1.3.0

NPM commands v5.0.x

npm package manager for the Node platform. Installs packages so that node can find them, and manages dependency and conflicts.

Install - global

npm install <package-name> --global

Global mode (--global or -g) installs packages and bins globally. Typically:

- packages in `/usr/lib/node_modules`
- and bins in `/usr/bin`

Install latest eslint globally (e.g. '/usr/local/bin')

```
$ npm install eslint --global
```

Install - local

npm install <name> [args] installs in to the current project and saves package(s) into dependencies property of package.json.

- packages in `./node_modules`
- and bins in `./node_modules/.bin`

`--save` to dependencies (now default behavior in v8.x)
`--save-dev` to save package to dev-dependencies

Install latest express and save to dependencies

```
$ npm install express
```

Install several packages and save to dev-dependencies

```
$ npm install mocha chai chai-http --save-dev
```

Install a specific version

```
$ npm install package@1.2.3 --save
```

NPM scripts and NPM run-script

Execute script defined in `script` object in `package.json` file
`npm [start|stop|test]` runs script associated with command.
`npm run <name>` runs arbitrary command from `scripts` object

Define scripts in the `script` object *package.json*

```
"scripts" : {
  "start" : "node server.js", // this is the default
  "dev" : "nodemon server.js",
  "test" : "mocha"
}
```

Run start command (defaults to node server.js)

```
$ npm start
```

Run test command like mocha

```
$ npm test
```

Run arbitrary command like dev

```
$ npm run dev
```

CommonJS Modules

CommonJS Modules - a format to cleanly expose (export) code in one file (module) so it can be loaded (required) as a dependency in another file.

Import Module

`require('module')` - loads a module where `'module'` can be:

- **Core Node** module like `'http'`
- **Package** (3rd party module) in `'node_modules'`
- **File Module** path to file, starts with `'/'`, `'./'` or `'../'`)
  - '.js' file extension is optional
- **Folder Module** path to a directory which contains:
  - `'package.json'`, `'index.js'` or `'index.node'`

Load module (aka npm package) *parent.js*

```
const myMod = require( 'package' );
```

Looks for `'package'` in:

- `core node module`
- `../node_modules/`
- `../../node_modules/ (up to root)`

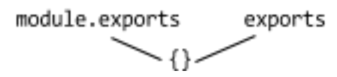
Load custom `module.js` into parent *parent.js*

```
const myMod = require( './<path>/module' );
myMod.getAnswer();
myMod.getAdvice();
```

Note: you do not need to provide the `.js` extension

Export a module

The `module.exports` object exposes properties from the current scope. And `exports` is a "shorthand" which points to `module.exports`



Export an object *module.js*

```
module.exports = {
  getAnswer: () => 42,
  getAdvice: () => "Don't Panic"
};
```

Export individual properties *module.js*

```
module.exports.getAnswer = () => 42;
module.exports.getAdvice = () => "Don't Panic";
```

Or use `exports` shorthand *module.js*

```
exports.getAnswer = () => 42;
exports.getAdvice = () => "Don't Panic";
```

Warning: Do not redefine `exports`. It will not work as intended.

```
exports = {
  getAnswer: () => 42,
  getAdvice: () => "Don't Panic"
};
```