MENTORMATE™
DevCamp

Debugging, Error Handling & Testing in Node.js

Yulia Tenincheva

Senior Cloud Engineer, MentorMate

Today's Agenda

- NPM & Yarn
- Debugging Node.js
- Testing Node.js Applications
- Build a HTTP Server

NPM

NPM

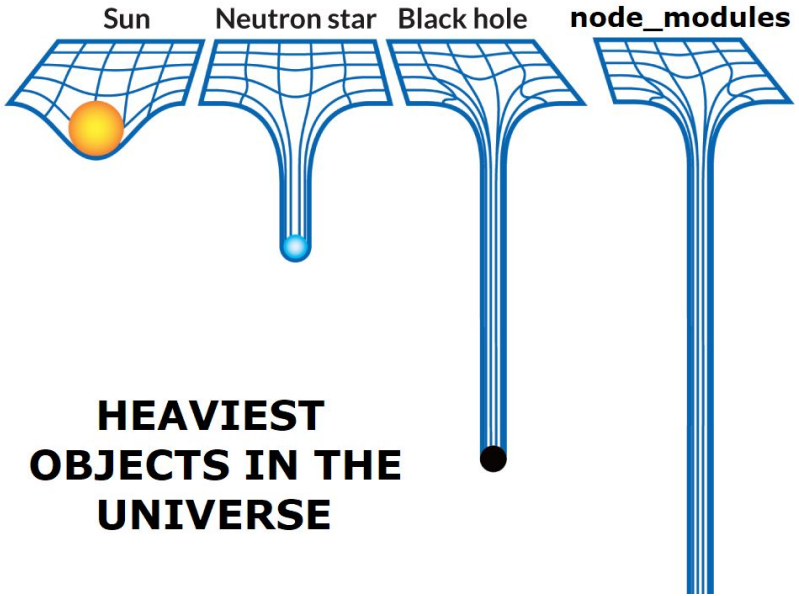
- What is NPM?
- Install, remove, update & list
- Local & global packages
- Dev/Prod dependencies

```
npm install (with no args, in package dir)
npm install [<@scope>/]<name>
npm install [<@scope>/]<name>@<tag>
npm install [<@scope>/]<name>@<version>
npm install [<@scope>/]<name>@<version range>
npm install <git-host>:<git-user>/<repo-name>
npm install <git repo url>
npm install <tarball file>
npm install <tarball url>
npm install <folder>

aliases: npm i, npm add
```

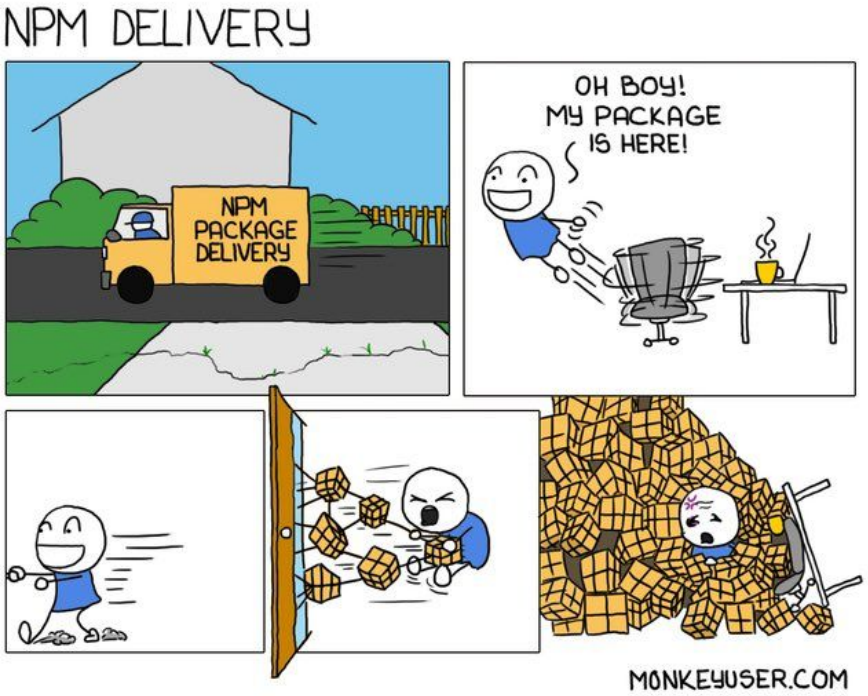
NPM

- Commands & shortcuts
- Security
- NPM Scripts
- Package-lock vs NPM Shrinkwrap
- npx tool



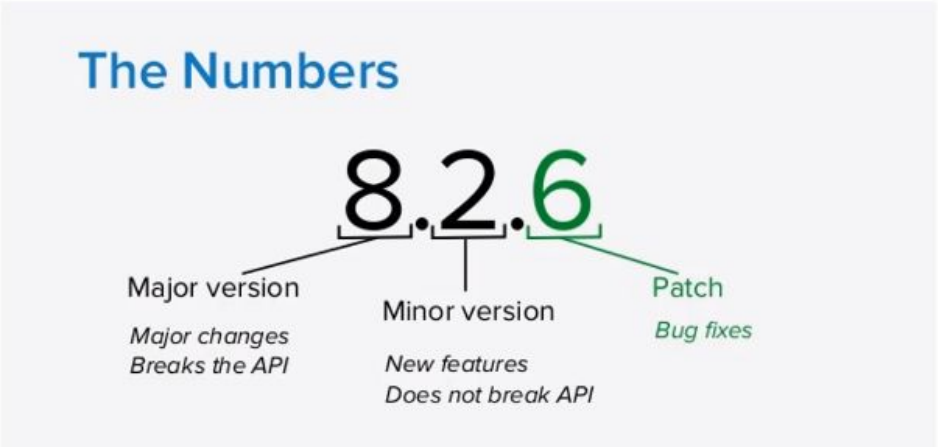
package.json

- Manifest file with app info and config
- List dependencies (nam & version)
- Specify if versions should be updated
- Create NPM Scripts



Semantic Versioning

1. **MAJOR** version when you make incompatible API changes
2. **MINOR** version when you add functionality in a backwards compatible manner, and
3. **PATCH** version when you make backwards compatible bug fixes.

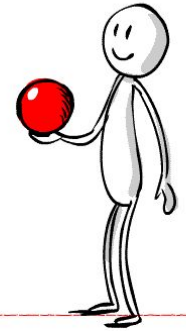


[Semver Specification]

Error Handling

Error Handling

- `try, catch, finally, throw`
- Errors vs Exceptions
  - Error is what you throw
  - Exception is what happens when you throw something



```
try {
  setTimeout(function () {
    throw new Error('boom!');
  }, 0);
} catch (e) {
  console.log('Aha! I caught the error!');
}
```

```
$ node try-settimeout.js
/Users/lewis/dev/node-error-talk/try-settimeout.js:3
    throw new Error('boom!');
    ^
Error: boom!
    at Timeout._onTimeout (/Users/lewis/dev/node-error-talk/try-settimeout.js:3:11)
    at ontimeout (timers.js:365:14)
    at tryOnTimeout (timers.js:237:5)
    at Timer.listOnTimeout (timers.js:207:5)
```

```
throw new Error('something bad happened!');
throw 'something bad happened' // avoid this
```

MENTORMATE™ DevCamp

Error Handling

- Error Class
  - <EvalError>, <SyntaxError>, <RangeError>, <ReferenceError>, <TypeError>, <URIError>, User-specified errors, AssertionError
- *Operational Errors* vs *Programming Errors*
  - Expect to handle operational errors (e.g *network timeouts, database is down, disk got full, unexpected missing user inputs*)
  - Programming errors to avoid (nonrecoverable): silently ignoring, classic JavaScript Errors,
  - Invoking a callback twice

```
throw new Error('Ran out of coffee')
```

or

```
class NotEnoughCoffeeError extends Error {
  //...
}
throw new NotEnoughCoffeeError()
```

Error Mechanisms in Node

- Try/Catch - throw and try/catch
- Callbacks - `err` first argument is and if (any)
- Promises - reject(err) and .catch()
- EventEmitters - error events and .on('error');
- Express - next(err) and error-handling middleware

Best Practices

- Always know when your errors happen, don't ignore
- Use a process manager so shutting down is no big deal
- Handle what you can, avoid what you can't
- Lint your code!
- Avoid global catch-all

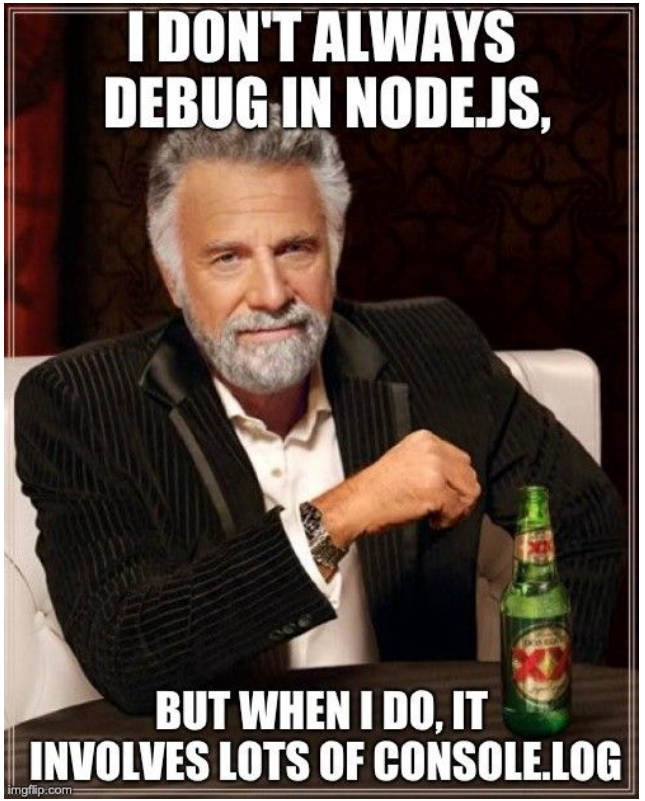
```
process.on('uncaughtException', function (err) {
  console.log('Uncaught exception! Oh no!');
  console.error(err);
  process.exit(1);
});
```

Debugging

Debugging

- Importance of **Logging** & **Monitoring**
- Logging **libraries** (Winston, Morgan)
- Your IDE/editor built-in debugger
- Chrome Debugger

```
node --inspect-brk file.js | chrome://inspect
```



Testing

Testing for Quality

- **Static Code Analysis**

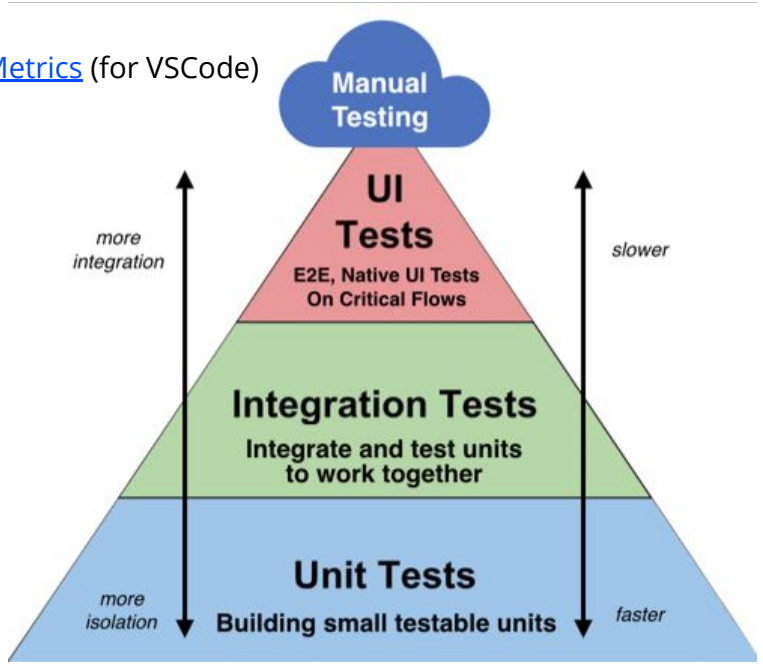
  SonarQube, formatters and linters like Prettier, ESLint, CodeMetrics (for VSCode)

- Testing **Pyramid**

- Test **Coverage** (nyc)

- **TDD**, **BDD** vs **ATDD**

- Testing for **Security** (snyk)



Unit Testing in JavaScript - Youtube Playlist

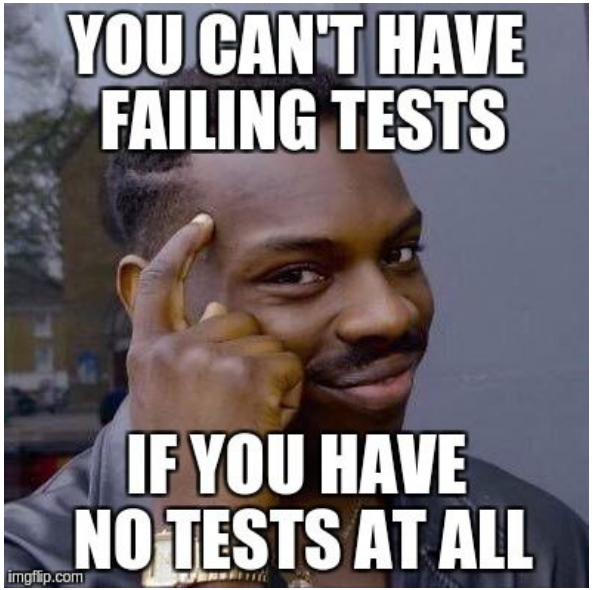
Testing Tools

Mocha - Getting Started, [Documentation]

```
var assert = require('assert');
describe('Array', function () {
  describe('#indexOf()', function () {
    it('should return -1 when no value', function () {
      assert.equal([1, 2, 3].indexOf(4), -1);
    });
  });
});
```

*update package.json

```
it('should save without error', function (done) {
  var user = new User('Luna');
  user.save(done);
});
```



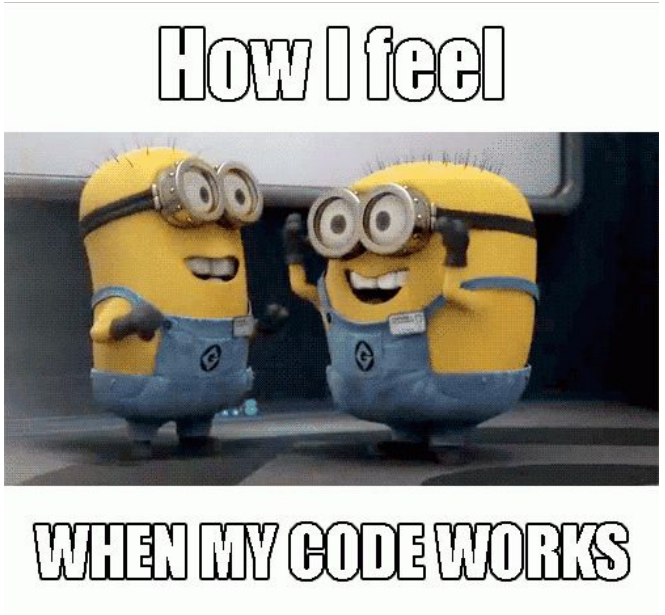
Testing Tools

Chai - Getting Started, [Documentation]

```
  var chai = require('chai'),
      expect = chai.expect,
      should = chai.should();
```

```
var should = require('chai').should();

db.get(1234, function (err, doc) {
  should.not.exist(err);
  should.exist(doc);
  doc.should.be.an('object');
});
```



Testing Tools

- **Supertest** - Getting Started, Documentation

```
const request = require('supertest');
const http = require('http');

const requestListener = function (req, res) {
    res.writeHead(200, {'Content-Type': 'application/json'});
    res.write(JSON.stringify({id: 1, username: "Pesho"}));
    res.end();
};

const server = http.createServer(requestListener);
```

```
describe('Up and Running', function() {
    it('responds with json', function(done) {
        request(server)
            .get('/')
            .expect('Content-Type', /json/)
            .expect(200)
            .end(function(err, res) {
                if (err) return done(err);
                    done();
            });
    });
});
```

Let’s build something

Homework

1. CATtering **API**
2. CATtering **Client** (CLI)
3. CATtering API **Testing**

[Homework Assignment]