

# Crear una librería

## Librerías en node

- Suelen ser pequeñas
- Es un buen ejemplo de ciclo de desarrollo en node.js
- Ayuda a tener claro el concepto de paquetes de node

## Microlibrerías

- Ventajas
  - Poco código, se entiende y modifica con facilidad
  - Reusable
  - Fácil hacer tests
- Desventajas
  - Tienes que gestionar muchas dependencias
  - Control de versiones de todas ellas

## Funcionalidad librería

- Obtiene una marca de cerveza y sus características
- Obtiene una o varias marcas de cerveza al azar.

## Control de versiones

- Utilizaremos git como control de versiones
- Utilizaremos github como servidor git en la nube para almacenar nuestro repositorio:
  - Haz login con tu usuario (o crea un usuario nuevo)
  - Crea un nuevo repositorio en GitHub (lo llamaré *cervezas*)
  - Sigue las indicaciones de GitHub para crear el repositorio en local y asociarlo al repositorio remoto (GitHub)

## Instalación de node

- Lo más sencillo es [instalar mediante el gestor de paquetes](#)

```
curl -sL <https://deb.nodesource.com/setup_5.x> | sudo -E bash -  
sudo apt-get install -y nodejs
```

- Comprobamos que esté correctamente instalado

```
node -v  
npm -v
```

## npm

- Es el gestor de paquetes de node
- **Debemos crear un usuario** en <https://www.npmjs.com/>
- Podemos buscar los paquetes que nos interese instalar
- Podemos publicar nuestra librería :-)

## Configuración de npm

- Cuando creamos un nuevo proyecto nos interesa que genere automáticamente datos como nuestro nombre o email
- Ver [documentación para su configuración](#) o mediante consola (`npm --help`) :
- Mediante `npm config --help` vemos los comandos de configuración
- Mediante `npm config ls -l` vemos los parámetros de configuración

```
npm set init-author-name pepe  
npm set init-author-email pepe@pepe.com  
npm set init-author-url <http://pepe.com>  
npm set init-license MIT  
npm adduser
```

- Los cambios se guardan en el fichero `$HOME/.npmrc`
- `npm adduser` genera un authtoken = login automático al publicar en el registro de npm

## Versiones en node

- Se utiliza [Semantic Versioning](#)

```
npm set save-exact true
```

- Las versiones tienen el formato MAJOR.MINOR.PATCH
- Cambios de versión:
  - MAJOR: Cambios en compatibilidad de API,
  - MINOR: Se añade funcionalidad. Se mantiene la compatibilidad.
  - PATCH: Se solucionan bug. Se mantiene compatibilidad.
- ¡Puede obligarnos a cambiar el MAJOR muy a menudo!

## Creamos el proyecto

- Dentro del directorio cervezas:

```
npm init
```

- El *entry-point* lo pondremos en *src/index.js*, así separaremos nuestro código fuente de los tests.
- El resto de parámetros con sus valores por defecto
- ¡Ya tenemos nuestro **package.json** creado!

## Listar todas las cervezas:

- Editamos nuestro fichero *src/index.js*

```
var cervezas = require('./cervezas.json')
module.exports = {
  todas: cervezas
}
```

- Abrimos una consola y comprobamos que funcione nuestra librería:

```
node
> var cervezas = require('./index.js')
undefined
> cervezas.todas
```

# Ahora queremos obtener una cerveza al azar:

- Instalamos el paquete [uniqueRandomArray](#)

```
npm i -S unique-random-array
```

- Configuramos nuestro fuente:

```
cervezas = require('./cervezas.json')
var uniqueRandomArray = require('unique-random-array')
module.exports = {
  todas: cervezas,
  alazar: uniqueRandomArray(cervezas)
}
```

- Comprobamos que funcione. Ojo, ¡alazar es una función!

## Subimos la librería a github

- Necesitamos crear un **.gitignore** para la carpeta no sincronizar **node\_modules**
- Los comandos que habrá que hacer luego son:

```
git status
git add -A
git status
git commit -m "versión inicial"
```

- Ojo que haya cogido los cambios del .gitignore para hacer el push

```
git push
```

- Comprobamos ahora en github que esté todo correcto.

## Publicamos en npm

```
npm publish
```

- Podemos comprobar la información que tiene npm de cualquier paquete mediante

```
npm info <nombre paquete>
```

# Probamos nuestra librería

- Creamos un nuevo proyecto e instalamos nuestra librería
- Creamos un index para utilizarla:

```
var cervezas = require('cervezas')
console.log(cervezas.alazar())
console.log(cervezas.todas)
```

- Ejecutamos nuestro fichero:

```
node index.js
```

# Versiones en GitHub

- Nuestro paquete tiene la versión 1.0.0 en npm
- Nuestro paquete no tiene versión en GitHub, lo haremos mediante el uso de etiquetas:

```
git tag v1.0.0
git push --tags
```

- Comprobamos ahora que aparece en la opción Releases y que la podemos modificar.
- También aparece en el botón de seleccionar branch, pulsando luego en la pestaña de tags.

# Modificar librería

- Queremos mostrar las cervezas ordenadas por nombre
- Utilizaremos la **librería lodash** (navaja suiza del js) para ello:

```
var cervezas = require('./cervezas.json');
var uniqueRandomArray = require('unique-random-array');
var _ = require('lodash');
```

```
module.exports = {
  todas: _.sortBy(cervezas, ['nombre']),
  alazar: uniqueRandomArray(cervezas)
}
```

- Ahora tendremos que cambiar la versión a 1.1.0 (semver) en el package.json y publicar el paquete de nuevo
- También añadiremos la tag en GitHub ¿Lo vamos pillando?

## Versiones beta

- Vamos a añadir una cerveza nueva, pero todavía no se está vendiendo.
- Aumentamos nuestra versión a 1.2.0-beta.0 (nueva funcionalidad, pero en beta)
- Al subirlo a npm:

```
npm publish --tag beta
```

- Con npm info podremos ver un listado de nuestras versiones (¡mirá las dist-tags)
- Para instalar la versión beta:

```
npm install <nombre paquete>@beta
```

## Tests

- Utilizaremos **Mocha** y **Chai**
- Las instalaremos como dependencias de desarrollo:

```
npm i -D mocha chai
```

- Añadimos el comando para test en el package.json (-w para que observe):

```
"test": "mocha src/index.test.js -w"
```

- Creamos un fichero src/index.test.js con las pruebas

```
var expect = require('chai').expect;
describe('cervezas', function () {
  it('should work!', function (done) {
    expect(true).to.be.true;
    done();
  });
});
```

```
});  
});
```

- Utiliza los paquetes **Mocha Snippets** y **Chai Completions** de Sublime Text para completar el código
- Ahora prepararemos una estructura de tests algo más elaborada:

```
var expect = require('chai').expect;  
var cervezas = require('./index');  
  
describe('cervezas', function () {  
  describe('todas', function () {  
    it('Debería ser un array de objetos', function (done) {  
      // se comprueba que cumpla la condición de ser array de objetos  
      done();  
    });  
    it('Debería incluir la cerveza Ambar', function (done) {  
      // se comprueba que incluya la cerveza Ambar  
      done();  
    });  
  });  
});  
  
describe('alazar', function () {  
  it('Debería mostrar una cerveza de la lista', function (done) {  
    //  
    done();  
  });  
});  
});
```

- Por último realizamos los tests:

```
var expect = require('chai').expect;  
var cervezas = require('./index');  
var _ = require('lodash')  
  
describe('cervezas', function () {  
  describe('todas', function () {  
    it('Debería ser un array de objetos', function (done) {  
      expect(cervezas.todas).to.satisfy(isArrayOfObjects);  
      function isArrayOfObjects(array){
```

```

        return array.every(function(item){
            return typeof item === 'object';
        });
    }
    done();
});

it('Debería incluir la cerveza Ambar', function (done) {
    expect(cervezas.todas).to.satisfy(contieneAmbar);
    function contieneAmbar (array){
        return _.some(array, { 'nombre': 'ÁMBAR ESPECIAL' });
    }
    done();
});

});

describe('alazar', function () {
    it('Debería mostrar un elemento de la lista de cervezas', function (done) {
        var cerveza = cervezas.alazar();
        expect(cervezas.todas).to.include(cerveza);
        done();
    });
});

});
});

```

# Automatizar tareas

- Cada vez que desarrollamos una versión de nuestra librería:
  - Ejecutar los tests
  - Hay que realizar un commit
  - Hay que realizar un tag del commig
  - Push a GitHub
  - Publicar en npm
  - ...
- Vamos a intentar automatizar todo:
  - **Semantic Release** para la gestión de versiones
  - **Travis** como CI (continuous integration)

# Instalación Semantic Release



- Paso previo (en Ubuntu 14.04, si no fallaba la instalación):

```
sudo apt-get install libgnome-keyring-dev
```

- Instalación y configuración:

```
sudo npm i -g semantic-release-cli  
semantic-release-cli setup
```

- **.travis.yml**: contiene la configuración de Travis
- Cambios en package.json:
  - Incluye un nuevo script (*semantic-release*)
  - Quita la versión
  - Añade la dependencia de desarrollo de Semantic Release

## Versiones del software

- Utilizamos semantic versioning
- Semantic Release se ejecuta a través de Travis CI
- Travis CI se ejecuta al hacer un push (hay que configurarlo desde la web)
- Los commit tienen que seguir las [reglas del equipo de Angular](#)

## Uso de commitizen

- **commitizen** que nos ayudará en la generación de los mensajes de los commit.
- La instalación, siguiendo su [documentación](#):

```
sudo npm install commitizen -g  
commitizen init cz-conventional-changelog --save-dev --save-exact
```

- Habrá que ejecutar **git cz** en vez de **git commit** para que los commits los gestione commitizen

## Cambio de versión

- Vamos a comprobar nuestro entorno añadiendo una funcionalidad
- Si pedimos cervezas.alazar() queremos poder recibir más de una
- Los tests:

```
it('Debería mostrar varias cervezas de la lista', function (done) {
  var misCervezas = cervezas.alazar(3);
  expect(misCervezas).to.have.length(3);
  misCervezas.forEach(function( cerveza){
    expect(cervezas.todas).to.include(cerveza);
  });
  done();
});
```

- Añadimos la funcionalidad en el `src/index.js`: ```` var cervezas = require('./cervezas.json'); var uniqueRandomArray = require('unique-random-array'); var = require('lodash'); var getCerveza = uniqueRandomArray(cervezas) module.exports = { todas: .sortBy(cervezas, ['nombre']), alazar: alazar }`

```
function alazar(unidades) { if (unidades===undefined){ return getCerveza(); } else { var misCervezas = []; for (var i = 0; i<unidades; i++) { misCervezas.push(getCerveza()); } return misCervezas; } }
```

- Hagamos ahora el `git cz & git push` y veamos como funciona todo
- Podríamos añadir un issue y hacer el fix en este commit escribiendo closes `#issue` en el footer del commit message.

### ## Git Hooks

- Son una manera de ejecutar scripts antes de que ocurra alguna acción
- Sería ideal pasar los tests antes de que se hiciera el commit
- Los Git Hooks son locales:
  - Si alguien hace un clone del repositorio, no tiene los GitHooks
  - Instalaremos un paquete de npm para hacer git hooks de forma universal

`npm i -D ghooks`

- Lo configuraremos en el `package.json` en base a la [documentación del paquete](<https://www.npmjs.com/package/ghooks>):

```
"config": { "ghooks": { "pre-commit": "npm test" } }
```

### ## Coverage

- Nos interesa que todo nuestro código se pruebe mediante tests.
- Necesitamos una herramienta que compruebe el código mientras se realizan los tests:

`npm i -D istanbul`

- Modificaremos el script de tests en el package.json:

```
istanbul cover -x *.test.js _mocha -- -R spec src/index.test.js
```

- Istanbul analizará la cobertura de todos los ficheros excepto los de test ejecutando a su vez \_mocha (un wrapper de mocha proporcionado por ellos) con los tests.
- Si ejecutamos ahora `*npm test*` nos ofrecerá un resumen de la cobertura de nuestros tests.
- Por último nos crea una carpeta en el proyecto `*coverage*` donde podemos ver los datos, por ejemplo desde un navegador (fichero `index.html`)
- ¡Ojo, recordar poner la carpeta `coverage` en el `.gitignore`!

```
## Check coverage
```

- Podemos también evitar los commits si no hay un porcentaje de tests óptimo:

```
"pre-commit": "npm test && npm run check-coverage"
```

- Creamos el script `check-coverage` dentro del package.json:

```
"check-coverage": "istanbul check-coverage --statements 100 --branches 100 --functions 100 -lines 100"
```

- Podemos comprobar su ejecución desde el terminal mediante `*npm run check-coverage*` y añadir una función nueva sin tests, para comprobar que el `check-coverage` no termina con éxito.
- Lo podemos añadir también en Travis, de modo que no se haga una nueva release si no hay ciertos estándares (el test si lo hace por defecto):

script:

- `npm run test`
- `npm run check-coverage ```

# Gráficas

- Utilizaremos la herramienta [codecov.io](https://codecov.io):

```
npm i -D codecov.io
```

- Crearemos un script que recoge los datos de istanbul:

```
"report-coverage": "cat ./coverage/lcov.info | codecov"
```

- Lo añadimos en travis de modo que genere un reporte:

```
after success:  
- npm run report-coverage  
- npm run semantic-release
```

- Integrado con github (chrome extension)
- Por último podemos añadir etiquetas de muchos servicios: npm, codecov, travis... una fuente habitual es <http://www.shields.io>

---

Revision #2

Created 19 April 2023 17:58:13 by molombo

Updated 19 April 2023 18:05:31 by molombo