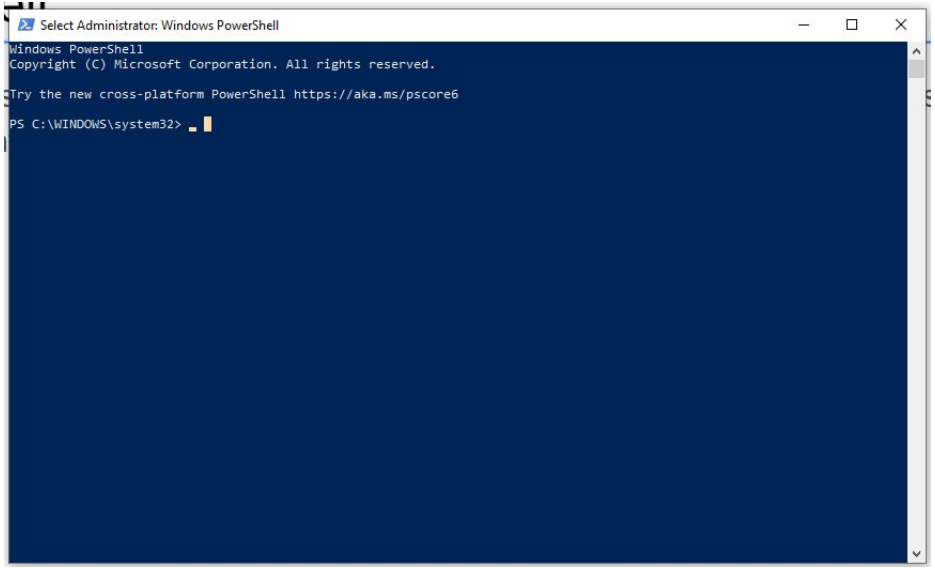
Entorno / Herramientas (Incompleto)

Powershell

Powershell es una **terminal** que nos va a permitir utilizar varias herramientas de desarrollo front



Powershell - Comandos básicos

Cambio de directorio

```
cd ruta/directorio
cd ~/Documentos
cd ~
```

Crear nuevo directorio

```
mkdir ruta/directorio
```

Eliminar directorio o fichero

```
rm ruta/directorio
rm ruta/fichero
```

Powershell - Comandos básicos

Mover directorio o fichero (también **renombrar**)

```
mv antigua/ruta nueva/ruta
```

Copiar directorio o fichero

```
cp ruta/origen ruta/destino
```

Listar directorio

```
ls ruta/directorio
ls
```

Chocolatey

Chocolatey es un **gestor de paquetes** que nos va a permitir instalar el resto de herramientas de forma sencilla

- Instalación

Node

Node es un entorno que permite ejecutar **Javascript** fuera del **navegador**

```
choco install nodejs
```

Npm

npm es un gestor de paquetes que permite instalar **librerías** y aplicaciones de **Javascript**

Viene incluído en la instalación de **node**

```
choco install nodejs
```

Npm

npm permite instalar librerías JS fácilmente desde la terminal

```
npm install jquery --save
```

```
npm uninstall jquery --save
```

```
npm update jquery --save
```

- Las librerías se guardan en la carpeta **node_modules**
- La lista de dependencias se guarda en **package.json**

Git

Git

Herramienta de **control de versiones**

- Permite guardar un historial de los cambios en el código
- Facilita la gestión de las versiones de una aplicación
- Facilita la publicación de **repositorios** de código
- Facilita la **edición simultánea** de ficheros entre varias personas

Git

Herramienta de **control de versiones**

Instalación

```
choco install git
```

Referencia comandos

https://github.github.com/training-kit/downloads/es_ES/github-git-cheat-sheet/

Git

Crear un repositorio

```
cd ruta/proyecto
git init
```

`git init` Crea directorio `.git` que incluye la información del repositorio

Git

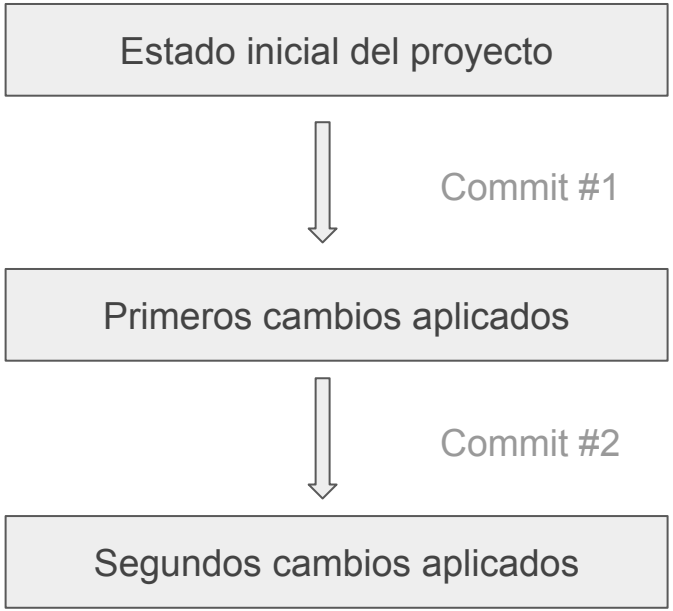
Estado del repositorio

`git status`

```
PS C:\Users\anon\workspace\curso> git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        package.json

nothing added to commit but untracked files present (use "git add" to track)
```

Commit

Un commit empaqueta un **conjunto de cambios** en el proyecto



Commit

Un commit contiene información de este estilo:

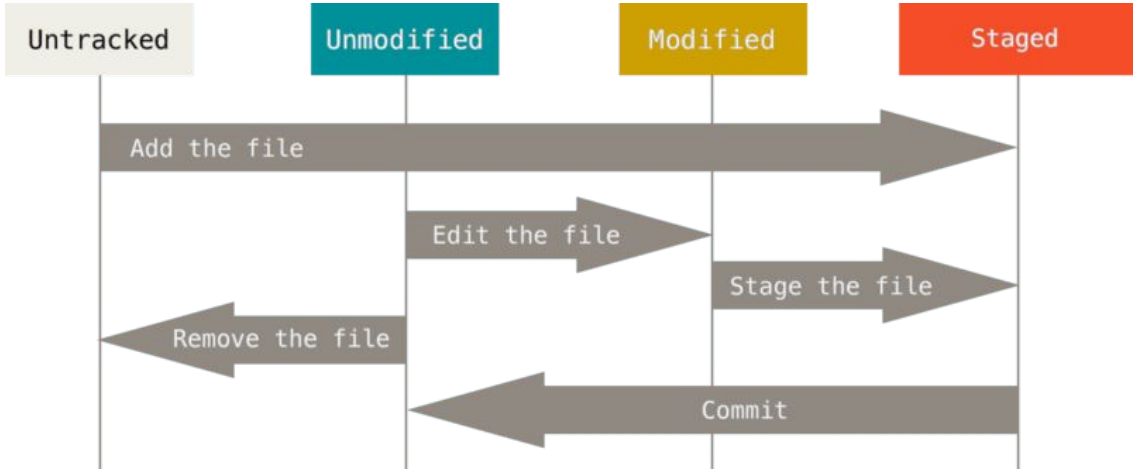
- El fichero **index.html** ha cambiado en las líneas 23 y 24
- Se ha creado un nuevo fichero **main.js**
- Se ha eliminado el fichero **test.txt**

Deshaciendo estos cambios podemos volver atrás en el tiempo

Staging

Para **incluir** los cambios de un fichero en el commit utilizamos

```
git add ruta/fichero
```



Commit

Una vez incluidos los cambios podemos **commitearlos**

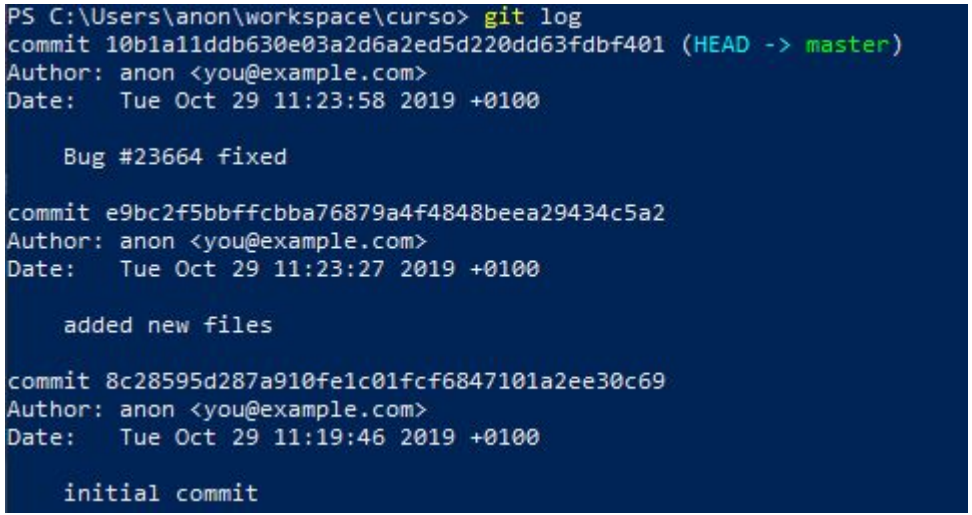
```
git add ...
git commit -m ”added new files”
```

```
git add ...
git commit -m ”Bug #23664 fixed”
```

Git log

Podemos acceder a la **lista de commits** usando

`git log`



Push

Podemos publicar los cambios a un **repositorio remoto** (Eg: Github, Bitbucket)

```
git push origin rama
```

Ejemplo:

```
git push origin master
```

Pull

Podemos descargar los cambios de un **repositorio remoto**

```
git pull origin rama
```

Ejemplo:

```
git pull origin master
```

(Comprueba que estás en la rama en la quieres aplicar los cambios)

Demo repositorio Bitbucket

Ejercicio git

- Crea una cuenta en Github
- Crea un nuevo repositorio
- Realiza 3 cambios y comitealos
- Pushea los cambios al repositorio

Clone

Utilizamos `git clone https://github.com/usuario/repositorio.git ruta/destino`

Para descargar (clonar) **repositorios remotos** en nuestro ordenador

Ramas

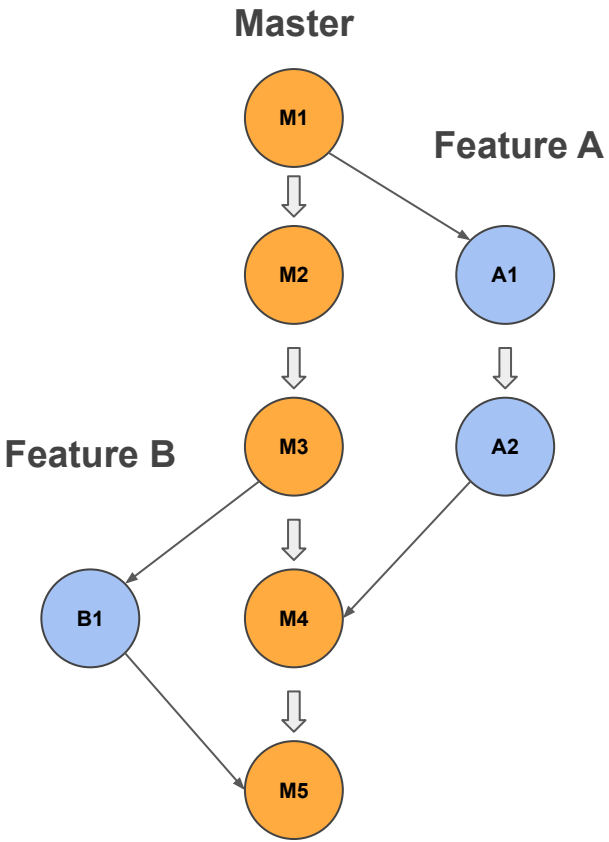
Ramas

Habitualmente varios desarrolladores trabajan en diferentes **features** de un mismo proyecto

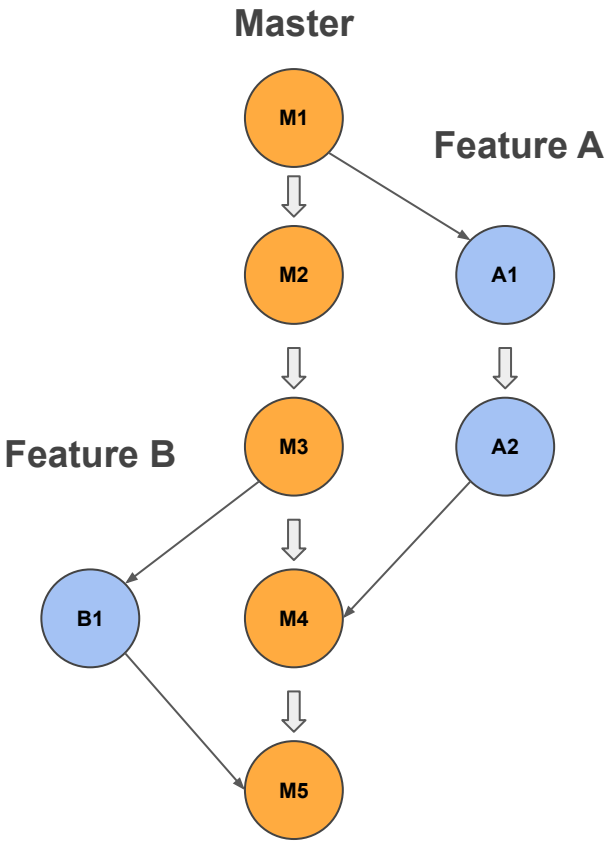
Si trabajan sobre los mismos ficheros puede haber **conflictos**

Las **ramas** nos permiten trabajar en varias features **de forma aislada** y luego ponerlas en común

Ramas



HEAD



Checkout

Mover HEAD a otra rama

```
git checkout rama
```

Crear una nueva rama a partir de HEAD

```
git checkout -B rama
```

Merge

Mergear rama

```
git checkout ramaDestino
git merge ramaOrigen
```

Si commits de ambas ramas modifican las mismas líneas del mismo archivo surge un **conflicto** que hay que resolver

Git nos muestra **en qué líneas** está el conflicto y cuáles son **las dos versiones** para que escojamos la definitiva

Fetch

Fetch actua como pull, pero no mergea la rama automáticamente.

```
git fetch origin rama
```

Ejemplos:

```
git fetch origin
git fetch origin master
```

Las ramas se descargan en origin/rama y se pueden mergear manualmente si es necesario

Ejercicio git II

- Crea una nueva rama en tu repositorio
- Comitea un nuevo cambio en la rama
- Pushea la rama a Github
- Mergea el cambio de la nueva rama en tu rama master
- Pushea la rama master

Ejercicio git III

- Modifica la misma zona de un fichero en dos ramas diferentes
- Mergea ambas ramas para crear un conflicto
- Soluciona el conflicto y pushea los cambios

Git avanzado

Cherry picking

Cherry picking nos permite mergear el commit específico de una rama, en vez de todos los commits que se han hecho en esa rama

```
git checkout rama/destino
```

```
git cherry-pick commit-hash
```

Ejemplos:

```
git checkout master
git cherry-pick fsd6s87sg678j587sfj58s76a7sd8g5a8sdh4
```

Las ramas se descargan en origin/rama y se pueden mergear manualmente si es necesario

amend

La opción amend nos permite modificar el **último commit**

Ejemplo: añadir un fichero que nos hemos dejado

```
git add file
git commit --amend --no-edit
```

Ejemplo: arreglar un error en el mensaje de commit

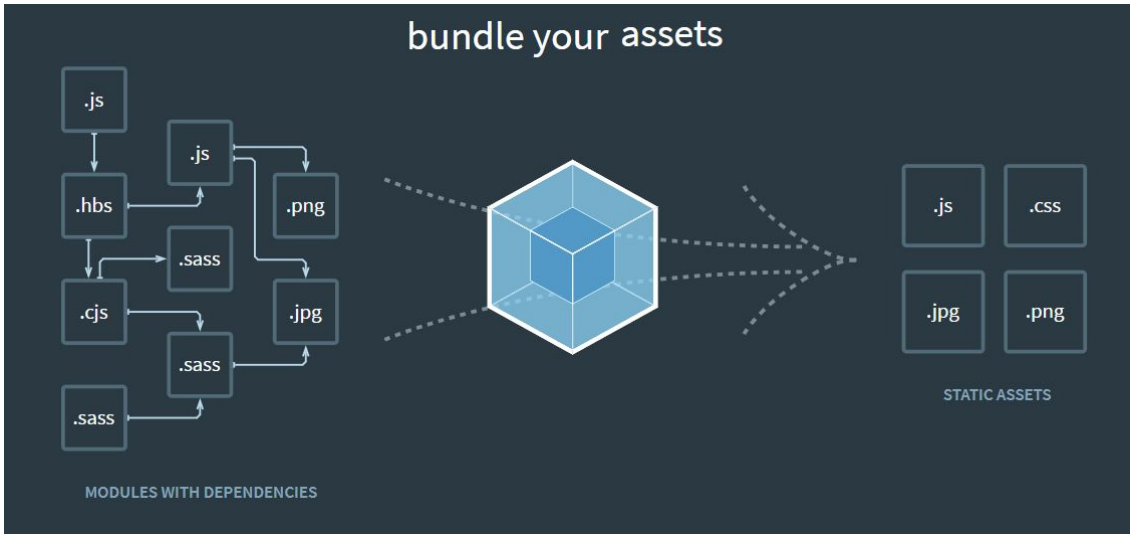
```
git commit --amend -m “new error message”
```

Rebase (pendiente)

Webpack

Webpack

Webpack es un **module bundler** de Javascript



Webpack

Además de **empaquetar,** webpack puede aplicar **transformaciones**

- Minificar y *uglificar* código
- Aplicar **preprocesadores** (Sass)
- **Transpilar** a versiones antiguas (Babel)
- Module **import/export** syntax

Webpack

npm permite instalar webpack fácilmente

```
npm install webpack --save-dev
```

Las instrucciones que determinan

- Qué va a empaquetar webpack
- Qué transformaciones va a aplicar

Se configuran en el fichero **webpack.config.js**

Webpack

En este repositorio se encuentra una configuración básica

- Babel
- Polyfills
- Sass
- Import/Export

Webpack

Una vez tenemos la configuración adecuada podemos empaquetar utilizando

```
npm run build
```

También podemos utilizar el script **watch** para que aplique build automáticamente cada vez que modificamos un fichero

```
npm run watch
```

Explicación configuración

Ejercicio webpack

- Descarga la plantilla de webpack del repositorio anterior
- Instala las dependencias con npm
- Cambia los contenidos de el fichero src/js/main.js para imprimir “lorem ipsum” por pantalla

Require

Require

Javascript permite importar otros ficheros mediante **require**

```
// main.js
let x = require('./utils')          // importa el export del fichero ./utils.js
console.log(x)                      // ???
```

Para poder importar, primero hay que exportarlo en el fichero de origen

```
// utils.js
module.exports = 'lorem ipsum'
```

Require

Cómo puedo exportar varias cosas?

```
// utils.js
let a = 10
let b = 10
let c = 10

module.exports = a
```

Require

Utilizando objetos

```
// utils.js
module.exports = {a: a, b: b, c: c}
```

```
// main.js
let x = require('./utils')
console.log(x.a)
console.log(x.b)
console.log(x.c)
```

ES6 Modules

ES6 Modules

ES6 permite importar otros ficheros mediante **import**

```
// main.js
import x from './utils'
```

Para poder importar, primero hay que exportarlo en el fichero de origen

```
// utils.js
export default "lorem ispum"
```

ES6 Modules

Exportar multiples variables

```
// utils.js
export let a = 10
export let b = 20
export let c = 30

export function suma(){
      return a + b
}
```

```
// main.js
import {a, b, c, suma} from './utils'
```

ES6 Modules

Exportar multiples variables

```
// utils.js
export let a = 10
export let b = 20
export let c = 30

export function suma(){
      return a + b
}
```

Importar multiples variables en un objeto

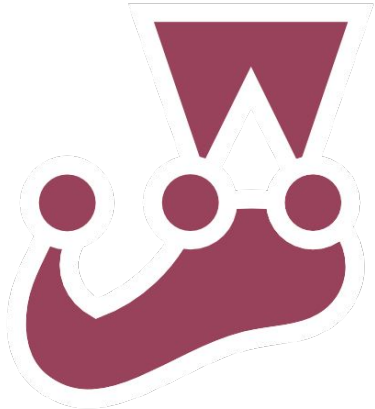
```
// main.js
import * as whatever from './utils'
console.log(whatever.a)
```

Ejercicio ES6 Modules

- Crea un fichero **numbers.js** que exporte **uno**, **dos** y **tres**
- Crea un fichero **index.js** que los importe todos y loguee su valor por la consola

Ejercicio ES6 Modules II

- Crea un fichero **numbers.js** que exporte **uno, dos** y **tres**
- Crea otro fichero **chars.js** que exporte **a, b**, y **c**
- Crea un fichero **numbers_and_letters.js** que exporte dos nombres:
  - **letters**, con **todas** las letras importadas de **chars.js**
  - **numbers**, con **todos** los números importados de **numbers.js**
- Modifica **index.js** para que importe e imprima todos los números y letras desde **numbers_and_letters.js**



Jest

Jest

Jest es un **testing framework** de Javascript muy sencillo de utilizar

Los tests nos permiten comprobar que diferentes piezas de nuestro código devuelven los resultados esperados

Cuando realizamos un cambio en la aplicación podemos correr todos los tests y comprobar que ese cambio no ha roto ninguna de las piezas

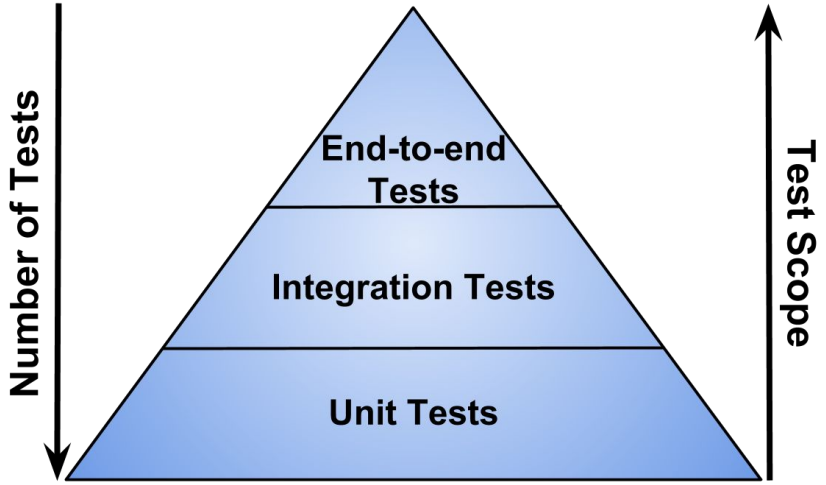
Testing

Los tests son una parte fundamental del desarrollo software

Algunos equipos de desarrollo aplican **test-driven development**

Testing

Hay diferentes niveles de test en función de su alcance



Testing

La mayoría de lenguajes relevantes tienen testing frameworks como Jest para realizar tests unitarios y tests de integración

Los tests end-to-end a menudo requieren herramientas más avanzadas que permitan simular las acciones del usuario final (eg: Selenium)

Jest

npm permite instalar Jest fácilmente

```
npm install jest --save-dev
```

Para poder lanzar los tests desde npm es necesario añadir un script en **package.json**:

```
{
  "scripts": {
    "test": "jest"
  }
}
```

Jest

```
// Fichero en nuestro código
function sum(a, b) {
  return a + b;
}
module.exports = sum;
```

Los tests se crean en ficheros **nombre.test.js (**este formato permite a Jest encontrarlos)

```
// sum.test.js
const sum = require('./sum');

test('adds 1 + 2 to equal 3', () => {
  expect(sum(1, 2)).toBe(3);
});
```

Jest

Lanzamos los tests con npm utilizando el script creado

```
npm run test
```

El comando test nos muestra el resultado de todos los tests

```
PASS  ./sum.test.js
✓ adds 1 + 2 to equal 3 (5ms)
```

Ejercicio Jest pendiente

- Monta el entorno Jest y ejecuta un test

Matchers

Matchers

Expect crea un **expectation object** al que podemos aplicar **matchers**

```
test('two plus two is four', () => {
  expect(2 + 2).toBe(4)
});
```

El matcher **.toBe()** compara la igualdad exacta entre dos valores (**===**)

Matchers

Si queremos comparar el valor de dos objetos no nos vale con igualdad exacta. **toEqual()** efectua **deepComparison**

```
test('object assignment', () => {
  const data = {one: 1}
  data['two'] = 2
  expect(data).toEqual({one: 1, two: 2})
});
```

Matchers

También es posible comprobar la no igualdad

```
test('adding positive numbers is not zero', () => {
  for (let a = 1; a < 10; a++) {
    for (let b = 1; b < 10; b++) {
      expect(a + b).not.toBe(0)
    }
  }
})
```

Matchers

Herramientas para gestionar truthiness

```
test('null', () => {
  const n = null;
  expect(n).toBeNull();
  expect(n).toBeDefined();
  expect(n).not.toBeUndefined();
  expect(n).not.toBeTruthy();
  expect(n).toBeFalsy();
});
```

Matchers

Comparación numérica

```
test('two plus two', () => {
  const value = 2 + 2;
  expect(value).toBeGreaterThan(3);
  expect(value).toBeGreaterThanOrEqual(3.5);
  expect(value).toBeLessThan(5);
  expect(value).toBeLessThanOrEqual(4.5);
});
```

Matchers

Expresiones regulares

```
test('there is no I in team', () => {
  expect('team').not.toMatch(/I/);
});

test('but there is a "stop" in Christoph', () => {
  expect('Christoph').toMatch(/stop/);
});
```

Matchers

Arrays

```
const shoppingList = [
  'diapers',
  'kleenex',
  'trash bags',
  'paper towels',
  'beer',
];

test('the shopping list has beer on it', () => {
  expect(shoppingList).toContain('beer');
});
```

Ejercicio Jest pendiente

Ejercicio Jest pendiente

Setup and Teardown

Setup and Teardown

En ocasiones debemos realizar las mismas tareas una y otra vez antes de cada test

- Eg: Inicializar una base de datos

```
beforeEach(() => {
  initializeCityDatabase() ;
});

afterEach(() => {
  clearCityDatabase() ;
});

test('city database has Vienna', () => {
  expect(isCity('Vienna')).toBeTruthy() ;
});

test('city database has San Juan', () => {
  expect(isCity('San Juan')).toBeTruthy() ;
});
```

Setup and Teardown

En ocasiones debemos realizar una tarea **antes de poder ejecutar cualquier test**

```
beforeAll(() => {
  return initializeCityDatabase();
});

afterAll(() => {
  return clearCityDatabase();
});

test('city database has Vienna' , () => {
  expect(isCity( 'Vienna')).toBeTruthy();
});

test('city database has San Juan' , () => {
  expect(isCity( 'San Juan')).toBeTruthy();
});
```

Setup and Teardown

En ocasiones debemos realizar una tarea **antes de poder ejecutar cualquier test**

```
beforeAll(() => {
  return initializeCityDatabase();
});

afterAll(() => {
  return clearCityDatabase();
});
```

Si beforeAll y afterAll devuelven promesas los tests no se ejectuaran hasta que estas se cumplan

Demo

Ejercicio Jest pendiente