ESPE UNIVERSIDAD DE LAS FUERZAS ARMADAS INNOVACIÓN PARA LA EXCELENCIA	**INFORME DE PRÁCTICA DE LABORATORIO**

DEPARTAMENTO:	**CIENCIAS DE LA COMPUTACION**	**CARRERA:**	**SOFTWARE**		
ASIGNATURA:	Pruebas de Software	**NIVEL:**	Sexto	**FECHA:**	7/2/2026
DOCENTE:	Ing. Luis Castillo, Mgtr.	**PRÁCTICA N°:**	2	**CALIFICACIÓN:**	

CI/CD usando GitHub Actions

Leonardo Vinicio Narváez Criollo

RESUMEN

El presente laboratorio describe la implementación de un flujo de trabajo de Integración Continua (CI) y una simulación de Entrega Continua (CD) utilizando GitHub Actions sobre un proyecto basado en Node.js. Durante la práctica, se configuró un entorno automatizado para la gestión de dependencias, la validación de la calidad del código mediante el análisis estático con ESLint y la verificación de la lógica de negocio a través de pruebas unitarias con el framework Jest. El proceso permitió observar cómo los disparadores (triggers) de Git facilitan la detección temprana de errores al ejecutar de forma automática los flujos de trabajo ante cada cambio en el repositorio. Finalmente, se concluyó que la automatización de estas etapas no solo reduce la probabilidad de introducir fallos en producción, sino que estandariza los criterios de calidad dentro de un equipo de desarrollo, optimizando los tiempos de entrega y la fiabilidad del software desarrollado.

Palabras Claves: Integración Continua, Automatización, Pruebas Unitarias.

1. INTRODUCCIÓN:

La integración continua (CI) es una práctica fundamental del desarrollo de software moderno. Este laboratorio tiene como propósito familiarizar con la automatización de tareas esenciales como la instalación de dependencias, la ejecución de pruebas unitarias y la verificación de calidad del código mediante ESLint, todo ello gestionado a través de GitHub Actions. A través de una aplicación sencilla en Node.js, se experimentará el poder de los flujos automatizados y se comprenderá la importancia de detectar errores temprano en el ciclo de vida del desarrollo.

2. OBJETIVO(S):

2.1 Configurar un flujo de integración continua (CI) en GitHub Actions que se active automáticamente con cada push o pull request a la rama principal del repositorio.
2.2 Implementar pruebas unitarias usando Jest, garantizando que la lógica del sistema funcione correctamente en cada actualización del código.
2.3 Aplicar análisis estático de código con ESLint, reforzando buenas prácticas de programación y detección temprana de errores o inconsistencias.
2.4 Simular un proceso de despliegue automatizado, demostrando cómo se automatizan las etapas previas al paso final de entrega continua (CD), aún sin depender de un proveedor de hosting.

3. MARCO TEÓRICO:

Para el desarrollo de esta práctica, es fundamental comprender los pilares de la automatización moderna en el ciclo de vida de desarrollo de software (SDLC):

- **Integración Continua (CI) y Entrega Continua (CD):** la Integración Continua es una práctica de desarrollo en la que los desarrolladores integran su código en un repositorio compartido de forma frecuente. Cada integración es verificada por una compilación automatizada y pruebas para detectar errores lo antes posible. Por otro lado, la Entrega Continua asegura que el software pueda ser liberado a producción en cualquier momento de forma confiable.

- **GitHub Actions:** es una plataforma de automatización que permite crear flujos de trabajo (workflows) directamente en el repositorio de GitHub. Utiliza archivos de configuración en formato YAML para definir eventos (como un push o pull_request) que desencadenan una serie de trabajos (jobs) ejecutados en máquinas virtuales denominadas runners.

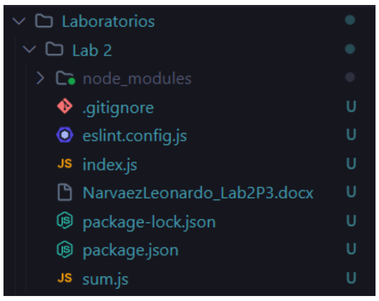
- **Jest y Pruebas Unitarias:** jest es un framework de pruebas de JavaScript diseñado con un enfoque en la simplicidad. En el contexto de CI, las pruebas unitarias permiten validar que pequeñas unidades de código (como funciones matemáticas) se comporten de la manera esperada ante diferentes entradas, sirviendo como la primera línea de defensa contra regresiones.

- **ESLint (Análisis Estático):** ESLint es una herramienta de análisis estático que identifica patrones problemáticos en el código JavaScript. Ayuda a mantener un estilo de codificación consistente y a prevenir errores comunes de sintaxis o lógica antes de que el código sea ejecutado.

4. DESCRIPCIÓN DEL PROCEDIMIENTO:

PARTE 1: Establecimiento de la estructura del proyecto base

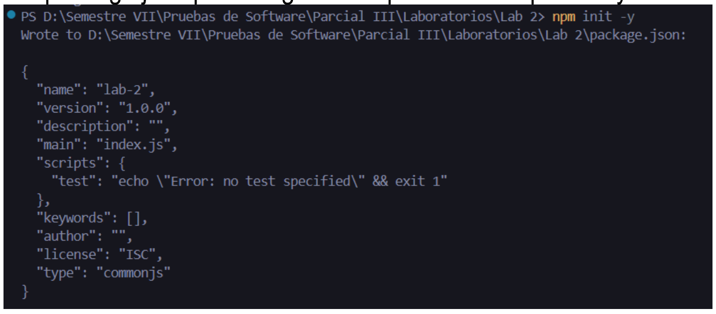
Paso 1: Creación de la estructura básica.



```
Laboratorios
  Lab 2
    node_modules
    .gitignore                      U
    eslint.config.js                U
    index.js                        U
    NarvaezLeonardo_Lab2P3.docx     U
    package-lock.json               U
    package.json                    U
    sum.js                          U
```

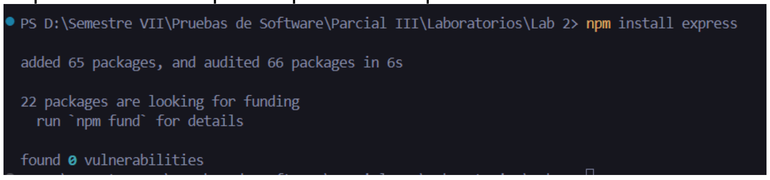
Paso 2: Instalación de dependencias necesarias.

a. Creamos el archivo package.json para cargar las dependencias npm init -y



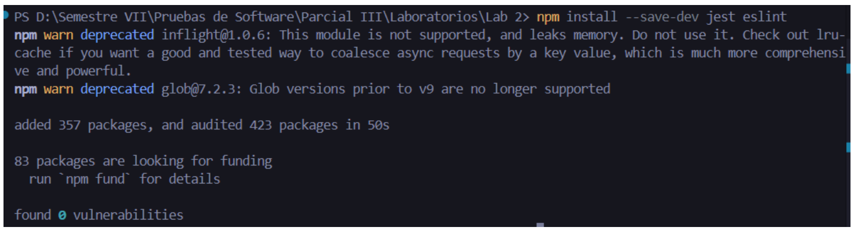
```
PS D:\Semestre VII\Pruebas de Software\Parcial III\Laboratorios\Lab 2> npm init -y
Wrote to D:\Semestre VII\Pruebas de Software\Parcial III\Laboratorios\Lab 2\package.json:

{
  "name": "lab-2",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "type": "commonjs"
}
```

b. Instalamos la dependencia de Express npm install express



```
PS D:\Semestre VII\Pruebas de Software\Parcial III\Laboratorios\Lab 2> npm install express

added 65 packages, and audited 66 packages in 6s

22 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

c. Instalamos las dependencias de Jest y ESLint npm install --save-dev jest eslint para que se puedan ejecutar en modo desarrollador

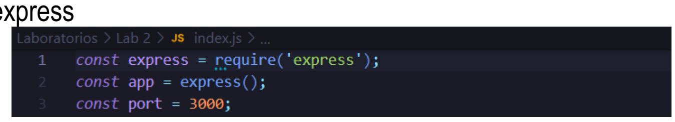


```
PS D:\Semestre VII\Pruebas de Software\Parcial III\Laboratorios\Lab 2> npm install --save-dev jest eslint
npm warn deprecated inflight@1.0.6: This module is not supported, and leaks memory. Do not use it. Check out lru-
cache if you want a good and tested way to coalesce async requests by a key value, which is much more comprehensi
ve and powerful.
npm warn deprecated glob@7.2.3: Glob versions prior to v9 are no longer supported

added 357 packages, and audited 423 packages in 50s

83 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

PARTE 2: Creación de archivos base

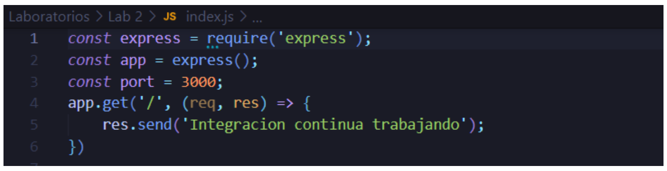
Paso 1: Crear archivo index.js.

a. Usar el servidor express



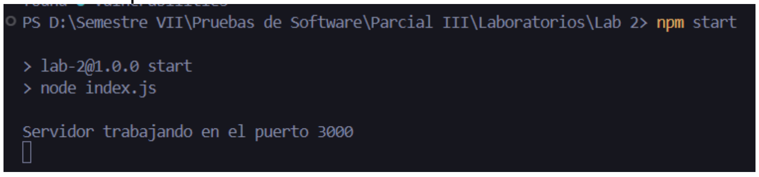
```
Laboratorios > Lab 2 > JS index.js > ...
const express = require('express');
const app = express();
const port = 3000;
```

b. Implementar un endpoint sencillo que responda con un mensaje



```
Laboratorios > Lab 2 > JS index.js > ...
const express = require('express');
const app = express();
const port = 3000;
app.get('/', (req, res) => {
    res.send('Integracion continua trabajando');
})
```

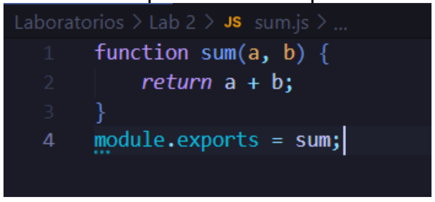
c. Levantar el servidor en el puerto 3000



```
PS D:\Semestre VII\Pruebas de Software\Parcial III\Laboratorios\Lab 2> npm start

> lab-2@1.0.0 start
> node index.js

Servidor trabajando en el puerto 3000
```

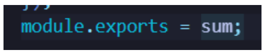
Paso 2: Crear archivo sum.js.

a. Crear una función que sume dos números pasados como parámetros



```
Laboratorios > Lab 2 > JS sum.js > ...
function sum(a, b) {
    return a + b;
}
module.exports = sum;
```

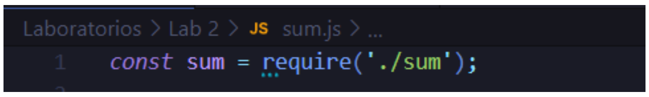
b. Exportar la función.



```
module.exports = sum;
```

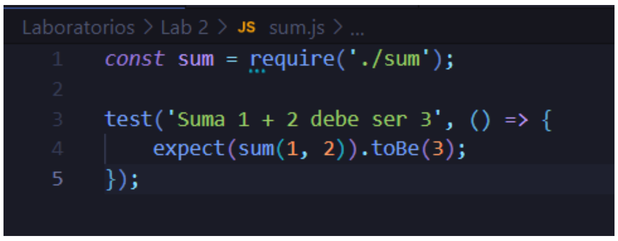
Paso 3: Crear archivo sum.test.js.

a. Usar el archivo con la función de suma



```
Laboratorios > Lab 2 > JS sum.js > ...
const sum = require('./sum');
```

b. Crear una prueba para la función de suma.

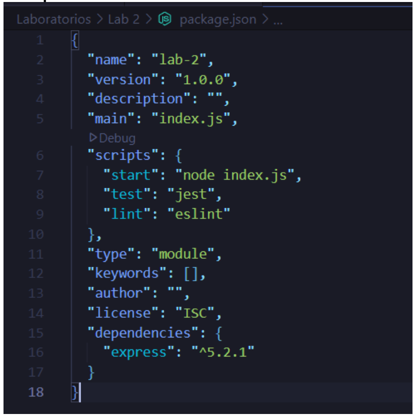


```
Laboratorios > Lab 2 > JS sum.js > ...
const sum = require('./sum');

test('Suma 1 + 2 debe ser 3', () => {
    expect(sum(1, 2)).toBe(3);
});
```

Paso 4: Configurar package.json.

a. Agregar o editar los scripts para start, test y lint

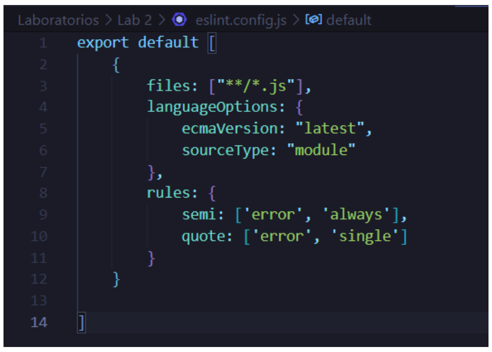
b. Agregar la característica type para que ESLint funcione como módulo.



```
Laboratorios > Lab 2 > package.json > ...
{
  "name": "lab-2",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  ▷Debug
  "scripts": {
    "start": "node index.js",
    "test": "jest",
    "lint": "eslint"
  },
  "type": "module",
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^5.2.1"
  }
}
```

Paso 5: Crear el archivo ESLint.

a. Trabajar con reglas sencillas

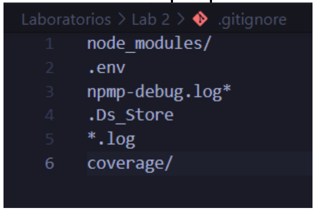


Laboratorios > Lab 2 > eslint.config.js > default

```
export default [
    {
        files: ["**/*.js"],
        languageOptions: {
            ecmaVersion: "latest",
            sourceType: "module"
        },
        rules: {
            semi: ['error', 'always'],
            quote: ['error', 'single']
        }
    }

]
```

Paso 6: Ignorar node_modules.

a. En el archivo .gitignore ignorar todos los archivos que puedan causar conflictos para un proyecto NodeJS



Laboratorios > Lab 2 > .gitignore

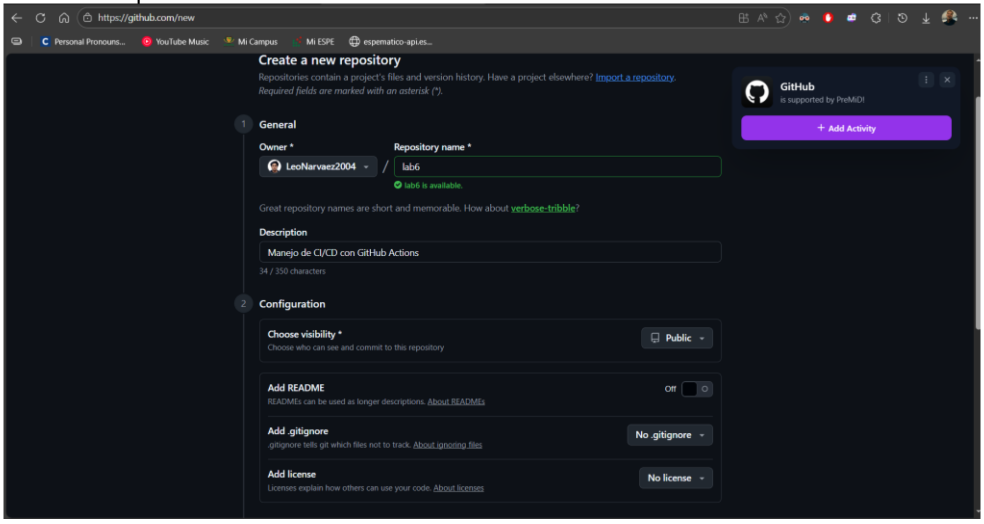
```
node_modules/
.env
npmp-debug.log*
.Ds_Store
*.log
coverage/
```

PARTE 3: Configuración de Git

Paso 1: Crear repositorio en la cuenta de Git.

a. Abrir la cuenta de Git en el navegador

b. Crear un nuevo repositorio vacío



https://github.com/new

Create a new repository

Repositories contain a project's files and version history. Have a project elsewhere? Import a repository.
Required fields are marked with an asterisk (*).

1 General

Owner * LeoNarvaez2004 / Repository name * lab6

lab6 is available.

Great repository names are short and memorable. How about verbose-tribble?

Description: Manejo de CI/CD con GitHub Actions

34 / 350 characters

2 Configuration

Choose visibility * — Choose who can see and commit to this repository: Public

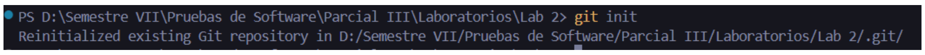
Add README — READMEs can be used as longer descriptions. About READMEs: Off

Add .gitignore — .gitignore tells git which files not to track. About ignoring files: No .gitignore

Add license — Licenses explain how others can use your code. About licenses: No license

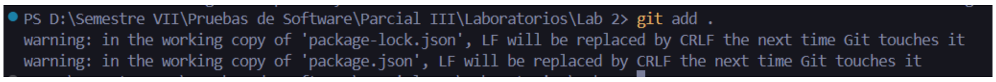
Paso 2: Ejecución de comandos para clonar al repositorio.

a. git init



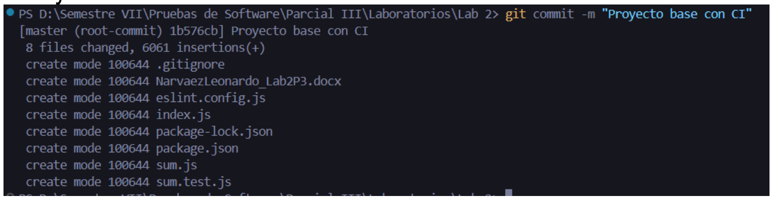
```
PS D:\Semestre VII\Pruebas de Software\Parcial III\Laboratorios\Lab 2> git init
Reinitialized existing Git repository in D:/Semestre VII/Pruebas de Software/Parcial III/Laboratorios/Lab 2/.git/
```

b. git add .



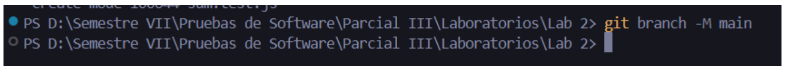
```
PS D:\Semestre VII\Pruebas de Software\Parcial III\Laboratorios\Lab 2> git add .
warning: in the working copy of 'package-lock.json', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'package.json', LF will be replaced by CRLF the next time Git touches it
```

c. git commit -m "Proyecto base con CI"



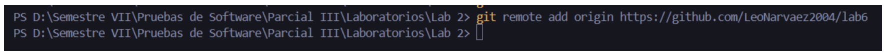
```
PS D:\Semestre VII\Pruebas de Software\Parcial III\Laboratorios\Lab 2> git commit -m "Proyecto base con CI"
[master (root-commit) 1b576cb] Proyecto base con CI
 8 files changed, 6061 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 NarvaezLeonardo_Lab2P3.docx
 create mode 100644 eslint.config.js
 create mode 100644 index.js
 create mode 100644 package-lock.json
 create mode 100644 package.json
 create mode 100644 sum.js
 create mode 100644 sum.test.js
```

d. git branch -M main



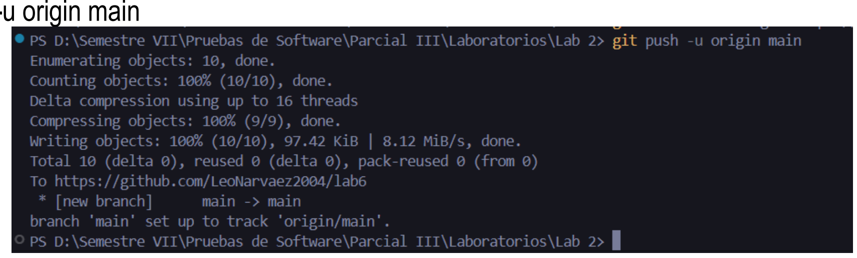
```
PS D:\Semestre VII\Pruebas de Software\Parcial III\Laboratorios\Lab 2> git branch -M main
PS D:\Semestre VII\Pruebas de Software\Parcial III\Laboratorios\Lab 2>
```

e. git remote add origin https://github.com/TU_USUARIO/nombreRepositorio.git



```
PS D:\Semestre VII\Pruebas de Software\Parcial III\Laboratorios\Lab 2> git remote add origin https://github.com/LeoNarvaez2004/lab6
PS D:\Semestre VII\Pruebas de Software\Parcial III\Laboratorios\Lab 2>
```

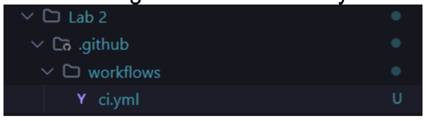
f. git push -u origin main



```
PS D:\Semestre VII\Pruebas de Software\Parcial III\Laboratorios\Lab 2> git push -u origin main
Enumerating objects: 10, done.
Counting objects: 100% (10/10), done.
Delta compression using up to 16 threads
Compressing objects: 100% (9/9), done.
Writing objects: 100% (10/10), 97.42 KiB | 8.12 MiB/s, done.
Total 10 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/LeoNarvaez2004/lab6
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.
PS D:\Semestre VII\Pruebas de Software\Parcial III\Laboratorios\Lab 2>
```

Paso 3: Crear el workflow de GitHub Actions.

a. Crear un archivo nuevo para el workflow .github/workflows/ci.yml.

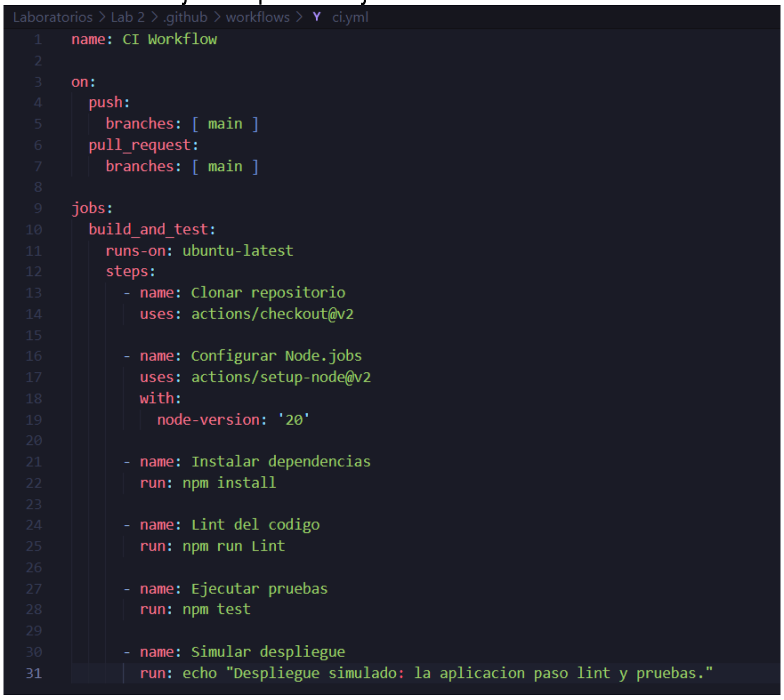


```
Lab 2
  .github
    workflows
      ci.yml                U
```

b. Configurar los triggers.

c. Configurar los trabajos a realizar

d. Configurar dentro de los trabajos los pasos a ejecutarse.

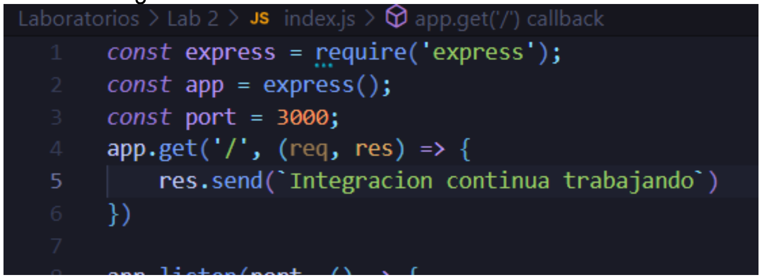


Laboratorios > Lab 2 > .github > workflows > ci.yml

```yaml
name: CI Workflow

on:
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]

jobs:
  build_and_test:
    runs-on: ubuntu-latest
    steps:
      - name: Clonar repositorio
        uses: actions/checkout@v2

      - name: Configurar Node.jobs
        uses: actions/setup-node@v2
        with:
          node-version: '20'

      - name: Instalar dependencias
        run: npm install

      - name: Lint del codigo
        run: npm run Lint

      - name: Ejecutar pruebas
        run: npm test

      - name: Simular despliegue
        run: echo "Despliegue simulado: la aplicacion paso lint y pruebas."
```

Paso 4: Probar la CI.

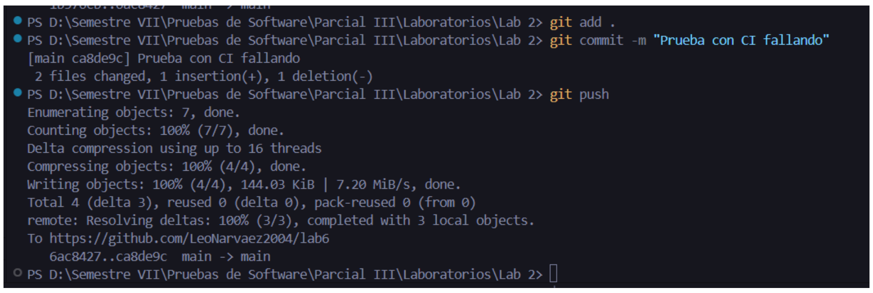
a. Realizar un cambio al código.



Laboratorios > Lab 2 > index.js > app.get('/') callback

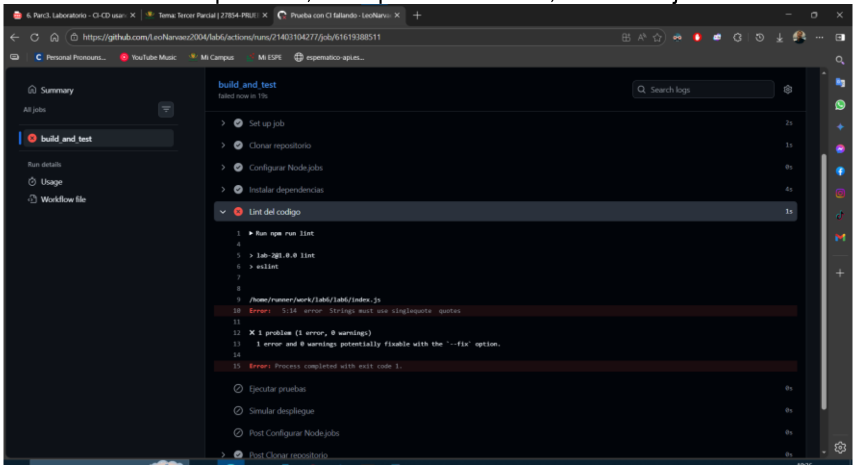
```javascript
const express = require('express');
const app = express();
const port = 3000;
app.get('/', (req, res) => {
    res.send(`Integracion continua trabajando`)
})

```

b. Ejecutar de nuevo los comandos para realizar un nuevo push.



```
● PS D:\Semestre VII\Pruebas de Software\Parcial III\Laboratorios\Lab 2> git add .
● PS D:\Semestre VII\Pruebas de Software\Parcial III\Laboratorios\Lab 2> git commit -m "Prueba con CI fallando"
  [main ca8de9c] Prueba con CI fallando
   2 files changed, 1 insertion(+), 1 deletion(-)
● PS D:\Semestre VII\Pruebas de Software\Parcial III\Laboratorios\Lab 2> git push
  Enumerating objects: 7, done.
  Counting objects: 100% (7/7), done.
  Delta compression using up to 16 threads
  Compressing objects: 100% (4/4), done.
  Writing objects: 100% (4/4), 144.03 KiB | 7.20 MiB/s, done.
  Total 4 (delta 3), reused 0 (delta 0), pack-reused 0 (from 0)
  remote: Resolving deltas: 100% (3/3), completed with 3 local objects.
  To https://github.com/LeoNarvaez2004/lab6
     6ac8427..ca8de9c  main -> main
○ PS D:\Semestre VII\Pruebas de Software\Parcial III\Laboratorios\Lab 2>
```

c. Revisar en GitHub dentro del repositorio, en la pestaña Actions, como se ejecutan los Worflows



https://github.com/LeoNarvaez2004/lab6/actions/runs/21403104277/job/61619388511

build_and_test
failed now in 19s

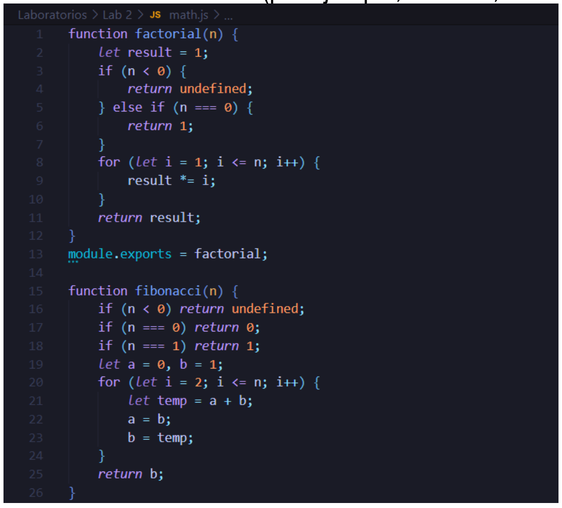
- Set up job 2s
- Clonar repositorio 1s
- Configurar Node.jobs 0s
- Instalar dependencias 4s
- Lint del codigo 1s

```
▶ Run npm run lint

> lab-2@1.0.0 lint
> eslint


/home/runner/work/lab6/lab6/index.js
Error:   5:14  error  Strings must use singlequote  quotes

✖ 1 problem (1 error, 0 warnings)
  1 error and 0 warnings potentially fixable with the `--fix` option.

Error: Process completed with exit code 1.
```

- Ejecutar pruebas 0s
- Simular despliegue 0s
- Post Configurar Node.jobs 0s
- Post Clonar repositorio 0s

5. PREGUNTAS/ACTIVIDADES:

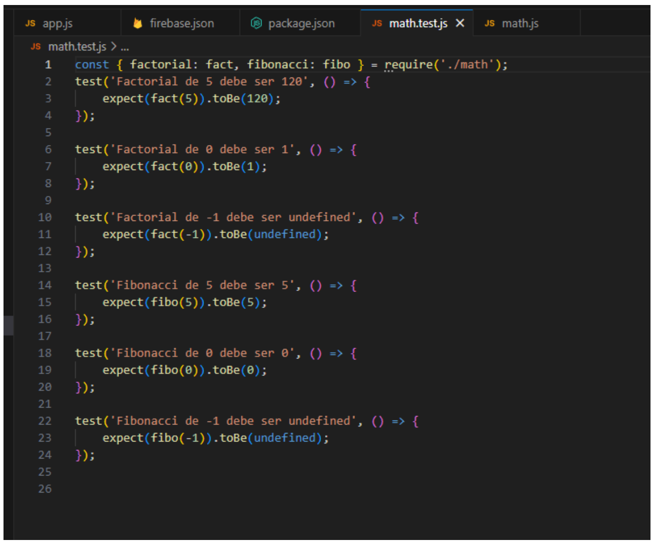
- Agregar más pruebas unitarias
  - Agregar al menos 2 funciones nuevas (por ejemplo, factorial, fibonacci) en un archivo math.js.



Laboratorios > Lab 2 > JS math.js > ...

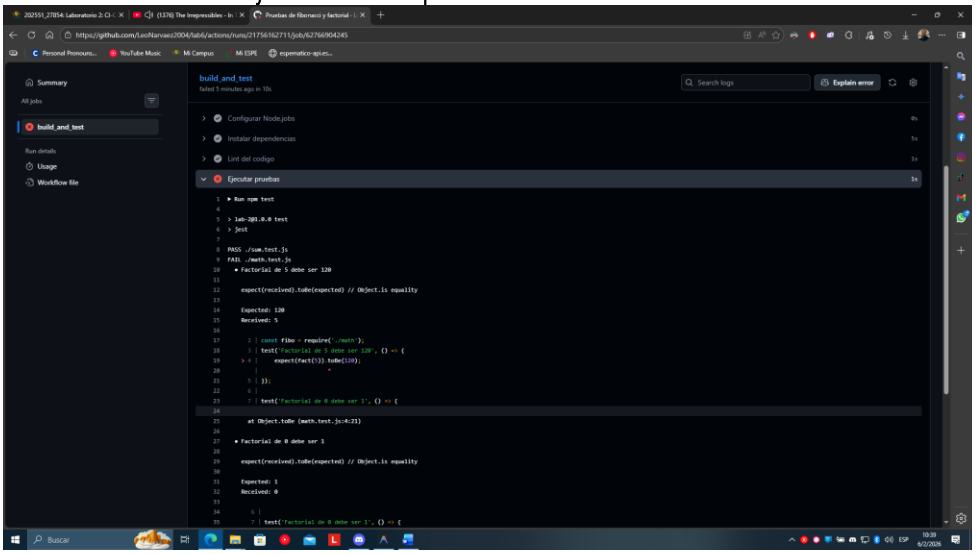
```javascript
function factorial(n) {
    let result = 1;
    if (n < 0) {
        return undefined;
    } else if (n === 0) {
        return 1;
    }
    for (let i = 1; i <= n; i++) {
        result *= i;
    }
    return result;
}
module.exports = factorial;

function fibonacci(n) {
    if (n < 0) return undefined;
    if (n === 0) return 0;
    if (n === 1) return 1;
    let a = 0, b = 1;
    for (let i = 2; i <= n; i++) {
        let temp = a + b;
        a = b;
        b = temp;
    }
    return b;
}
```

  - Crear su correspondiente archivo math.test.js con pruebas Jest.



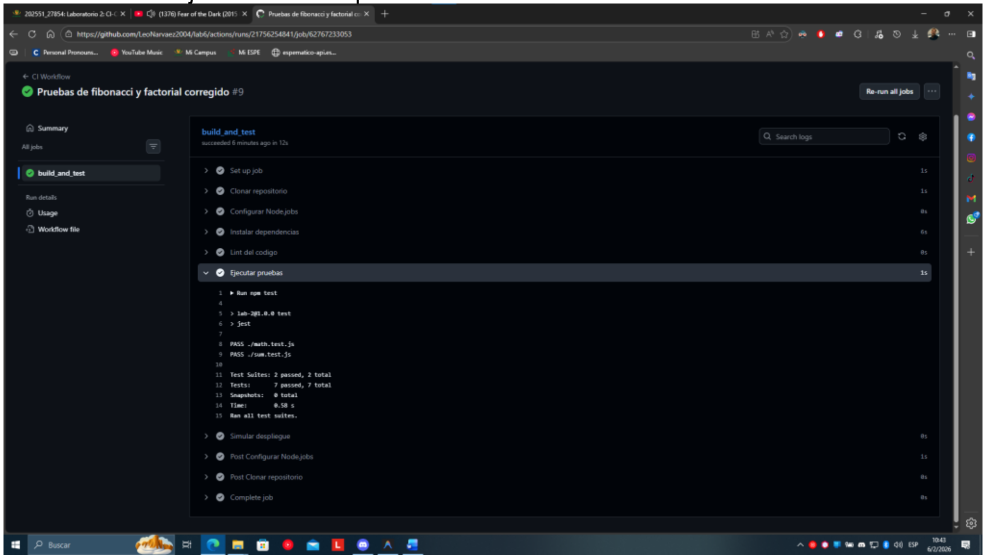
```javascript
const { factorial: fact, fibonacci: fibo } = require('./math');
test('Factorial de 5 debe ser 120', () => {
    expect(fact(5)).toBe(120);
});

test('Factorial de 0 debe ser 1', () => {
    expect(fact(0)).toBe(1);
});

test('Factorial de -1 debe ser undefined', () => {
    expect(fact(-1)).toBe(undefined);
});

test('Fibonacci de 5 debe ser 5', () => {
    expect(fibo(5)).toBe(5);
});

test('Fibonacci de 0 debe ser 0', () => {
    expect(fibo(0)).toBe(0);
});

test('Fibonacci de -1 debe ser undefined', () => {
    expect(fibo(-1)).toBe(undefined);
});
```

- Asegurarse de que GitHub Actions ejecute todas las pruebas con éxito.
    - Primera ejecución de las pruebas con errores:



```
▶ Run npm test

> lab-2@1.0.0 test
> jest

PASS ./sum.test.js
FAIL ./math.test.js
  ● Factorial de 5 debe ser 120

    expect(received).toBe(expected) // Object.is equality

    Expected: 120
    Received: 5

      2 | const fibo = require('./math');
      3 | test('Factorial de 5 debe ser 120', () => {
    > 4 |     expect(fact(5)).toBe(120);
        |                     ^
      5 | });
      6 |
      7 | test('Factorial de 0 debe ser 1', () => {

      at Object.toBe (math.test.js:4:21)

  ● Factorial de 0 debe ser 1

    expect(received).toBe(expected) // Object.is equality

    Expected: 1
    Received: 0

       6 |
       7 | test('Factorial de 0 debe ser 1', () => {
```

- Ejecución de las pruebas con tasa de éxito del 100%



```
▶ Run npm test

> lab-2@1.0.0 test
> jest

PASS ./math.test.js
PASS ./sum.test.js

Test Suites: 2 passed, 2 total
Tests:       7 passed, 7 total
Snapshots:   0 total
Time:        0.58 s
Ran all test suites.
```

6. CONCLUSIONES:

- Se logró configurar exitosamente un workflow de GitHub Actions, demostrando que la automatización mediante archivos YAML permite centralizar la lógica de integración, eliminando la necesidad de procesos manuales para la validación del código.
- La implementación de pruebas unitarias con Jest permitió verificar la integridad de la lógica aritmética del sistema, asegurando que nuevas funcionalidades (como factorial o fibonacci) no rompan el comportamiento existente, cumpliendo así con el principio de regresión.
- El uso de ESLint garantizó que el código cargado al repositorio cumpla con estándares de calidad definidos, lo que facilita la mantenibilidad del proyecto y reduce la deuda técnica desde las etapas iniciales del desarrollo.
- A través de la simulación del despliegue automatizado, se comprendió la importancia de las etapas de "pre-flight" (instalación, linting y testing), las cuales actúan como filtros críticos que garantizan que solo el código estable pueda avanzar hacia una fase de producción o entrega final

7. RECOMENDACIONES:

- Modularización de Pruebas: Se recomienda separar las pruebas unitarias por módulos funcionales (por ejemplo, math.test.js separado de api.test.js) para facilitar la depuración cuando un flujo de trabajo de CI falle.
- Gestión de Secretos: Aunque en esta práctica no se conectó a un proveedor de hosting real, se recomienda para proyectos futuros utilizar el apartado de Secrets de GitHub para manejar credenciales de despliegue de forma segura.
- Optimización del Cache: Para reducir el tiempo de ejecución en GitHub Actions, es aconsejable configurar el caché de las dependencias de node_modules dentro del archivo YAML, evitando la descarga repetitiva de paquetes en cada ejecución.

8. BIBLIOGRAFÍA:

GITHUB. (2024). GitHub Actions Documentation. Recuperado de: https://docs.github.com/en/actions.

JEST. (2024). Jest: Delightful JavaScript Testing. Recuperado de: https://jestjs.io/.

FOWLER, M. (2006). Continuous Integration. MartinFowler.com. Recuperado de: https://martinfowler.com/articles/continuousIntegration.html.

SOMMERVILLE, I. (2011). Ingeniería de Software. 9na Edición. Pearson Educación.