

DEPARTAMENTO:	CIENCIAS DE LA COMPUTACIÓN	CARRERA:	SOFTWARE		
ASIGNATURA:	PRUEBAS DE SOFTWARE	NIVEL:	6to	FECHA:	10-08-2025
DOCENTE:	Ing. Luis Castillo, Mgtr.	PRÁCTICA N°:	2 – 3P	CALIFICACIÓN:	

## CI/CD usando GitHub Actions

Anthony Néstor Villarreal Macías

### RESUMEN

En esta práctica de laboratorio, se configuró un flujo de integración continua (CI) con GitHub Actions para una aplicación Node.js. Se desarrolló un proyecto base con Express, pruebas unitarias con Jest y análisis estático con ESLint. El objetivo fue automatizar la instalación de dependencias, pruebas y verificación de código. Se añadieron funciones factorial y Fibonacci con sus pruebas en Jest, y se provocó un error intencional para verificar el flujo CI, corrigiendo posteriormente el fallo. Las capturas de pantalla muestran las ejecuciones exitosa y fallida. GitHub Actions optimizó la detección temprana de errores, mejorando la calidad del software. La experiencia reforzó la importancia de automatizar procesos para mantener un desarrollo eficiente y confiable, destacando la facilidad de integración de herramientas modernas.

**Palabras Claves:** CI/CD, GitHub Actions, Pruebas Unitarias.

#### 1. INTRODUCCIÓN:

La integración continua (CI) es una práctica clave en el desarrollo de software moderno que permite automatizar procesos para garantizar la calidad del código. Este laboratorio busca familiarizar al estudiante con la configuración de flujos CI/CD mediante GitHub Actions, utilizando una aplicación Node.js. Se realizaron actividades como la creación de pruebas unitarias con Jest, análisis estático con ESLint y simulación de despliegue automatizado, respetando las buenas prácticas de programación y la disciplina en el laboratorio.

#### 2. OBJETIVO(S):

2.1 Describir los alcances o metas de la práctica

- 2.1.1 Configurar un flujo de integración continua en GitHub Actions que se active con cada push o pull request.
- 2.1.2 Implementar pruebas unitarias con Jest para validar la lógica del sistema.
- 2.1.3 Aplicar análisis estático con ESLint para garantizar buenas prácticas de codificación.
- 2.1.4 Simular un proceso de despliegue automatizado, enfocándose en las etapas previas al despliegue continuo.

#### 3. MARCO TEÓRICO:

GitHub Actions es una herramienta de automatización que permite crear flujos de trabajo (workflows) para CI/CD. Ejecuta tareas como instalación de dependencias, pruebas y análisis de código en respuesta a eventos en el repositorio, como push o pull requests. Jest es un framework de pruebas unitarias para JavaScript, conocido por su simplicidad y soporte para pruebas asíncronas. ESLint es una herramienta de linting que analiza el código estáticamente para detectar errores y asegurar consistencia en el estilo de programación. Express es un framework minimalista para Node.js que facilita la creación de APIs. Estas herramientas, combinadas, permiten desarrollar software robusto y automatizado.

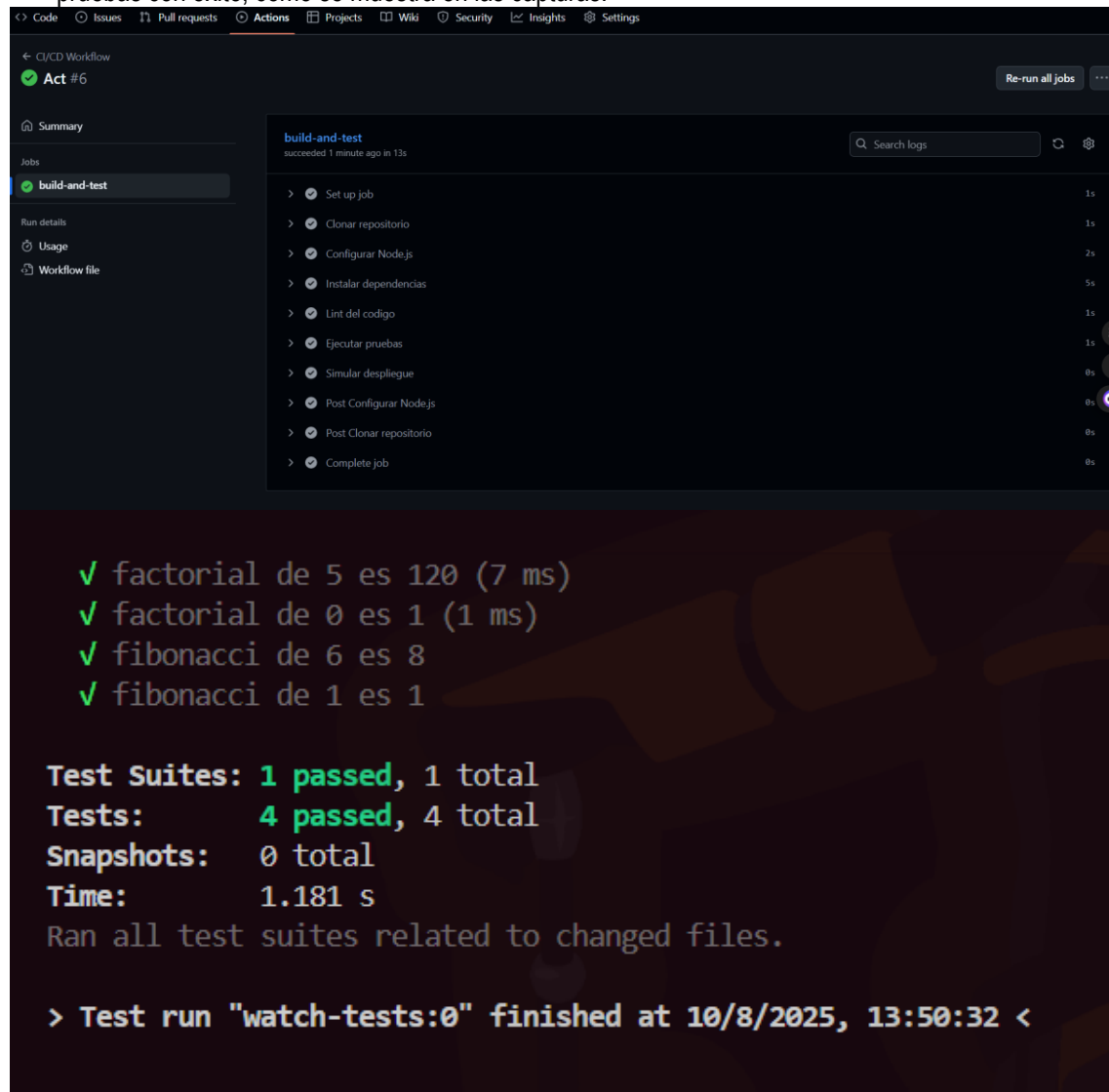
#### 4. DESCRIPCIÓN DEL PROCEDIMIENTO:

Se creó un proyecto Node.js con Express, configurando un servidor básico en el puerto 3000. Se implementó una función de suma en sum.js y pruebas unitarias en sum.test.js usando Jest. Se configuró ESLint con reglas básicas y se ignoró node\_modules en '.gitignore'. Se inicializó un repositorio Git, se subieron los cambios y se configuró un flujo en .github/workflows/ci.yml para ejecutar pruebas y linting automáticamente. Se añadieron funciones factorial y

Fibonacci en math.js con pruebas en math.test.js. Finalmente, se provocó un error intencional en una prueba, se verificó el fallo en GitHub Actions, y se corrigió.

## 5. PREGUNTAS/ACTIVIDADES:

- Agregar más pruebas unitarias: Se implementaron funciones factorial y fibonacci en math.js. Se crearon pruebas en math.test.js para validar casos como  $\text{factorial}(5) = 120$  y  $\text{fibonacci}(6) = 8$ . GitHub Actions ejecutó todas las pruebas con éxito, como se muestra en las capturas.



The screenshot displays the GitHub Actions interface for a workflow named 'build-and-test'. The workflow is shown as 'Act #6' and 'succeeded 1 minute ago in 13s'. The left sidebar shows the workflow file and a summary of the job. The main area lists the steps of the workflow, including 'Set up job', 'Clonar repositorio', 'Configurar Node.js', 'Instalar dependencias', 'Lint del código', 'Ejecutar pruebas', 'Simular despliegue', 'Post Configurar Node.js', 'Post Clonar repositorio', and 'Complete job'. Below the steps, the output of the 'Ejecutar pruebas' step is shown, indicating that all tests passed. The output includes the following text:

```
✓ factorial de 5 es 120 (7 ms)
✓ factorial de 0 es 1 (1 ms)
✓ fibonacci de 6 es 8
✓ fibonacci de 1 es 1

Test Suites: 1 passed, 1 total
Tests:       4 passed, 4 total
Snapshots:   0 total
Time:        1.181 s
Ran all test suites related to changed files.

> Test run "watch-tests:0" finished at 10/8/2025, 13:50:32 <
```

- Provocar un error intencional y corregirlo: Se modificó la prueba de sum.test.js para esperar un resultado incorrecto ( $\text{sum}(2,2) = 5$ ). El flujo CI falló, como se evidencia en la captura de GitHub Actions. Posteriormente, se corrigió la prueba ( $\text{sum}(2,2) = 4$ ) y el flujo se ejecutó correctamente.



```
Summary
Jobs
  build-and-test
Run details
  Usage
  Workflow file

build-and-test
failed now in 15s

> Configurar Nodejs 2s
> Instalar dependencias 7s
> Lint del código 0s
  Ejecutar pruebas 2s
    9 PASS ./math.test.js
    10 FAIL ./sum.test.js
    11   • Suma de 2 + 2 debe ser 4
    12     expect(received).toBe(expected) // Object.is equality
    13       Expected: 5
    14       Received: 4
    15       2 |
    16       > | test('Suma de 2 + 2 debe ser 4', () => {
    17         > 4 |   expect(sum(2, 2)).toBe(5);
    18         |   ^
    19         | });
    20       at Object.toBe (sum.test.js:4:23)
    21 Test Suites: 1 failed, 1 passed, 2 total
    22 Tests:      1 failed, 4 passed, 5 total
    23 Snapshots:  0 total
    24 Time:       0.605 s
    25 Ran all test suites.
    31 Error: Process completed with exit code 1.

  Simular despliegue 0s
  Post-Configurar Nodejs 0s
```

**FAIL** ./sum.test.js

- ✖ Suma de 2 + 2 debe ser 4 (10 ms)
- Suma de 2 + 2 debe ser 4

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

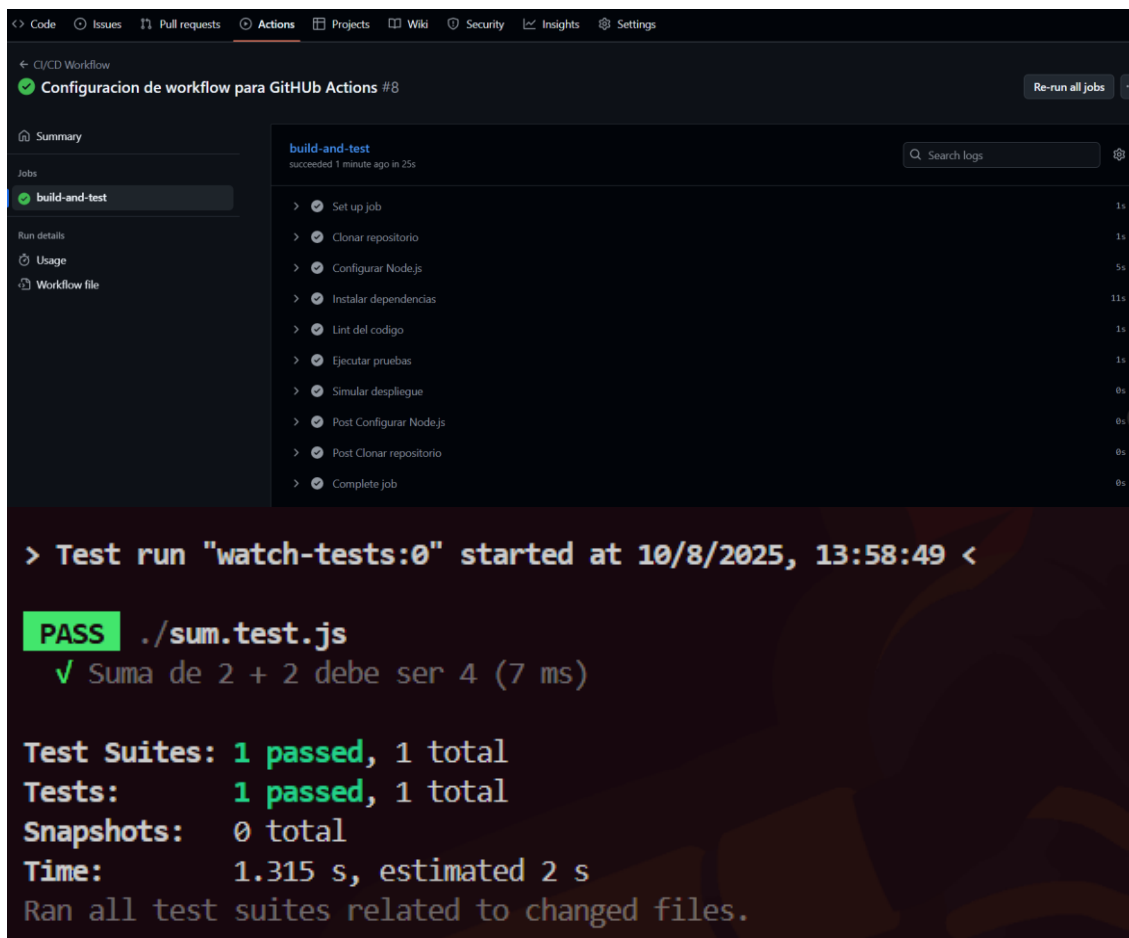
Expected: 5  
Received: 4

```
2 |
3 | test('Suma de 2 + 2 debe ser 4', () => {
> 4 |   expect(sum(2, 2)).toBe(5);
    |   ^
5 | });

at Object.toBe (sum.test.js:4:23)
```

> Test run "watch-tests:0" finished at 10/8/2025, 13:57:16 <

Test Suites: 1 failed, 1 total  
Tests: 1 failed, 1 total  
Snapshots: 0 total  
Time: 1.413 s  
Ran all test suites related to changed files.



## 6. RESULTADOS OBTENIDOS:

El proyecto se configuró exitosamente con un flujo CI/CD en GitHub Actions que automatizó dependencias, pruebas y linting. Las capturas muestran la ejecución exitosa y el fallo intencional. A continuación, se presentan los archivos de código actualizados:

- .gitignore:



- Eslint.config.js:

```
1 export default [  
2   {  
3     files: ['**/*.js'],  
4     languageOptions: {  
5       ecmaVersion: 'latest',  
6       sourceType: 'module',  
7     },  
8     rules: {  
9       semi: ['error', 'always'],  
10      quotes: ['error', 'single']  
11    },  
12  },  
13 ];
```

- **Package.json:**

```
1  {  
2    "name": "lab2",  
3    "version": "1.0.0",  
4    "description": "",  
5    "main": "index.js",  
6    "scripts": {  
7      "test": "jest",  
8      "start": "node index.js",  
9      "lint": "npx eslint ."  
10   },  
11   "keywords": [],  
12   "author": "",  
13   "license": "ISC",  
14   "dependencies": {  
15     "express": "^5.1.0"  
16   },  
17   "devDependencies": {  
18     "eslint": "^9.32.0",  
19     "jest": "^30.0.5"  
20   },  
21   "type": "module"  
22 }  
23
```

- **Index.js:**

```
1 const express = require('express');
2 const app = express();
3 const PORT = process.env.PORT;
4
5 //endpoint que responde con un mensaje de bienvenida
6 app.get('/', (req, res) => {
7   res.send('Integración continua funcionando!');
8 });
9
10 app.listen(PORT, () => {
11   console.log(`Servidor corriendo en el puerto ${PORT}`);
12 });
```

- **Sum.js:**

```
1 function sum(a, b) {
2   return a + b;
3 }
4
5 module.exports = sum;
6
```

- **Sum.test.js:**

```
1 const sum = require('./sum');
2
3 test('Suma de 1 + 2 debe ser 3', () => {
4   expect(sum(1, 2)).toBe(3);
5 });
```

- **Math.js:**

```
1 // Función para calcular factorial
2 function factorial(n) {
3     if (n === 0 || n === 1) return 1;
4     return n * factorial(n - 1);
5 }
6
7 // Función para calcular Fibonacci
8 function fibonacci(n) {
9     if (n <= 1) return n;
10    return fibonacci(n - 1) + fibonacci(n - 2);
11 }
12
13 module.exports = { factorial, fibonacci };
```

- **Math.test.js:**

```
1 const math = require('./math.js');
2
3 // Pruebas para factorial
4 test('factorial de 5 es 120', () => {
5     expect(math.factorial(5)).toBe(120);
6 });
7
8 test('factorial de 0 es 1', () => {
9     expect(math.factorial(0)).toBe(1);
10 });
11
12 // Pruebas para fibonacci
13 test('fibonacci de 6 es 8', () => {
14     expect(math.fibonacci(6)).toBe(8);
15 });
16
17 test('fibonacci de 1 es 1', () => {
18     expect(math.fibonacci(1)).toBe(1);
19 });
```

- **.github/workflows/ci.yml:**

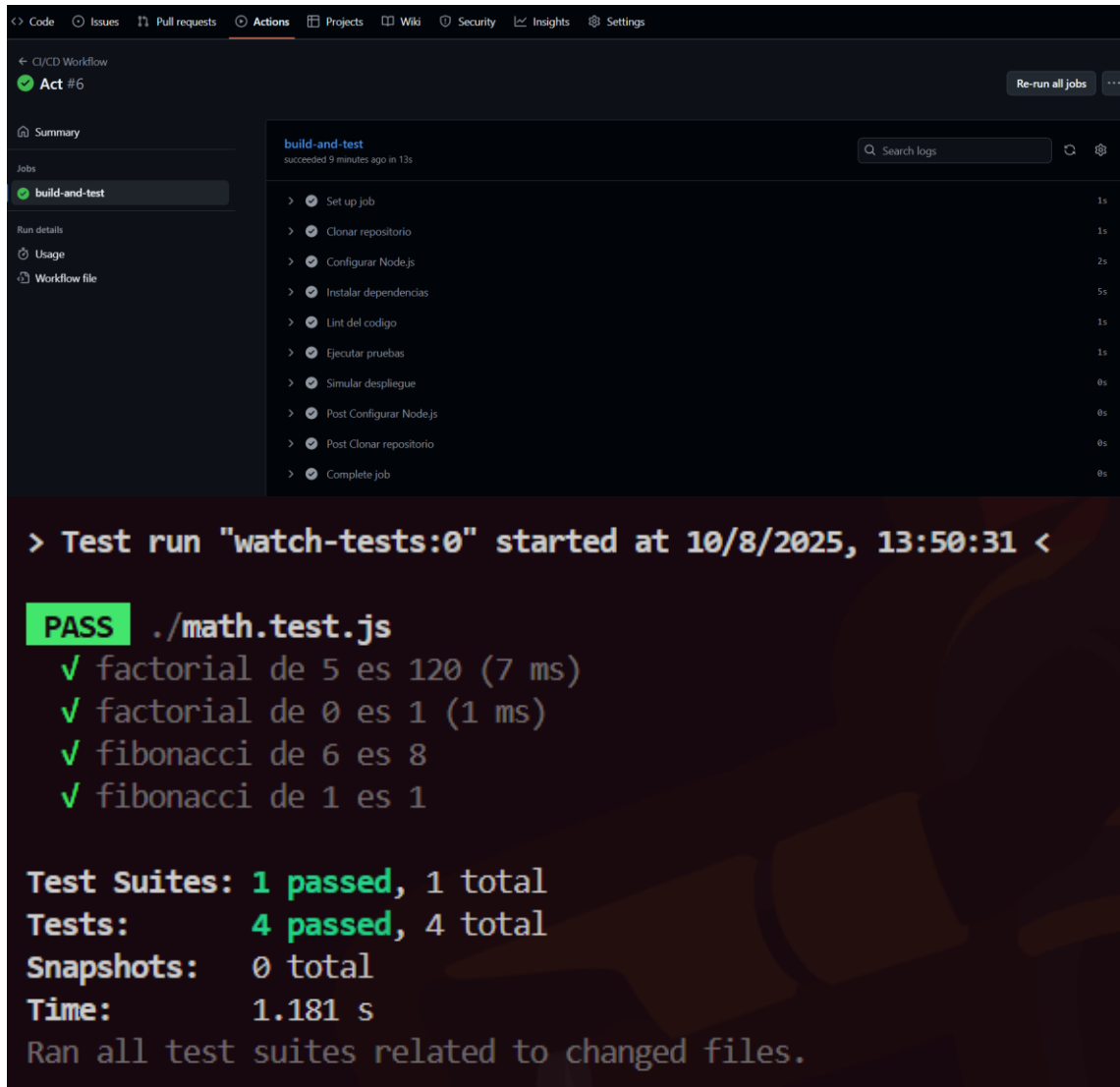
```
1  name: CI/CD Workflow
2
3  on:
4    push:
5      branches: [ main ]
6    pull_request:
7      branches: [ main ]
8
9  jobs:
10   build-and-test:
11     runs-on: ubuntu-latest
12
13     steps:
14       - name: Clonar repositorio
15         uses: actions/checkout@v4
16
17       - name: Configurar Node.js
18         uses: actions/setup-node@v4
19         with:
20           node-version: 20
21
22       - name: Instalar dependencias
23         run: npm install
24
25       - name: Lint del código
26         run: npm run lint
27
28       - name: Ejecutar pruebas
29         run: npm test
30
31       - name: Simular despliegue
32         run: |
33           echo "Despliegue simulado la aplicación paso
34           Lint y Pruebas correctamente"
```

- **.env:**

```
1  PORT=3000
```

- **GITHUB**





```
> Test run "watch-tests:0" started at 10/8/2025, 13:50:31 <

PASS ./math.test.js
  ✓ factorial de 5 es 120 (7 ms)
  ✓ factorial de 0 es 1 (1 ms)
  ✓ fibonacci de 6 es 8
  ✓ fibonacci de 1 es 1

Test Suites: 1 passed, 1 total
Tests:       4 passed, 4 total
Snapshots:   0 total
Time:        1.181 s
Ran all test suites related to changed files.
```

## 7. CONCLUSIONES:

- La configuración de GitHub Actions permitió automatizar pruebas y análisis de código, mejorando la detección temprana de errores.
- Las pruebas unitarias con Jest aseguraron la correcta funcionalidad de las funciones implementadas, fortaleciendo la calidad del software.

## 8. RECOMENDACIONES:

- Incluir más reglas en ESLint para proyectos más complejos, como compatibilidad con frameworks específicos.
- Explorar herramientas de cobertura de pruebas (como Jest coverage) para medir la calidad de las pruebas unitarias.

## 9. BIBLIOGRAFÍA:

- **GitHub.** (2025). GitHub Actions Documentation, GitHub Docs, <https://docs.github.com/en/actions>. Consultada: 10 de agosto de 2025.
- **Jest.** (2025). Jest Documentation, Jest, <https://jestjs.io/docs/getting-started>. Consultada: 10 de agosto de 2025.
- **ESLint.** (2025). ESLint Documentation, ESLint, <https://eslint.org/docs/latest/>. Consultada: 10 de agosto de 2025.