

DEPARTAMENTO:	Ciencias de la Computación	CARRERA:	Ingeniería en Software		
ASIGNATURA:	Pruebas de Software	NIVEL:	6to	FECHA:	10/08/2025
DOCENTE:	Ing. Luis Castillo	PRÁCTICA N°:	6	CALIFICACIÓN:	

CI-CD usando GitHub Actions

Elkin Andres Pabón González

RESUMEN

La presente práctica tuvo como objetivo implementar un flujo de Integración Continua (CI) en una aplicación Node.js utilizando GitHub Actions. Se configuró un repositorio en GitHub y se desarrolló un servidor básico con Express, junto con funciones y pruebas unitarias usando Jest. Se aplicó análisis estático de código mediante ESLint para garantizar la calidad y buenas prácticas de programación. El flujo de CI se configuró para ejecutarse automáticamente ante cada push o pull request a la rama principal, validando la instalación de dependencias, la ejecución de pruebas y la verificación de estilo. Además, se agregaron funciones adicionales para ampliar la cobertura de pruebas y se provocaron errores intencionales para comprobar el correcto funcionamiento del flujo de integración continua. Finalmente, se evidenció la importancia de la detección temprana de errores y de la automatización en el ciclo de desarrollo, mejorando la eficiencia y la confiabilidad del software.

Palabras Claves: Integración Continua, GitHub Actions, Node.js

1. INTRODUCCIÓN:

La presente práctica se enfoca en la implementación de un flujo de Integración Continua (CI) utilizando GitHub Actions en un proyecto Node.js. Esta actividad permite automatizar tareas como la instalación de dependencias, la ejecución de pruebas unitarias con Jest y el análisis estático de código con ESLint, garantizando así la calidad y la estabilidad del software. Durante el desarrollo, se fomentó el manejo responsable de los recursos del laboratorio y el cumplimiento de las normas establecidas para el uso del equipo y del entorno de trabajo. Además, se incentivó la disciplina en el versionamiento del código mediante el uso de Git y GitHub, asegurando un control preciso de cambios y facilitando la colaboración. La práctica busca reforzar competencias técnicas y organizativas esenciales para entornos de desarrollo profesional.

2. OBJETIVO(S):

- Implementar un flujo de Integración Continua en un proyecto Node.js utilizando GitHub Actions para automatizar tareas esenciales del desarrollo.
- Configurar workflows que se activen automáticamente en cada actualización del repositorio, verificando dependencias, ejecución de pruebas y análisis de código.
- Desarrollar y ejecutar pruebas unitarias con Jest que validen el correcto funcionamiento de las funciones y detecten errores tempranamente.
- Aplicar análisis estático de código con ESLint para mantener estándares de calidad y reforzar buenas prácticas de programación.

3. MARCO TEÓRICO:

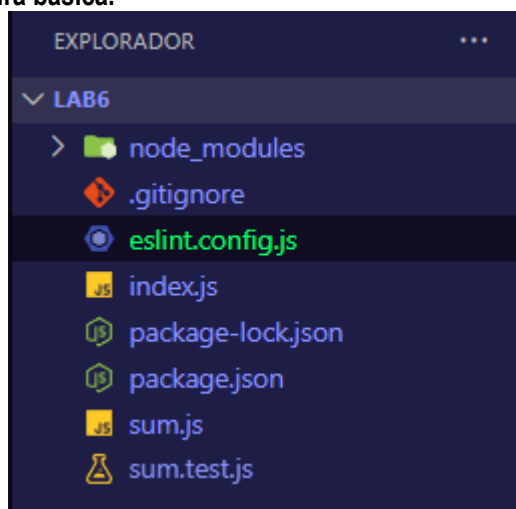
- Integración Continua (CI):** Es una práctica de desarrollo de software que consiste en integrar el código de forma frecuente en un repositorio compartido, ejecutando automáticamente pruebas y validaciones para detectar errores de manera temprana. Su objetivo es mantener la calidad y estabilidad del producto durante todo el ciclo de desarrollo.
- Entrega Continua (CD):** Es la extensión de la integración continua que permite que el software esté siempre listo para ser desplegado en producción. Automatiza las fases posteriores a la integración, como el empaquetado y la liberación del software, reduciendo tiempos y riesgos en el despliegue.

- **GitHub Actions:** Es una herramienta de automatización integrada en GitHub que permite ejecutar flujos de trabajo (workflows) definidos en archivos YAML. Estos flujos pueden ejecutarse en respuesta a eventos como commits, pull requests o programación programada, y permiten automatizar pruebas, compilaciones, despliegues y otras tareas.
- **Node.js:** Es un entorno de ejecución de JavaScript del lado del servidor basado en el motor V8 de Google Chrome. Permite construir aplicaciones escalables y de alto rendimiento, siendo ampliamente utilizado en proyectos que requieren rapidez y concurrencia.
- **Jest:** Es un framework de pruebas para JavaScript que facilita la creación y ejecución de pruebas unitarias. Ofrece funcionalidades como aserciones, mocks y medición de cobertura de código, siendo muy utilizado en proyectos Node.js y React.
- **ESLint:** Es una herramienta de análisis estático para JavaScript que permite identificar y corregir patrones problemáticos o no conformes a las reglas de estilo definidas. Ayuda a mantener un código limpio, consistente y libre de errores comunes.

4. DESCRIPCIÓN DEL PROCEDIMIENTO:

PARTE 1: Establecimiento de la estructura del proyecto base

Paso 1: Creación de la estructura básica.



Paso 2: Instalación de dependencias necesarias.

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

```
PS C:\Users\Elkin Andres\Desktop\LAB6> npm init -y
Wrote to C:\Users\Elkin Andres\Desktop\LAB6\package.json:

{
  "name": "lab6",
  "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 C:\Users\Elkin Andres\Desktop\LAB6> npm install express

added 67 packages, and audited 68 packages in 3s

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

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

```
found 0 vulnerabilities
PS C:\Users\Elkin Andres\Desktop\LAB6> 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
high is much more comprehensive and powerful.
npm warn deprecated glob@7.2.3: Glob versions prior to v9 are no longer supported

added 382 packages, and audited 450 packages in 18s

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

found 0 vulnerabilities
PS C:\Users\Elkin Andres\Desktop\LAB6>
```

PARTE 2: Creación de archivos base

Paso 1: Crear archivo index.js.

- Usar el servidor express
- Implementar un endpoint sencillo que responda con un mensaje
- Levantar el servidor en el puerto 3000

```
index.js x sum.test.js sum.js .gitignore eslint.config.js
index.js > ...
1  const express = require('express');
2  const app = express();
3  const port = process.env.PORT
4
5  // endpoint que responde un mensaje
6  app.get('/', (req, res) => {
7    res.send('Integración continua funcionando correctamente');
8  });
9
10 app.listen(PORT, () => {
11   console.log(`Servidor escuchando en el puerto ${PORT}`);
12 });
```

Paso 2: Crear archivo sum.js.

- Crear una función que sume dos números pasados como parámetros
- Exportar la función.

```
index.js sum.test.js sum.js x .gitignore
sum.js > ...
1  function suma(a, b) {
2    return a + b;
3  }
4
5  module.exports = suma;
6
7
```

Paso 3: Crear archivo sum.test.js.

- Usar el archivo con la función de suma
- Crear una prueba para la función de suma.

```
... index.js sum.test.js X sum.js .gitignore
sum.test.js > ...
1 const suma = require('./sum');
2
3 test('Suma 1+2 debe dar 3 ', () => {
4   expect(suma(1, 2)).toBe(3);
5 });
6
```

Paso 4: Configurar package.json.

- Agregar o editar los scripts para start, test y lint
- Agregar la característica type para que ESLint funcione como módulo.

```
package.json > {} devDependencies
1 {
2   "name": "lab6",
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  "type": "commonjs",
15  "dependencies": {
16    "express": "^5.1.0"
17  },
18  "devDependencies": {
19    "eslint": "^9.32.0",
20    "jest": "^30.0.5"
21  }
22 }
23
```

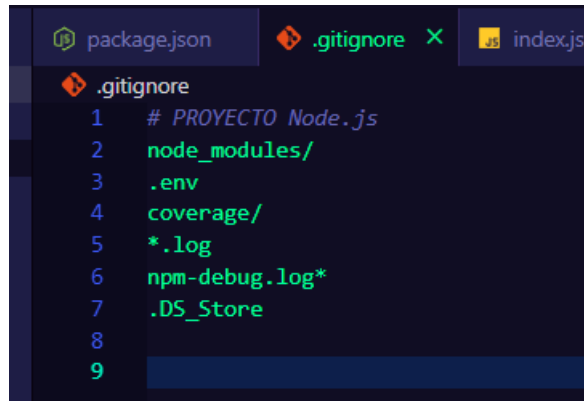
Paso 5: Crear el archivo ESLint.

- Trabajar con reglas sencillas

```
package.json index.js sum.test.js sum.js
eslint.config.js > [0] default
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 ];
```

Paso 6: Ignorar node_modules.

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

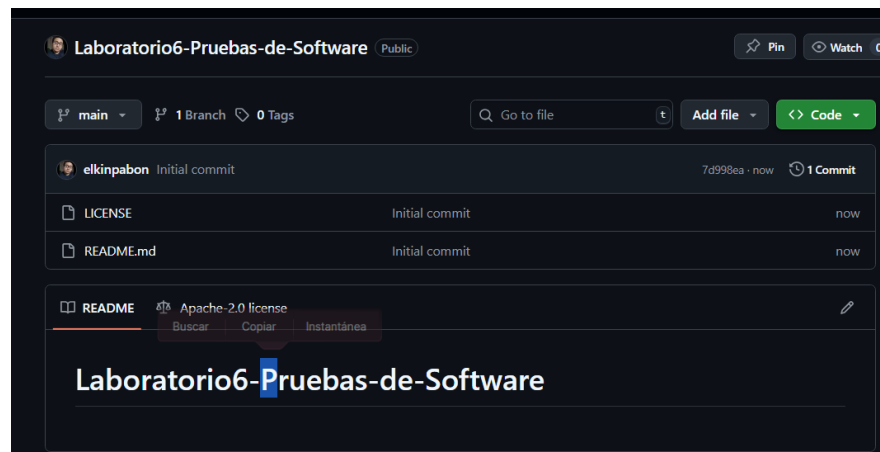


```
package.json .gitignore x index.js
.gitignore
1 # PROYECTO Node.js
2 node_modules/
3 .env
4 coverage/
5 *.log
6 npm-debug.log*
7 .DS_Store
8
9
```

PARTE 3: Configuración de Git

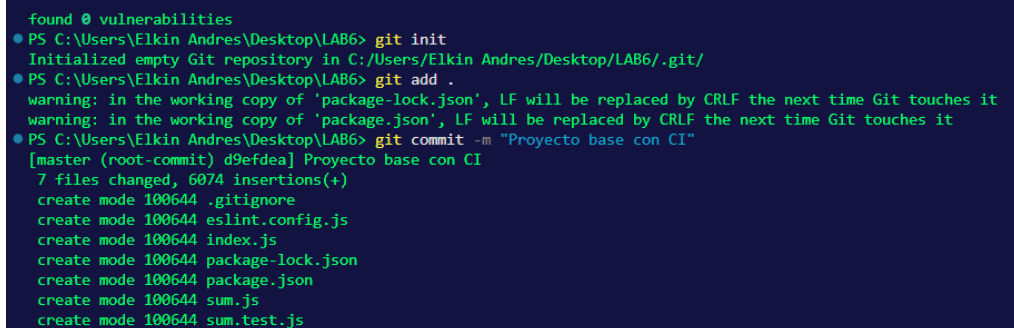
Paso 1: Crear repositorio en la cuenta de Git.

- Abrir la cuenta de Git en el navegador
- Crear un nuevo repositorio vacío



Paso 2: Ejecución de comandos para clonar al repositorio.

- git init
- git add .
- git commit -m "Proyecto base con CI"



```
found 0 vulnerabilities
PS C:\Users\Elkin Andres\Desktop\LAB6> git init
Initialized empty Git repository in C:/Users/Elkin Andres/Desktop/LAB6/.git/
PS C:\Users\Elkin Andres\Desktop\LAB6> 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
PS C:\Users\Elkin Andres\Desktop\LAB6> git commit -m "Proyecto base con CI"
[master (root-commit) d9efdea] Proyecto base con CI
7 files changed, 6074 insertions(+)
create mode 100644 .gitignore
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
```

- git branch -M main
- git remote add origin https://github.com/TU_USUARIO/nombreRepositorio.git
- git push -u origin main

```

PS C:\Users\Elkin Andres\Desktop\LAB6> git remote add origin https://github.com/elkinpabon/Laboratorio6-Pruebas-de-Software.git
error: remote origin already exists.
PS C:\Users\Elkin Andres\Desktop\LAB6> git pull origin main --allow-unrelated-histories
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 4 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Unpacking objects: 100% (4/4), 4.79 KiB | 272.00 KiB/s, done.
From https://github.com/elkinpabon/Laboratorio6-Pruebas-de-Software
* branch      main      -> FETCH_HEAD
* [new branch] main      -> origin/main
Merge made by the 'ort' strategy.
LICENSE | 201 ++++++
README.md | 1 +
2 files changed, 202 insertions(+)
create mode 100644 LICENSE
create mode 100644 README.md
PS C:\Users\Elkin Andres\Desktop\LAB6> git push -u origin main
Enumerating objects: 12, done.
Counting objects: 100% (12/12), done.
Delta compression using up to 16 threads
Compressing objects: 100% (11/11), done.
Writing objects: 100% (11/11), 50.75 KiB | 4.23 MiB/s, done.
Total 11 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (1/1), done.
To https://github.com/elkinpabon/Laboratorio6-Pruebas-de-Software.git
7d998ea..d487ced main -> main
branch 'main' set up to track 'origin/main'.
  
```

Laboratorio6-Pruebas-de-Software Public		
main 1 Branch 0 Tags Go to file Add file Code		
elkinpabon Merge branch 'main' of https://github.com/elkinpabon/Laboratorio6-Pruebas-de-Software:main d487ced · 2 minutes ago 3 Commits		
	.gitignore	Proyecto base con CI 5 minutes ago
	LICENSE	Initial commit 8 minutes ago
	README.md	Initial commit 8 minutes ago
	eslint.config.js	Proyecto base con CI 5 minutes ago
	index.js	Proyecto base con CI 5 minutes ago
	package-lock.json	Proyecto base con CI 5 minutes ago
	package.json	Proyecto base con CI 5 minutes ago
	sum.js	Proyecto base con CI 5 minutes ago
	sum.test.js	Proyecto base con CI 5 minutes ago

Paso 3: Crear el workflow de GitHub Actions.

- Crear un archivo nuevo para el workflow .github/workflows/ci.yml.
- Configurar los triggers.
- Configurar los trabajos a realizar
- Configurar dentro de los trabajos los pasos a ejecutarse.

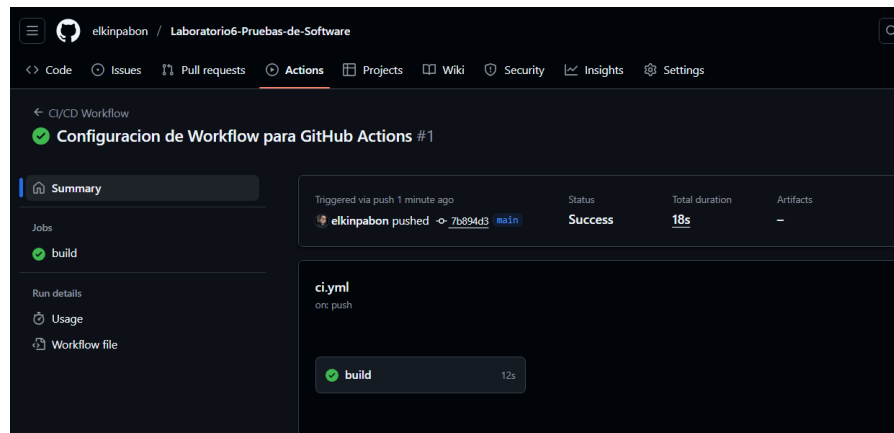
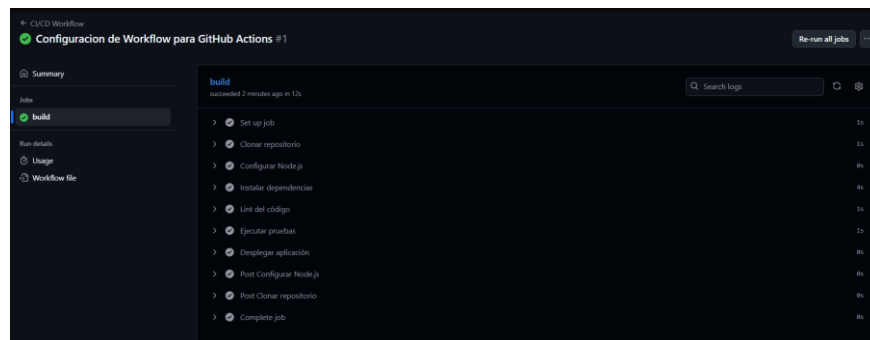
```

package.json  .gitignore  ci.yml  index.js  sum.test.js
.github > workflows > ci.yml > {} jobs > {} build > {} steps > {} 5 > run
GitHub Workflow - YAML GitHub Workflow (github-workflow.json)
1  name: CI/CD Workflow
2
3  on:
4    push:
5      branches: [main]
6    pull_request:
7      branches: [main]
8
9  jobs:
10   build:
11     runs-on: ubuntu-latest
12     steps:
13       - name: Clonar repositorio
14         uses: actions/checkout@v4
15       - name: Configurar Node.js
16         uses: actions/setup-node@v4
17         with:
18           node-version: '20'
19       - name: Instalar dependencias
20         run: npm install
21       - name: Lint del código
22         run: npm run lint
23       - name: Ejecutar pruebas
24         run: npm test
25       - name: Desplegar aplicación
26         run: |
27           echo "Despliegue simulado: La aplicación pasó
28           Lint y pruebas exitosamente."
29
  
```

```
PS C:\Users\Elkin Andres\Desktop\LAB6> git add .
PS C:\Users\Elkin Andres\Desktop\LAB6> git commit -m "Configuracion de Workflow para GitHub Actions"
[main 7b894d3] Configuracion de Workflow para GitHub Actions
1 file changed, 28 insertions(+)
create mode 100644 .github/workflows/ci.yml
PS C:\Users\Elkin Andres\Desktop\LAB6> git push
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 16 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (5/5), 712 bytes | 712.00 KiB/s, done.
Total 5 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/elkinpabon/Laboratorio6-Pruebas-de-Software.git
d487ced..7b894d3 main -> main
PS C:\Users\Elkin Andres\Desktop\LAB6>
```

Paso 4: Probar la CI.

- Realizar un cambio al código.
- Ejecutar de nuevo los comandos para realizar un nuevo push.
- Revisar en GitHub dentro del repositorio, en la pestaña Actions, como se ejecutan los Worflows



5. PREGUNTAS/ACTIVIDADES:

1. Agregar más pruebas unitarias

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

math.js

```

1  /**
2   * Calcula el factorial de un número
3   */
4  function factorial(n) {
5    if (n < 0) {
6      throw new Error('El factorial no está definido para números negativos');
7    }
8    if (n === 0 || n === 1) {
9      return 1;
10   }
11   return n * factorial(n - 1);
12 }
13
14 /**
15 * Calcula el n-ésimo número de la secuencia de Fibonacci
16 */
17 function fibonacci(n) {
18   if (n < 0) {
19     throw new Error('Fibonacci no está definido para números negativos');
20   }
21   if (n === 0) return 0;
22   if (n === 1) return 1;
23
24   let a = 0, b = 1;
25   for (let i = 2; i <= n; i++) {
26     [a, b] = [b, a + b];
27   }
28   return b;
29 }

```

math.test.js

```

30
31 /**
32 * Verifica si un número es primo
33 */
34 function isPrime(n) {
35   if (n <= 1) return false;
36   if (n <= 3) return true;
37   if (n % 2 === 0 || n % 3 === 0) return false;
38
39   for (let i = 5; i * i <= n; i += 6) {
40     if (n % i === 0 || n % (i + 2) === 0) return false;
41   }
42   return true;
43 }
44
45 /**
46 * Calcula la potencia de un número
47 */
48 function power(base, exponent) {
49   if (exponent === 0) return 1;
50   if (exponent < 0) return 1 / power(base, -exponent);
51
52   let result = 1;
53   for (let i = 0; i < exponent; i++) {
54     result *= base;
55   }
56   return result;
57 }
58
59 module.exports = {
60   factorial,
61   fibonacci,
62   isPrime,
63   power
64 };

```

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

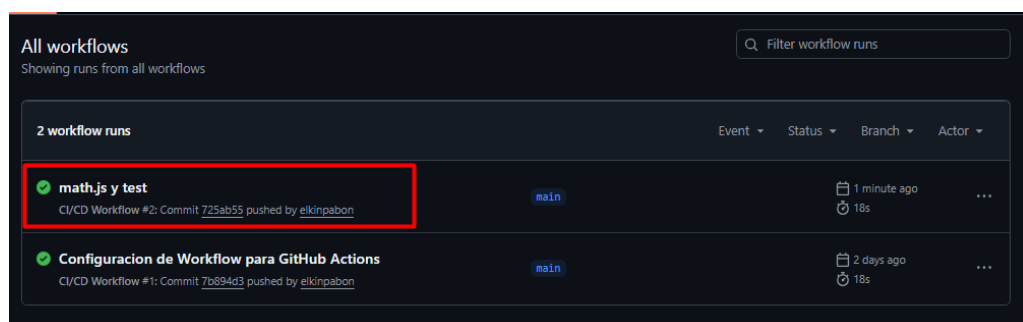
math.test.js

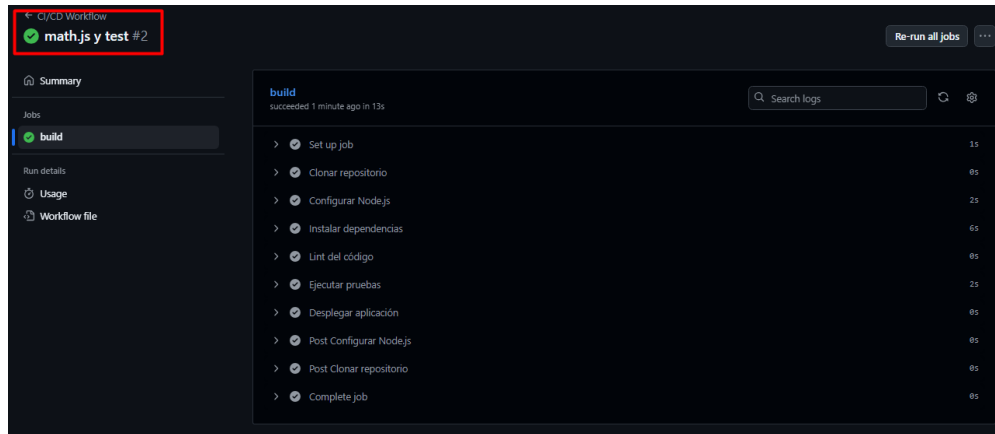
```

1  const { factorial, fibonacci, isPrime, power } = require('./math');
2
3  describe('Math Functions', () => {
4
5    describe('factorial', () => {
6      test('factorial de 0 debe ser 1', () => {
7        expect(factorial(0)).toBe(1);
8      });
9
10     test('factorial de 1 debe ser 1', () => {
11       expect(factorial(1)).toBe(1);
12     });
13
14     test('factorial de números positivos', () => {
15       expect(factorial(3)).toBe(6);
16       expect(factorial(4)).toBe(24);
17       expect(factorial(5)).toBe(120);
18       expect(factorial(6)).toBe(720);
19     });
20
21     test('factorial de número negativo debe lanzar error', () => {
22       expect(() => factorial(-1)).toThrow('El factorial no está definido para números negativos');
23       expect(() => factorial(-5)).toThrow('El factorial no está definido para números negativos');
24     });
25   });
26
27   describe('fibonacci', () => {
28     test('fibonacci de 0 debe ser 0', () => {
29       expect(fibonacci(0)).toBe(0);
30     });
31
32     test('fibonacci de 1 debe ser 1', () => {
33       expect(fibonacci(1)).toBe(1);
34     });
35
36     test('secuencia de fibonacci correcta', () => {
37       expect(fibonacci(2)).toBe(1);
38       expect(fibonacci(3)).toBe(2);
39     });
40   });
41 });

```

- c. Asegurarse de que GitHub Actions ejecute todas las pruebas con éxito.

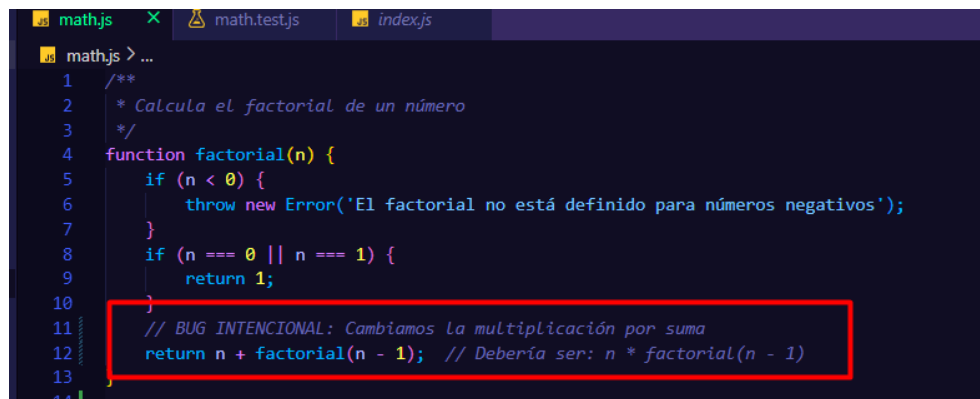




The screenshot shows the GitHub Actions interface for a workflow named "mathjs y test #2". The workflow is in a "Completed" state. The "Summary" tab is selected, showing a list of jobs: "build", "test", and "deploy". The "build" job is highlighted, showing its details. The job "build" succeeded 1 minute ago in 13s. The steps listed are: Set up job, Clonar repositorio, Configurar Node.js, Instalar dependencias, Lint del código, Ejecutar pruebas, Desplegar aplicación, Post Configurar Node.js, Post Clonar repositorio, and Complete job.

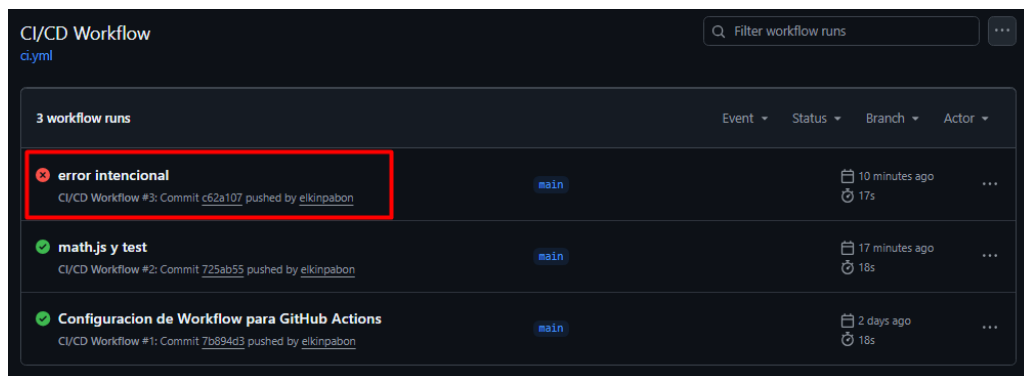
2. Provocar un error intencional y corregirlo

- a. Modificar cualquier función o el test de alguna de ellas para que falle intencionalmente.

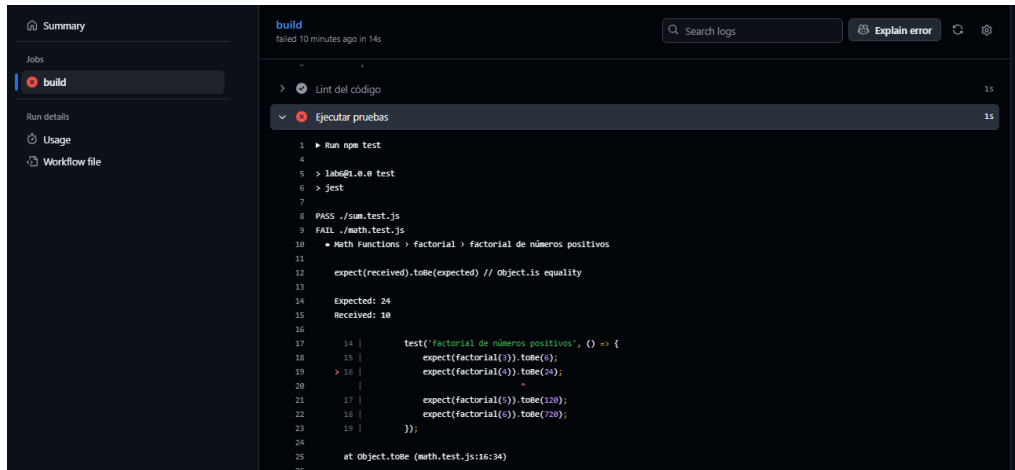


```
1  /**  
2   * Calcula el factorial de un número  
3   */  
4  function factorial(n) {  
5      if (n < 0) {  
6          throw new Error('El factorial no está definido para números negativos');  
7      }  
8      if (n === 0 || n === 1) {  
9          return 1;  
10     }  
11     // BUG INTENCIONAL: Cambiamos la multiplicación por suma  
12     return n + factorial(n - 1); // Debería ser: n * factorial(n - 1)  
13 }  
14
```

- b. Subir los cambios y verificar que el flujo CI falla.



The screenshot shows the GitHub Actions interface for a workflow named "ci.yml". The workflow is in a "Failed" state. The "Summary" tab is selected, showing a list of workflow runs. The first run, "error intencional", is highlighted with a red box. It failed 10 minutes ago in 17s. The second run, "mathjs y test", is in a "Completed" state. The third run, "Configuracion de Workflow para GitHub Actions", is in a "Completed" state.



Summary

Jobs

- build

Run details

Usage

Workflow file

build
failed 10 minutes ago in 14s

Search logs

Explain error

Lint del código

Ejecutar pruebas

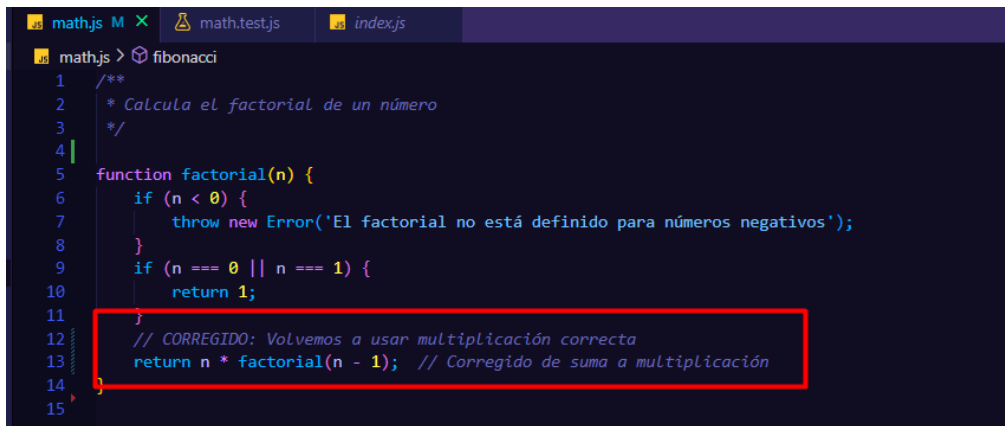
```

1 ▶ Run npm test
4
5 > lab@0.0.0 test
6 > jest
7
8 PASS ./sum.test.js
9 FAIL ./math.test.js
10   ● Math Functions > factorial > factorial de números positivos
11     expect(received).toBe(expected) // Object.is equality
12
13     Expected: 24
14     Received: 18
15
16   14 |     test('factorial de números positivos', () => {
17   15 |       expect(factorial(3)).toBe(6);
18   16 |       expect(factorial(4)).toBe(24);
19   17 |       expect(factorial(5)).toBe(120);
20   18 |       expect(factorial(6)).toBe(720);
21   19 |     });
22
23   at Object.toBe (math.test.js:16:34)
  
```

c. Corregir el error y volver a subir.

```

• PS C:\Users\Elkin Andres\Desktop\LAB6> git add -A
• PS C:\Users\Elkin Andres\Desktop\LAB6> git commit -m "correccion error"
[main e296ead] correccion error
1 file changed, 3 insertions(+), 3 deletions(-)
• PS C:\Users\Elkin Andres\Desktop\LAB6> git push origin main
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 16 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 405 bytes | 405.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/elkinpabon/Laboratorio6-Pruebas-de-Software.git
c62a107..e296ead main -> main
  
```



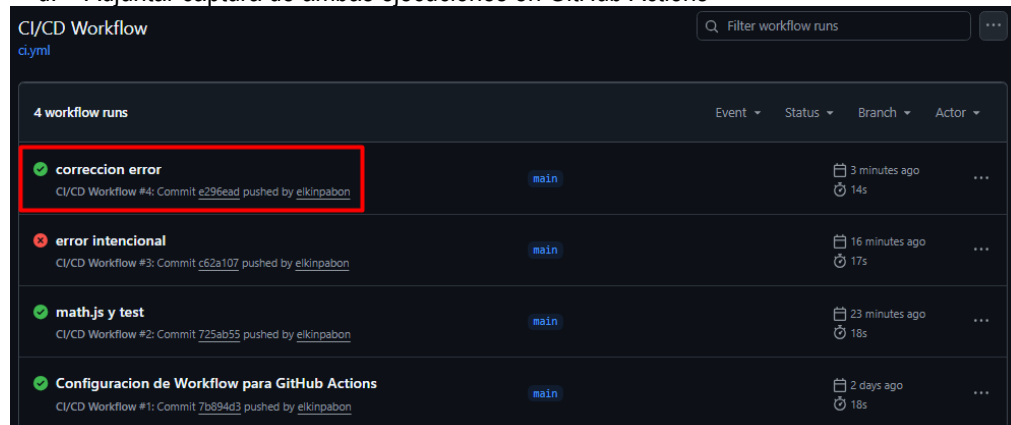
math.js M x math.test.js index.js

math.js > fibonacci

```

1 /**
2  * Calcula el factorial de un número
3  */
4
5 function factorial(n) {
6   if (n < 0) {
7     throw new Error('El factorial no está definido para números negativos');
8   }
9   if (n === 0 || n === 1) {
10    return 1;
11  }
12  // CORREGIDO: Volvemos a usar multiplicación correcta
13  return n * factorial(n - 1); // Corregido de suma a multiplicación
14 }
15
  
```

d. Adjuntar captura de ambas ejecuciones en GitHub Actions

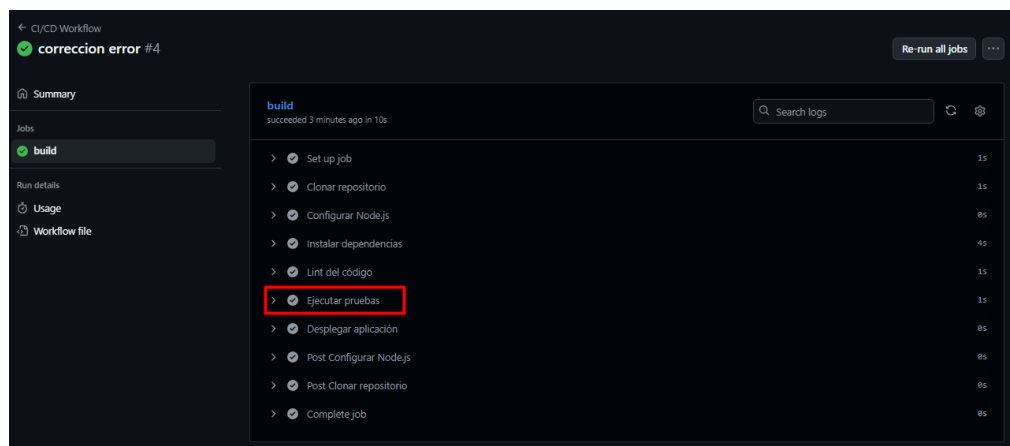


CI/CD Workflow

Filter workflow runs

4 workflow runs

Event	Status	Branch	Actor	Time	More
correccion error	Success	main	elkinpabon	3 minutes ago	...
error intencional	Failure	main	elkinpabon	16 minutes ago	...
math.js y test	Success	main	elkinpabon	23 minutes ago	...
Configuracion de Workflow para GitHub Actions	Success	main	elkinpabon	2 days ago	...



6. CONCLUSIONES:

- La implementación de un flujo de Integración Continua mediante GitHub Actions demostró ser una herramienta eficaz para automatizar tareas críticas del ciclo de desarrollo, como la instalación de dependencias, la ejecución de pruebas unitarias y el análisis estático de código, lo que permitió reducir significativamente la intervención manual y minimizar el riesgo de errores humanos.
- La integración de Jest como framework de pruebas permitió validar el correcto funcionamiento de las funciones implementadas en cada actualización del repositorio. Esto aseguró que las nuevas funcionalidades no afectaran negativamente las ya existentes, reforzando la estabilidad y confiabilidad del sistema.
- El uso de ESLint como herramienta de análisis estático contribuyó a mantener un código consistente y libre de errores comunes, promoviendo el cumplimiento de estándares de calidad y buenas prácticas de programación. Esto repercutió directamente en la mantenibilidad del software a largo plazo.
- La práctica permitió comprender la importancia de incorporar CI en los proyectos de desarrollo moderno, sentando las bases para una posterior implementación de Entrega Continua (CD) que optimizaría aún más los tiempos de despliegue y la calidad final del producto.

7. RECOMENDACIONES:

- Mantener actualizados los archivos de configuración del flujo de CI y las dependencias del proyecto para garantizar compatibilidad con nuevas versiones de librerías y herramientas, evitando fallos inesperados durante la ejecución de los workflows.
- Ampliar la cobertura de pruebas unitarias incluyendo casos de prueba que contemplen escenarios límite, datos atípicos y condiciones de error, con el fin de robustecer la validación del sistema y anticipar fallos potenciales en producción.
- Integrar notificaciones automáticas en GitHub Actions, como mensajes a correo electrónico o plataformas de mensajería (Slack, Teams), para recibir alertas inmediatas ante fallos, permitiendo una respuesta más rápida y eficiente del equipo de desarrollo.
- Considerar la incorporación progresiva de prácticas de Entrega Continua (CD) y despliegue automatizado, vinculando el flujo actual de CI con entornos de staging o producción controlados, para reducir el tiempo entre el desarrollo y la entrega final del software.

8. BIBLIOGRAFÍA:

- Fowler, M. 2020. Continuous Integration. ThoughtWorks, Addison-Wesley Professional. Disponible en: <https://martinfowler.com/articles/continuousIntegration.html>. Fecha de consulta: 09 de agosto de 2025.
- Kim, G., Humble, J., Debois, P., Willis, J. 2021. The DevOps Handbook. IT Revolution Press. Páginas 45–87. Fecha de consulta: 09 de agosto de 2025.
- Turnbull, J. 2014. The Node Beginner Book. Leanpub. Disponible en: <https://leanpub.com/nodebeginner>. Fecha de consulta: 09 de agosto de 2025.

- Raj, P., Seybold, C. 2018. Practical Node.js: Building Real-World Scalable Web Apps. Apress. Páginas 112–134.
Fecha de consulta: 09 de agosto de 2025.