

GUIA GIT ACTIONS

Objetivo

- Familiarizarse con el concepto DevOps.
- Practicar los comandos básicos de git.
- Definir como trabajar con ramas.
- Incorporar la herramienta *GitActions* al proyecto de la asignatura para implementar la integración continua (CI).

Requisitos

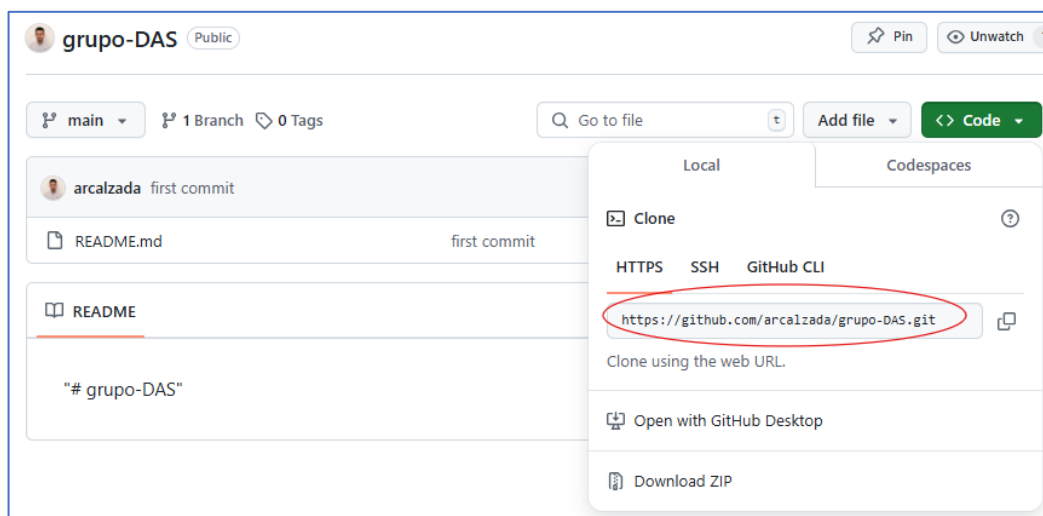
- Tener una cuenta en GitHub y haber creado el repositorio del grupo.
- Haberse inscrito como grupo en el apartado Proyectos del Moodle de la asignatura.
- Tener instalado Visual Studio Code.

Enunciado

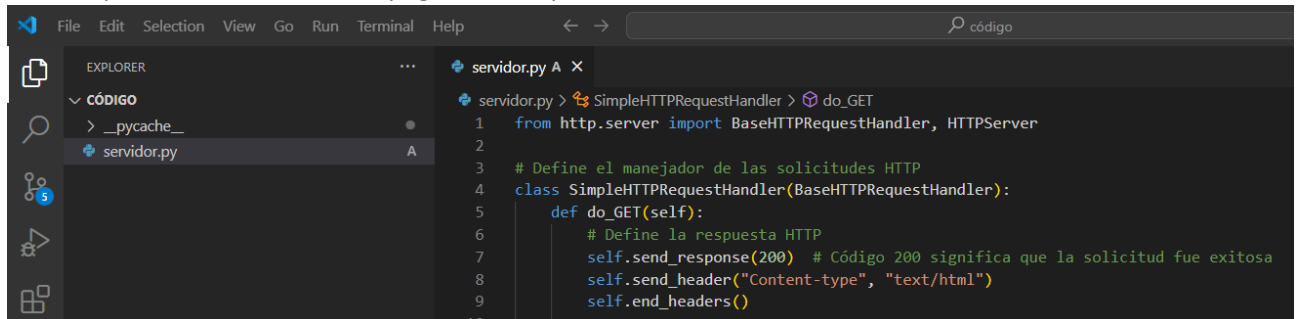
Primer paso: primer commit

1. Cree una carpeta en su ordenador. Será la carpeta en la que guardaremos el código del proyecto.
2. Acceda a la consola terminal y diríjase a la ruta de la carpeta del paso anterior.
 - Puede usar el comando "cd" para desplazarse por las rutas del ordenador. Por ejemplo, **cd "C:\Users\usuario \Desarrollo de aplicaciones y servicios 2025\proyecto"**.
3. En la consola, introduzca los siguientes comandos:
 - a) Inicializar repositorio: **git init**
 - b) Añadir todos los archivos al área de preparación: **git add .**
 - c) Realizar un *commit*: **git commit -m "Primer commit"**
 - d) Añadir un remoto: **git remote add origin URL_DEL_REPOSITORIO**

La URL del repositorio la obtenemos desde "<> Code", HTTPS:



4. A continuación, abra Visual Studio Code y abra la carpeta creada en el paso 1.
 - File > Open File
5. Copie el fichero “servidor.py”, facilitado justo a esta guía, a la carpeta del proyecto. El fichero contiene una versión muy básica de un manejador de solicitudes HTTP que responde a peticiones GET con una página HTML personalizada.



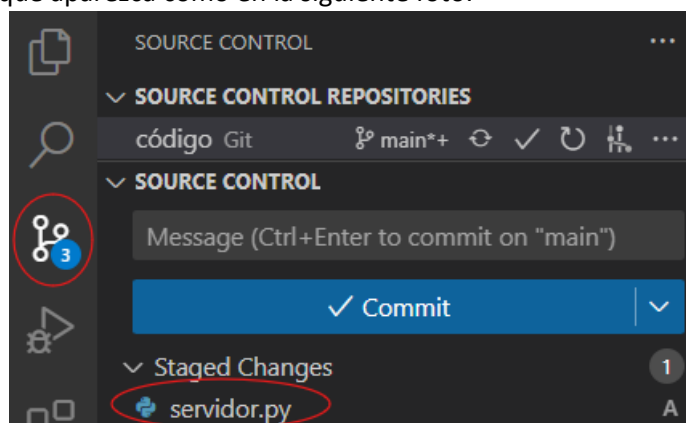
```
servidor.py A X
servidor.py > SimpleHTTPRequestHandler > do_GET
1 from http.server import BaseHTTPRequestHandler, HTTPServer
2
3 # Define el manejador de las solicitudes HTTP
4 class SimpleHTTPRequestHandler(BaseHTTPRequestHandler):
5     def do_GET(self):
6         # Define la respuesta HTTP
7         self.send_response(200) # Código 200 significa que la solicitud fue exitosa
8         self.send_header("Content-type", "text/html")
9         self.end_headers()
10
```

6. Para ejecutar este programa, abra una nueva ventana de terminal seleccionando:
 - Terminal > New Terminal
7. Por defecto, se abrirá una ventana en la parte inferior de la pantalla de Visual Studio Code. Ejecute el siguiente comando:
 - **python servidor.py**
 - Acceda a la siguiente url: <http://localhost:8000/> y debería ver:

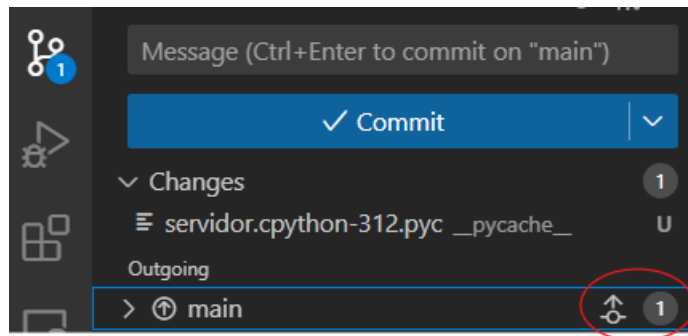
¡Hola desde un servidor Python!

Este es un servidor simple que responde a solicitudes GET.

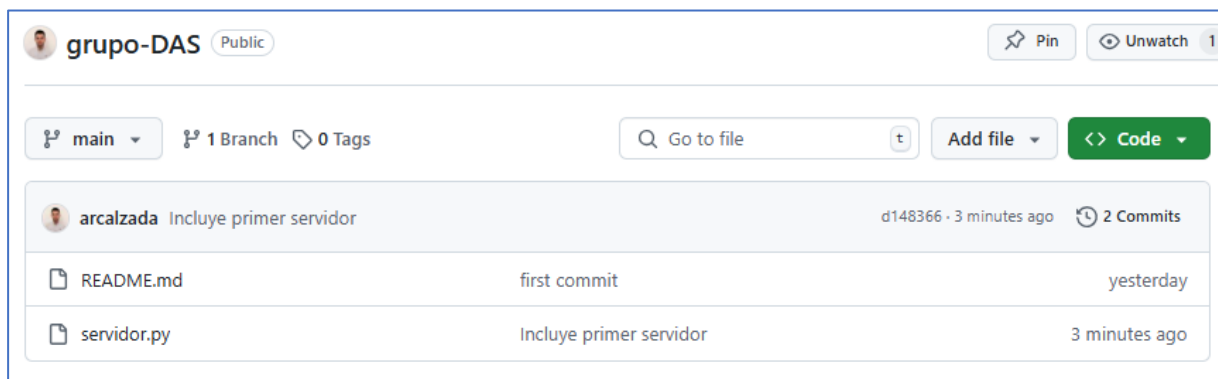
- “Control + C” para detener el programa.
8. Ahora, deberá subir este fichero a su repositorio de grupo. Aunque puede utilizar los comandos del paso 3 b) -> d), se recomienda usar Visual Studio Code. Para ello, seleccione el tercer botón de la barra lateral. A continuación, en el apartado *Change*, incluya al fichero servidor.py para que aparezca como en la siguiente foto.



9. Proceda escribiendo un mensaje para el commit en el campo de texto. Por ejemplo: “Incluye primer servidor”. Tras ello, clique en el botón azul de “Commit”.
10. Ahora que ya se ha creado un commit con el cambio de código, seleccione “push” para subir de su repositorio local al remoto.

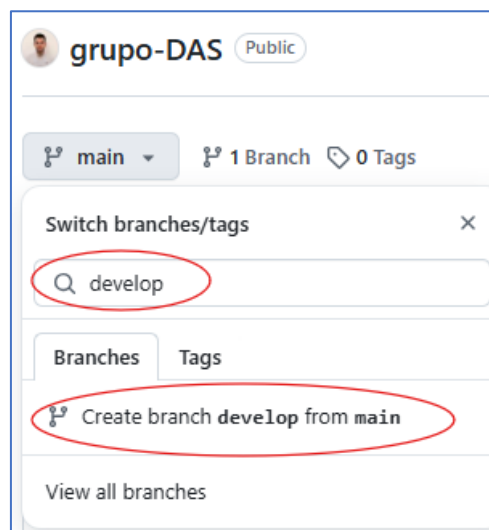


11. Compruebe que el fichero servidor.py ya aparece en su repositorio de grupo:

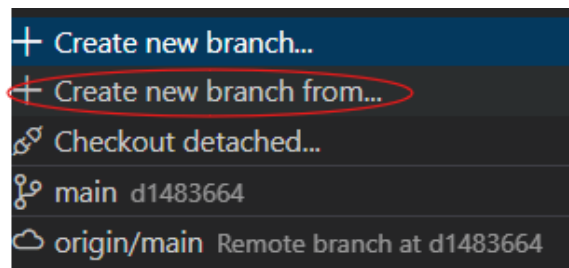
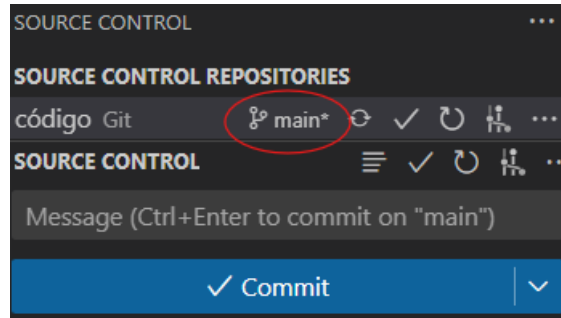


Segundo paso: crear una nueva rama (develop)

- El siguiente paso es crear una rama llamada "develop". En esta rama se acumularán los desarrollos de las nuevas funcionalidades hasta que estén maduros y se pueda crear una nueva versión. La rama "develop" deberá crearse como copia de "main". Hay diferentes formas de crear una rama:
 - Desde GitHub. Seleccione la rama actual ("main") y se mostrará un desplegable. Si en el buscador de ramas de repositorio se incluye un nombre (como "develop") que no existe, te ofrecerá la posibilidad de crear una rama con ese nombre, usando como base la rama sobre la que esté.



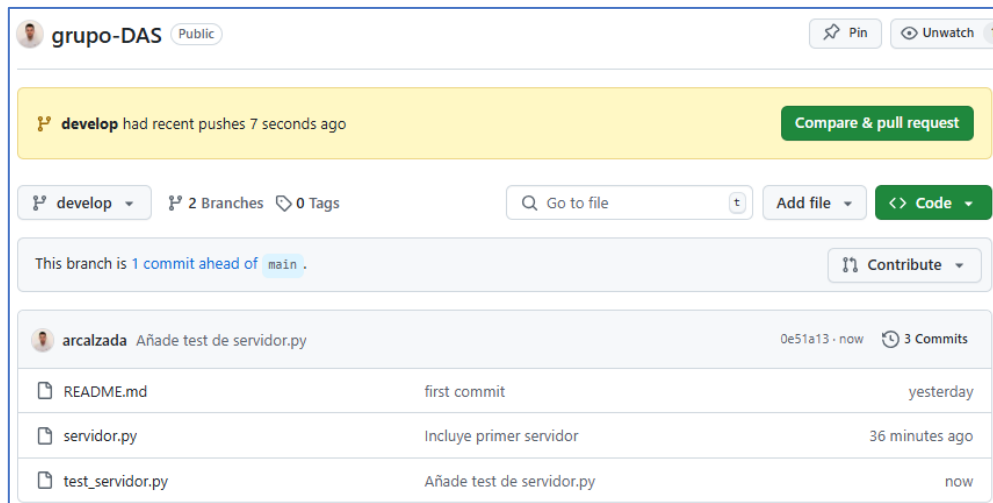
- Desde Visual Studio Code. Si selecciona “main” dentro de la opción de Git de Visual Studio Code, se le ofrece la opción de “Create new Branch from”.



- Por terminal de comandos: **git checkout -b develop**
2. Una vez ubicado en la nueva rama (“develop”), deberá copiar el fichero “test_servidor.py” facilitado con esta guía a su proyecto.
 3. Para lanzar el test, ejecute los siguientes comandos en el terminal de Visual Studio Code:
 - **pip install requests** (solo la primera vez para descargar las dependencias)
 - **python test_servidor.py**
 - El resultado debería ser:

```
go> python test_servidor.py
>>
127.0.0.1 - - [20/Jan/2025 09:17:25] "GET / HTTP/1.1" 200 -
.
-----
Ran 1 test in 2.550s
OK
```

4. Finalmente, deberá subir los cambios a la rama “develop”. Para ello, puede repetir los pasos del 8 al 10 de la primera sección.
 - Mensaje del commit “Añade test de servidor.py”.



Ejercicio

Para poner en práctica lo aprendido hasta el momento, se le pide que realicen dos evolutivos. El miembro del equipo de mayor edad realizará la primera tarea; y el otro, la segunda.

- Crear la rama “feature/incluir-info-grupo” desde la rama “develop”. En esta rama se deberá modificar el html del fichero “servidor.py”. En concreto, se pide incluir una etiqueta con el nombre del grupo:
 - `<p>Web del “nombre de su grupo”.</p>` **// NO INCLUIR ACENTOS**

Hola desde un servidor Python

Este es un servidor simple que responde a solicitudes GET.

Web del grupo-DAS.

Además, se deberá modificar el fichero “test_servidor.py” para que valide que la página HTML que devuelve el servidor incluye la nueva etiqueta.

- Crear la rama “feature/insertar-imagen” desde la rama “develop”. En esta ocasión se añadirá la etiqueta de una imagen al html del fichero “servidor.py”. La imagen queda a gusto del equipo.
 - ``
 - Ejemplo de url: https://cdn.prod.website-files.com/64fa82cbdeed167ebaefef84/64fa868eccc183a3dd76ab4c_603ec5023c4ad8fde1783428_lj2FnlaQX3wZEgCdfWmynR3kTFRlelaf-BXa21868XGfGWQiBv5FISkffcRaUhXrgoKiMX9FiLDGZ2jxwKGdt_vTyGUVHlqcm9uMjUBNQrgltzfgD3TulNwNixxWI2R3ay9vcAc7.jpeg

Hola desde un servidor Python

Este es un servidor simple que responde a solicitudes GET.

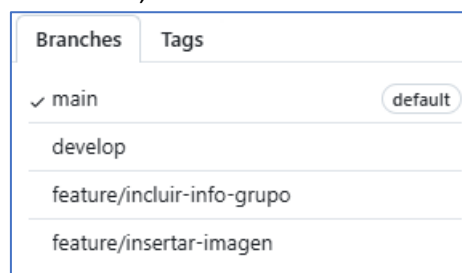


Además, se deberá modificar el fichero “test_servidor.py” para que valide que la página html que devuelve el servidor incluye la nueva etiqueta.

Enunciado

Tercer paso: mergear ramas

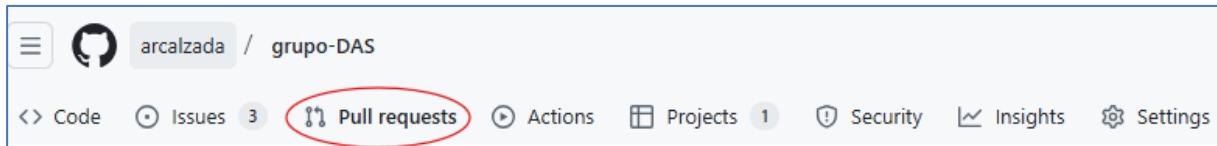
1. Una vez finalizado el ejercicio anterior, debería tener 4 ramas:



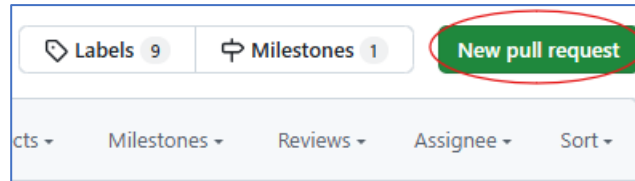
2. Es hora de añadir los dos evolutivos del ejercicio a la rama “develop”. Se realiza mediante el *mergeo* (o fusión) de ramas. Se quiere trasladar los cambios aplicados en las ramas “features” a la rama de “develop”. **Es importante el sentido de la fusión.**

Se van a ofrecer dos formas de hacer el *merge*. Se pide que cada miembro del equipo haga una, según la tarea que implementó.

- Desde GitHub.
 - a) Diríjase a la sección “Pull requests”.



b) Seleccione “New pull request”.

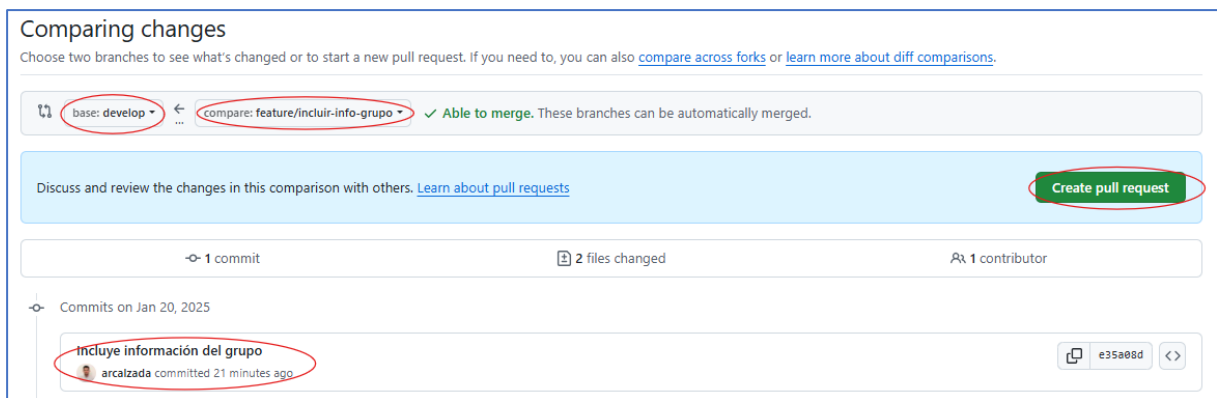


c) Indique:

- Base: “develop”
- Compare: “feature/incluir-info-grupo”

Saldrán los mensajes de commits que se han realizado en la rama feature.

Finalmente, pulse “Create pull request”.




d) Posteriormente, se deberá añadir información que permita identificar fácilmente los cambios que se quieren incluir en la rama “develop”.

e) Se debe asignar como “Reviewers” y “Assignee” al otro miembro del grupo.

Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#). [Learn more about diff comparisons here](#).

base: develop
←
compare: feature/incluir-info-grupo
✓ Able to merge. These branches can be automatically merged.


Add a title

Incluye información del grupo

Reviewers
No reviews

Assignees
No one—assign yourself

Labels
None yet

Projects
None yet

Milestone
No milestone

Development
Use [Closing keywords](#) in the description to automatically close issues

Add a description

Write
Preview

Se incluye información del grupo en el html principal

Markdown is supported
Paste, drop, or click to add files

Create pull request


Remember, contributions to this repository should follow our [GitHub Community Guidelines](#).

f) Con esto, ya tendría creado su primera Pull Request.

Incluye información del grupo #4

Open
arcalzada wants to merge 1 commit into develop from feature/incluir-info-grupo

Conversation 0 Commits 1 Checks 0 Files changed 2 +2 -0


arc alzada commented now

Se incluye información del grupo en el html principal

Incluye información del grupo
e35a88d

arc alzada self-assigned this now

Require approval from specific reviewers before merging
Rulesets ensure specific people approve pull requests before they're merged.
Add rule

This branch has no conflicts with the base branch
Merging can be performed automatically.

Merge pull request
You can also [open this in GitHub Desktop](#) or view [command line instructions](#).

Reviewers
No reviews
Still in progress? [Convert to draft](#)

Assignees
arc alzada

Labels
None yet

Projects
None yet

Milestone
No milestone

Development
Successfully merging this pull request may close these issues.

g) Si selecciona "File Changed", se muestra las modificaciones de los ficheros que se han realizado en la rama de "feature".

Incluye información del grupo #4

Open | arcalzada wants to merge 1 commit into develop from feature/incluir-info-grupo

Conversation 0 | Commits 1 | Checks 0 | Files changed 2

Changes from all commits | File filter | Conversations

Filter changed files

servidor.py | test_servidor.py

```

@@ -19,6 +19,7 @@ def do_GET(self):
    <body>
    <h1>Hola desde un servidor Python</h1>
    <p>Este es un servidor simple que responde a solicitudes GET.</p>
+   <p>web del grupo-DAS.</p>
    </body>
    </html>"""

```

```

@@ -35,6 +35,7 @@ def test_get_request(self):
    # verifica que el contenido HTML esperado esté en la respuesta
    self.assertIn("<h1>Hola desde un servidor Python</h1>", response.text)
    self.assertIn("<p>Este es un servidor simple que responde a solicitudes GET.</p>", response.text)
+   self.assertIn("<p>web del grupo-DAS.</p>", response.text)

```

if __name__ == '__main__':

h) El objetivo es que el compañero pueda revisar el código. Para dejar consejos de mejora y/o aprobar el cambio, use las opciones de “Review changes”.

i) Cuando la revisión haya finalizado, ya se podrá mergear:

Require approval from specific reviewers before merging
Rulesets ensure specific people approve pull requests before they're merged. Add rule X

✓ This branch has no conflicts with the base branch
Merging can be performed automatically.

Merge pull request You can also [open this in GitHub Desktop](#) or view [command line instructions](#).

- Desde terminal de comando.
 - a) Cambia a la rama “develop”:
 - **git checkout develop**
 - b) Asegúrate de que la rama “develop” esté actualizada:
 - **git pull origin develop**
 - c) Luego, realiza el merge de la rama “feature/insertar-imagen” a “develop”:
 - **git merge feature/insertar-imagen**

¡CONFLICTOS!

Un conflicto al mergear dos ramas ocurre cuando Git no puede combinar automáticamente los cambios porque hay discrepancias en las mismas partes de un archivo entre las dos ramas. Esto puede suceder en las siguientes situaciones comunes:

- **Cambios en las mismas líneas de un archivo:** Ambas ramas han modificado las mismas líneas del mismo archivo. Git no sabe cuál cambio aplicar o cómo combinarlos.
 - **Es nuestro caso,** ya la rama “develop” incluye el cambio que se hizo en “feature/incluir-info-grupo” por el cual modificó la línea 22 para agregar un párrafo. Este cambio ahora compite con el de la rama “feature/insertar-imagen”, porque también modificó la línea 22 para incluir la imagen.

- **Un archivo fue modificado en una rama y eliminado en otra.** Git no sabe si debe mantener el archivo, eliminarlo o combinarlo.
- **Un archivo fue modificado en ambas ramas de manera incompatible.** Si ambas ramas realizaron modificaciones distintas y no relacionadas, pero afectan el mismo contenido de un archivo, Git no puede resolver cuál versión debería prevalecer.
- **Reordenamiento o renombramiento de archivos.** Si una rama renombró o movió un archivo y la otra lo modificó, Git puede tener problemas para saber cómo integrarlos.

Por eso, en el editor de Visual Studio Code, le aparecerá en el fichero “servidor.py”:

```
<body>
  <h1>Hola desde un servidor Python</h1>
  <p>Este es un servidor simple que responde a solicitudes GET.</p>
Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
<<<<<< HEAD (Current Change)
  <p>Web del grupo-DAS.</p>
=====
  <img src='https://cdn.prod.website-files.com/64fa82cbdeed167ebaefef8
>>>>>> feature/insertar-imagen (Incoming Change)
  </body>
</html>"""
```

- HEAD, en verde, indica la rama actúa. En este caso, “develop”. Muestra que en esa línea 20, ya tiene la etiqueta <p>.
- “feature/insertar-imagen”, en azul, muestra que tiene la etiqueta .

Para resolverlo, basta con borrar las líneas y fusionar las versiones de la siguiente manera:

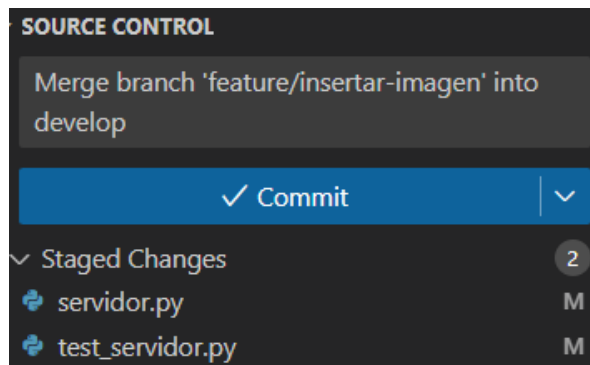
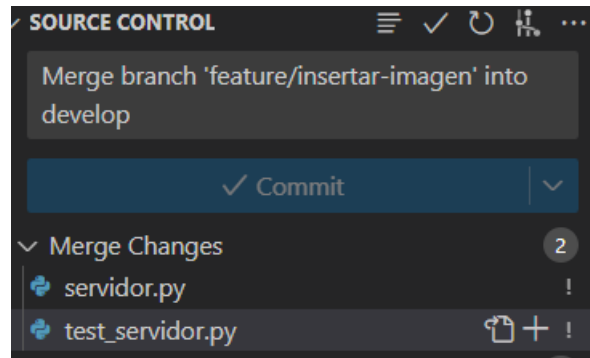
```
<body>
  <h1>Hola desde un servidor Python</h1>
  <p>Este es un servidor simple que responde a solicitudes GET.</p>
  <p>Web del grupo-DAS.</p>
  <img src='https://cdn.prod.website-files.com/64fa82cbdeed167ebaefef84/64fa868eccc183a
</body>
</html>"""
```

Proseguimos realizando el mismo arreglo para el fichero “test_servidor.py”:

```
Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
<<<<<< HEAD (Current Change)
  self.assertIn("<p>Web del grupo-DAS.</p>", response.text)
=====
  self.assertIn("<img src='https://cdn.prod.website-files.com/64fa82cbdeed167ebaefef84/64fa868eccc183a
>>>>>> feature/insertar-imagen (Incoming Change)
```

```
# Verifica que el contenido HTML esperado esté en la respuesta
self.assertIn("<h1>Hola desde un servidor Python</h1>", response.text)
self.assertIn("<p>Este es un servidor simple que responde a solicitudes GET.</p>", response.text)
self.assertIn("<p>Web del grupo-DAS.</p>", response.text)
self.assertIn("<img src='https://cdn.prod.website-files.com/64fa82cbdeed167ebaefef84/64fa868eccc183a
```

Una vez finalizado la resolución de conflictos en el editor, es necesario que añada la versión arreglada y la suba a su repositorio:



Seleccione “Commit and Push” del desplegable de opciones del botón azul de “Commit”.

- Tras finalizar el merge de las dos ramas "feature", compruebe que la rama “develop” incluye todos los evolutivos. Pruebe a ejecutarlo en local. Se deberían ver los dos evolutivos a la vez.

Hola desde un servidor Python

Este es un servidor simple que responde a solicitudes GET.

Web del grupo-DAS.



Cuarto paso: configurar CI

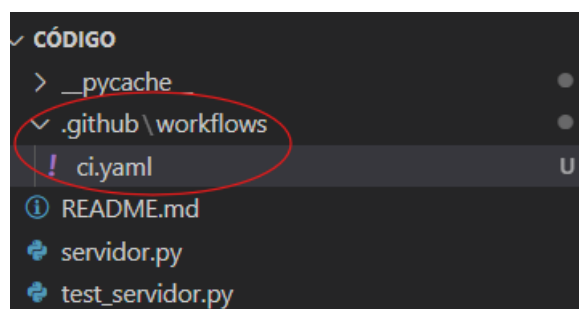
GitHub Actions

Es una plataforma de automatización de flujos de trabajo y CI/CD (Integración Continua y Despliegue Continuo) que permite a los desarrolladores automatizar tareas dentro de su repositorio de GitHub.

Con GitHub Actions, puedes configurar y ejecutar flujos de trabajo personalizados en respuesta a eventos específicos, como commits, pull requests, y lanzamientos. Estos flujos de trabajo pueden incluir compilaciones, pruebas, despliegues y muchas otras tareas automatizadas, mejorando la eficiencia y consistencia del desarrollo de software.

En este cuarto paso, configurará GitHub Actions para cada vez que se ejecute un commit se ejecuten las pruebas.

1. Desde la ruta de la carpeta principal (myFirstApiRest), cree la carpeta **“.github/workflows”** y el fichero **ci.yaml**.



2. Añada en el fichero **ci.yaml** el siguiente contenido:

```
name: CI

on:
  push:
    branches:
      - '*'
  pull_request:
    branches:
      - '*'

jobs:
  test:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout código
        uses: actions/checkout@v4

      - name: Configurar Python
        uses: actions/setup-python@v4
        with:
          python-version: 3.x

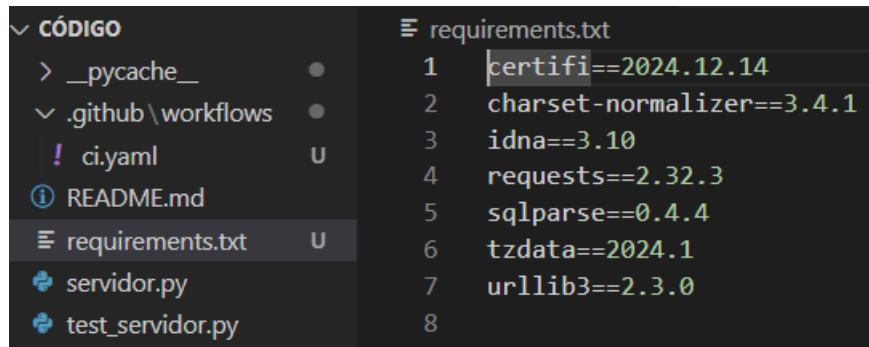
      - name: Instalar dependencias
        run: |
          python -m pip install --upgrade pip
          pip install -r requirements.txt

      - name: Ejecutar pruebas
        run: |
          python test_servidor.py
```

- Donde **on** configura cuándo se va a ejecutar el workflow. En este caso, con las acciones de subir cambios (**push**) y mergear los cambios entre ramos (**pull requests**). Además, se hará para todas las ramas (**branches: ***).
- Se define un job del flujo llamado test. Correrá en un servidor de Ubuntu con la última versión y necesitará un servicio base de datos PostgreSQL para que se ejecuten los test. La configuración será la misma que la que hayamos puesto en los settings, que también será la misma que tengamos en local.
- Se definen los pasos_
 - Actualiza los últimos cambios (checkout).
 - Determina la versión de Python a usar (set-up Python).
 - Baja las dependencias necesarias y definidas en el archivo requirements.txt.
 - Ejecuta las pruebas del servidor.

3. Obtenemos todas las dependencias de nuestro proyecto actual.

- **pip freeze > requirements.txt**



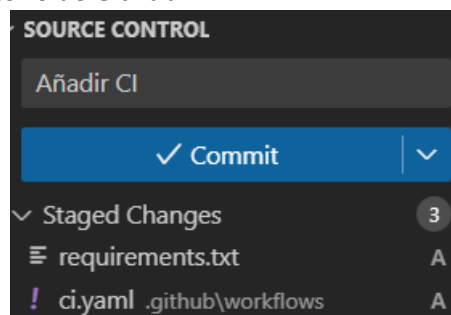
```

CÓDIGO
  > __pycache__
  > .github\workflows
  ! ci.yaml
  i README.md
  requirements.txt
  + servidor.py
  + test_servidor.py

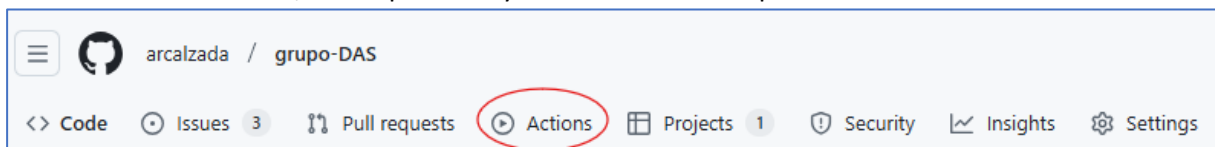
requirements.txt
1 certifi==2024.12.14
2 charset-normalizer==3.4.1
3 idna==3.10
4 requests==2.32.3
5 sqlparse==0.4.4
6 tzdata==2024.1
7 urllib3==2.3.0
8

```

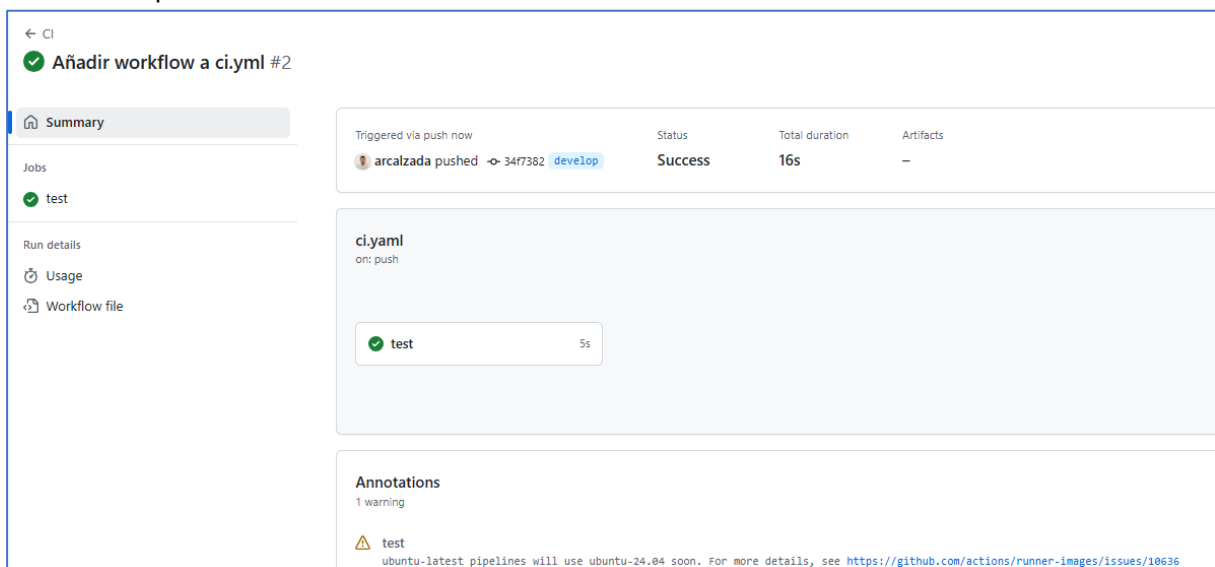
4. Suba los cambios al repositorio de GitHub.



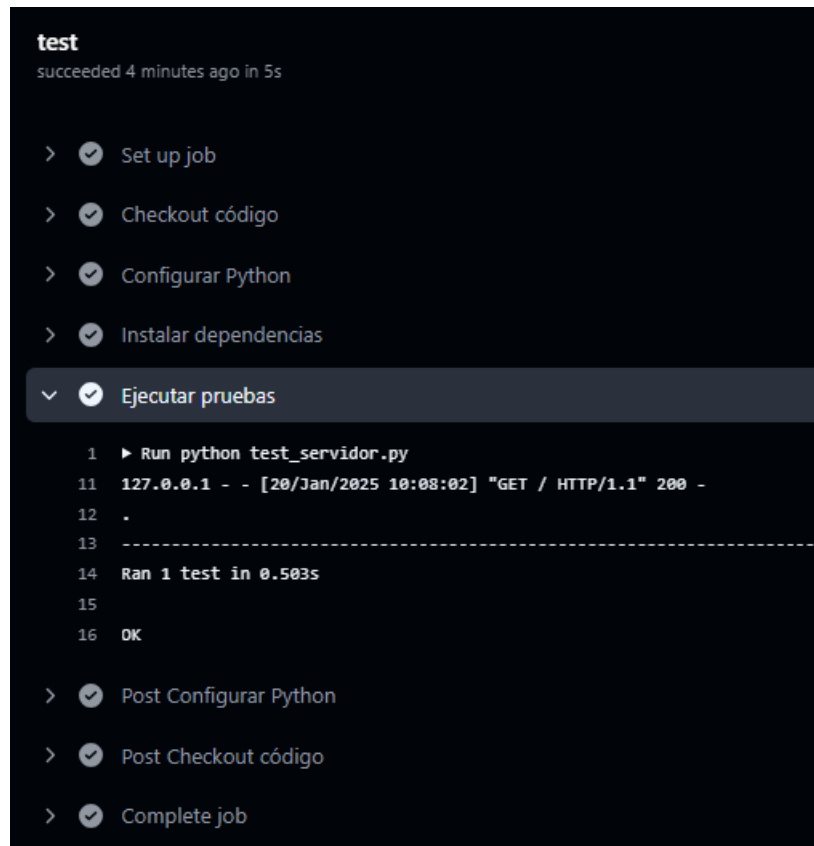
5. Accede a Github, a su repositorio y concretamente a la pestaña de “Actions”.



6. Compruebe el worflow liberado:



7. Si clicas sobre el botón “test” del centro de la imagen anterior puede obtener más información sobre el estado de cada uno de los pasos ejecutados. Será interesante que visualice la fase de “Ejecutar pruebas”.



```
test
succeeded 4 minutes ago in 5s

> ✓ Set up job
> ✓ Checkout código
> ✓ Configurar Python
> ✓ Instalar dependencias
▼ ✓ Ejecutar pruebas
  1 ▶ Run python test_servidor.py
 11 127.0.0.1 - - [20/Jan/2025 10:08:02] "GET / HTTP/1.1" 200 -
 12 .
 13 -----
 14 Ran 1 test in 0.503s
 15
 16 OK
> ✓ Post Configurar Python
> ✓ Post Checkout código
> ✓ Complete job
```

Codacy

Es una plataforma de análisis de código estático que ayuda a mejorar la calidad del código mediante la identificación de problemas de estilo, mantenibilidad, seguridad y mejores prácticas. Aquí están algunas de sus características:


- **Análisis Automático:** Codacy analiza automáticamente cada commit y pull request para encontrar posibles problemas en el código.
- **Soporte Multilenguaje:** Compatible con más de 40 lenguajes de programación, incluyendo JavaScript, Python, Java, PHP, y más.
- **Personalización:** Permite configurar reglas y estándares de calidad personalizados según las necesidades del equipo.
- **Integración con CI/CD:** Se integra fácilmente con herramientas CI/CD como GitHub Actions, GitLab CI, Bitbucket Pipelines, y otros.
- **Métricas y Reportes:** Ofrece informes detallados y métricas sobre:
 - La calidad del código se refiere a qué tan bien escrito y mantenible es el código fuente de un software. Esto incluye varios aspectos, tales como:
 - Legibilidad: Qué tan fácil es de leer y entender el código para otros desarrolladores. Código claro y bien documentado facilita el mantenimiento y la colaboración.

- Estructura: Qué tan bien estructurado está el código en términos de modularidad, separación de responsabilidades, y uso de patrones de diseño.
 - Consistencia: Uso coherente de estilos de codificación, convenciones de nombres, y prácticas de desarrollo a lo largo del proyecto.
 - Eficiencia: Qué tan bien optimizado está el código en términos de rendimiento y uso de recursos.
 - Seguridad: Qué tan bien se maneja la seguridad en el código, incluyendo la prevención de vulnerabilidades comunes como inyecciones de SQL, XSS, etc.
 - Fiabilidad: Qué tan bien se comporta el código bajo diversas condiciones y qué tan probable es que contenga errores.
- Deuda técnica: es un concepto que se refiere a las consecuencias de tomar atajos en el desarrollo de software. Estos atajos pueden llevar a una mayor facilidad y rapidez en el corto plazo, pero generan una carga que debe ser "pagada" en el futuro para mantener el código sostenible y manejable. Algunos ejemplos de deuda técnica incluyen:
 - Código No Refactorizado: Código que no ha sido optimizado o limpiado después de implementaciones rápidas.
 - Falta de Pruebas: Implementaciones sin suficientes pruebas unitarias, de integración o funcionales.
 - Diseño Deficiente: Soluciones rápidas que no siguen buenas prácticas de diseño y arquitectura.
 - Dependencias Obsoletas: Uso de bibliotecas y frameworks desactualizados que eventualmente necesitarán ser actualizados o reemplazados.
 - Documentación Incompleta: Falta de documentación adecuada que dificulta la comprensión y el mantenimiento del código.
 - Cobertura de pruebas: es una métrica que indica qué porcentaje del código fuente está cubierto por pruebas automatizadas. Es una medida de qué tan exhaustivamente se ha probado el software para asegurar su funcionamiento correcto.

8. Visite la página principal de [Codacy](#) e inicie sesión con su cuenta GitHub.

One-click login & signup

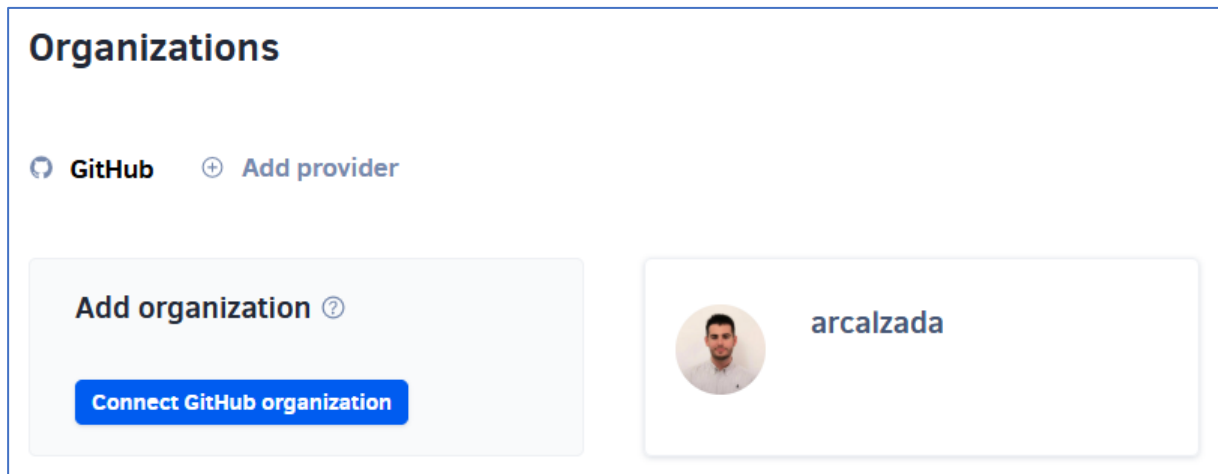
Start for free, no credit card required

 GitHub

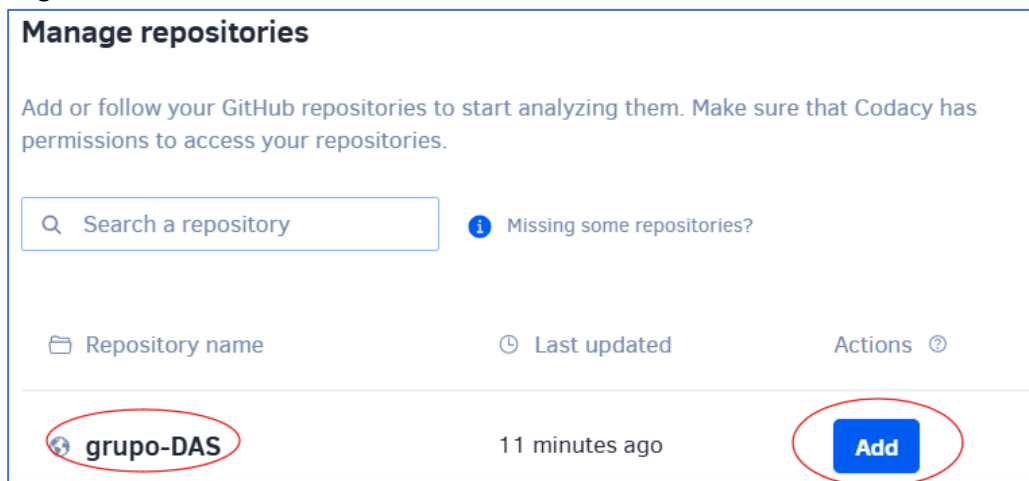
Or Sign up with: [Bitbucket](#) | [Gitlab](#)

By signing up you agree to the [Terms of Service](#) and [Privacy Policy](#).

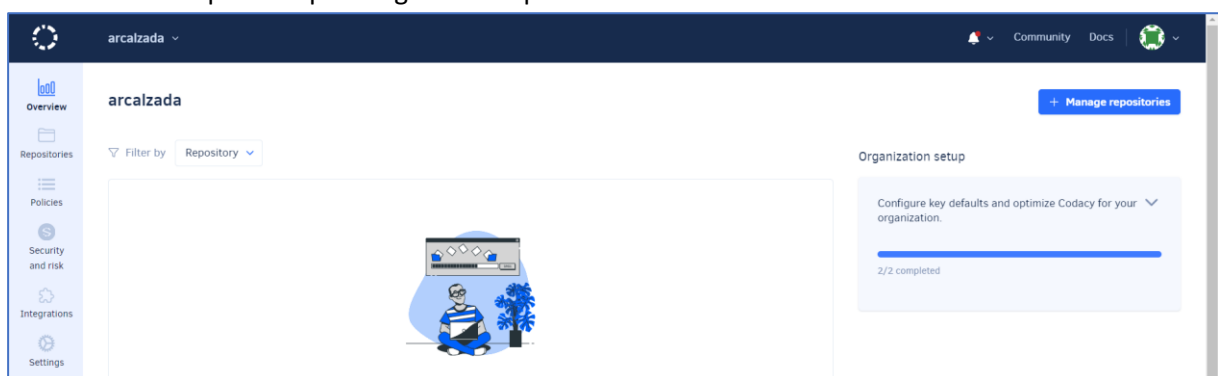
9. Seleccione su usuario.



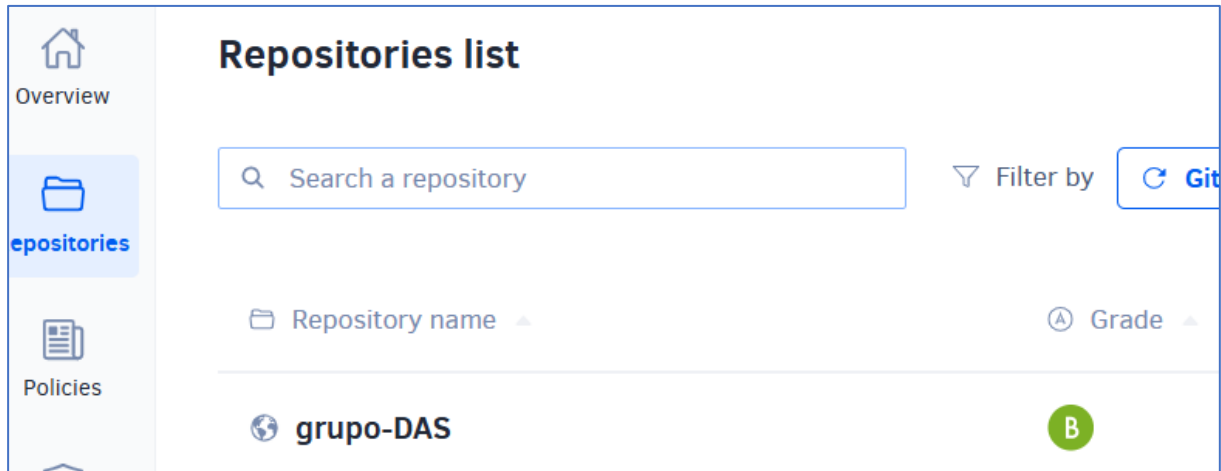
10. En la ventana principal, cliqué sobre “Manage repositories” y escoja el repositorio de esta asignatura.



11. Cierre la pestaña para regresar a la pantalla de inicio.



12. A continuación, navegue hasta “Repositoires” del menú lateral y seleccione el repositorio de su grupo.



Repositories list

Search a repository

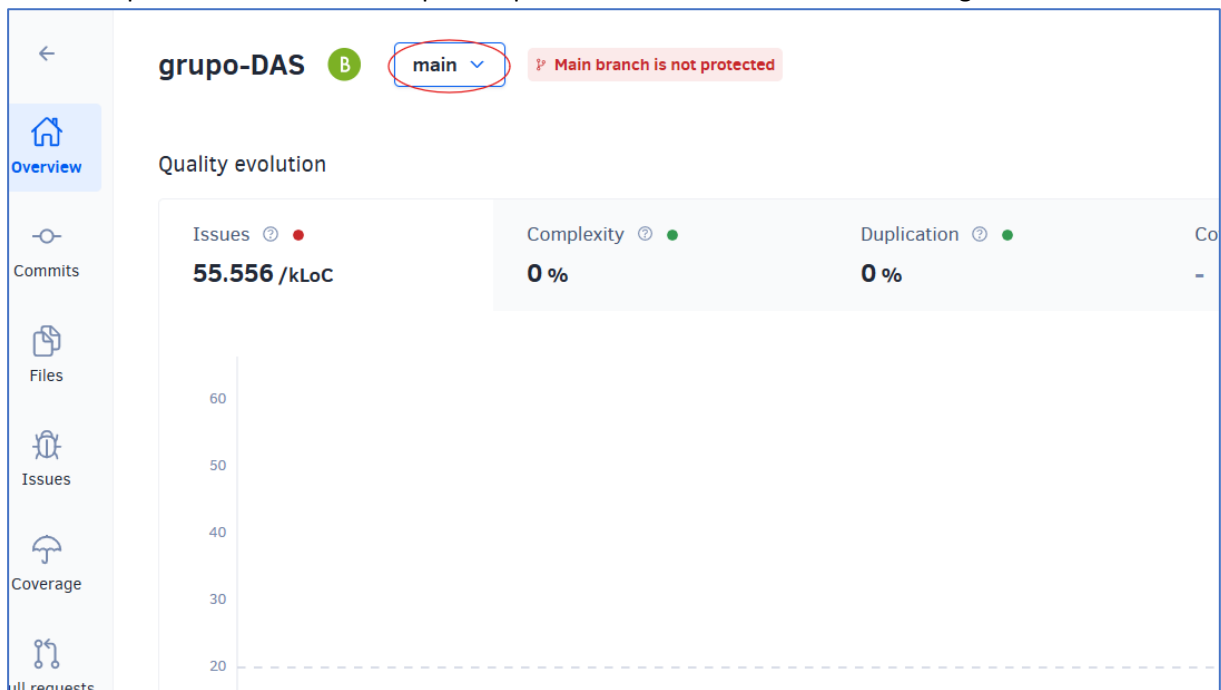
Filter by

Repository name

Grade

grupo-DAS B

13. Aquí encontrará el análisis previo que ha hecho sobre la calidad del código actual.



grupo-DAS B main Main branch is not protected

Quality evolution

Issues 55.556 /kLoC

Complexity 0 %

Duplication 0 %

60

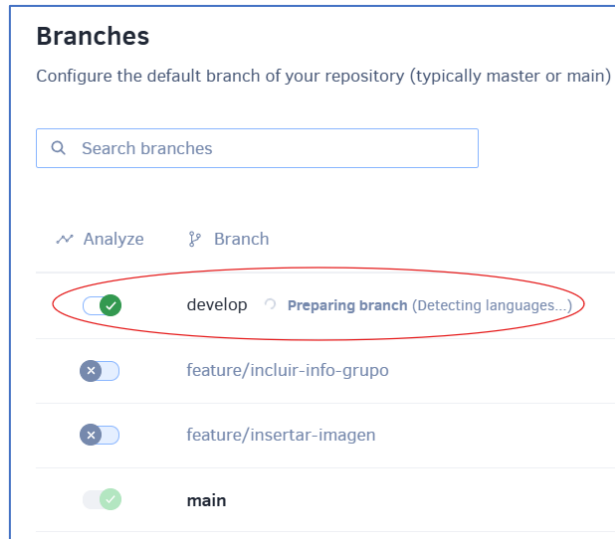
50

40

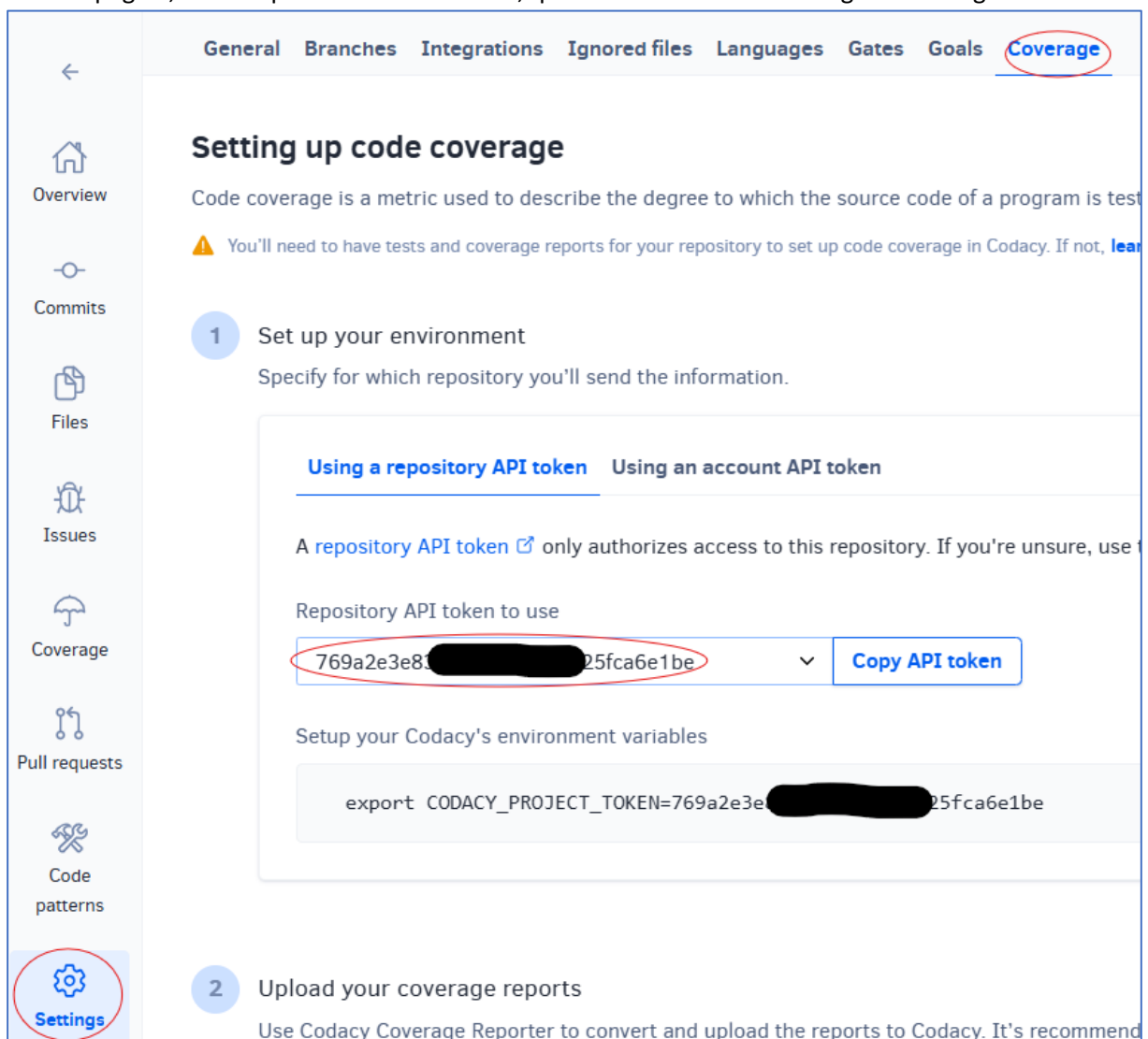
30

20

14. Añada también la rama “develop” ya que será una rama importante en el ciclo de vida del proyecto.



15. Para configurar también la parte de los tests de cobertura, pinche en el botón de “Settings” del navegador de la izquierda. Y una vez en ese menú, pulse en “Coverage”. Una vez en esta página, debe copiar el valor del Token, que es el remarcado en la siguiente imagen.



Setting up code coverage

Code coverage is a metric used to describe the degree to which the source code of a program is tested.

⚠ You'll need to have tests and coverage reports for your repository to set up code coverage in Codacy. If not, [learn more](#).

- Set up your environment**
Specify for which repository you'll send the information.

Using a repository API token | Using an account API token

A [repository API token](#) only authorizes access to this repository. If you're unsure, use the account token.

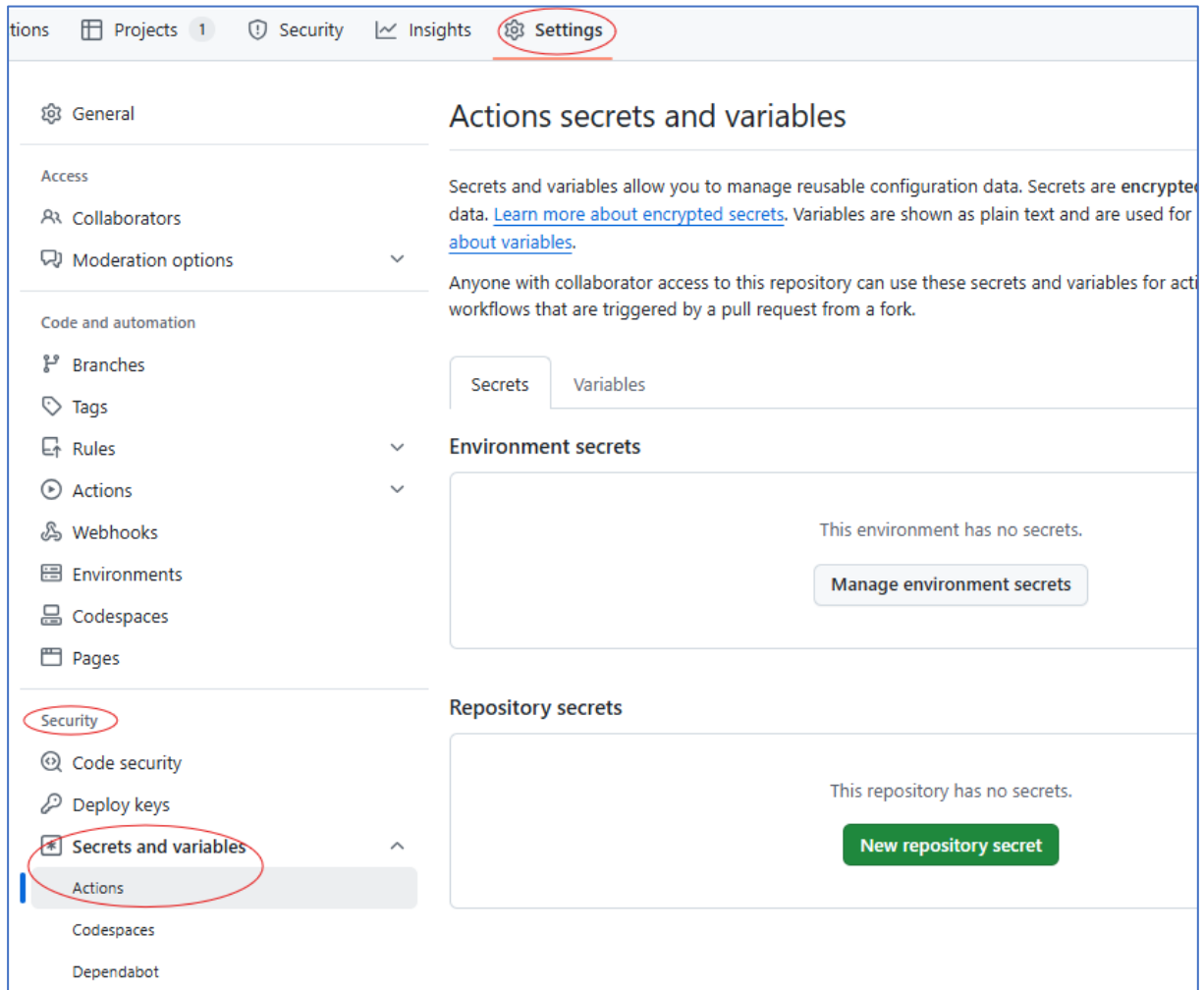
Repository API token to use

769a2e3e81[REDACTED]25fca6e1be Copy API token

Setup your Codacy's environment variables

```
export CODACY_PROJECT_TOKEN=769a2e3e[REDACTED]25fca6e1be
```
- Upload your coverage reports**
Use Codacy Coverage Reporter to convert and upload the reports to Codacy. It's recommended to use the CLI tool.

16. Debido al carácter sensible del token, es necesario guardarlos en un sitio seguro. Afortunadamente, GitHub Actions nos ofrece una solución. Por ello, regrese a la página de su repositorio de GitHub. Acceda a “Settings” del repositorio, y una vez ahí, en el desplegable de la izquierda, navegue a Security > Secrets and variables > Actions.



The screenshot shows the GitHub repository settings page. The top navigation bar includes 'Settings', which is circled in red. The left sidebar shows the 'Security' section selected, with 'Secrets and variables' highlighted. The main content area is titled 'Actions secrets and variables'. It includes a description of secrets and variables, a 'Secrets' tab, and two sections: 'Environment secrets' and 'Repository secrets'. Both sections indicate that there are no secrets currently stored. A 'New repository secret' button is visible in the 'Repository secrets' section.

17. Llegados a este punto, seleccione “New repository secret” con la clave “CODACY_PROJECT_TOKEN” y el valor del token del paso 15 de esta sección.

Actions secrets / New secret

Name *

CODACY_PROJECT_TOKEN

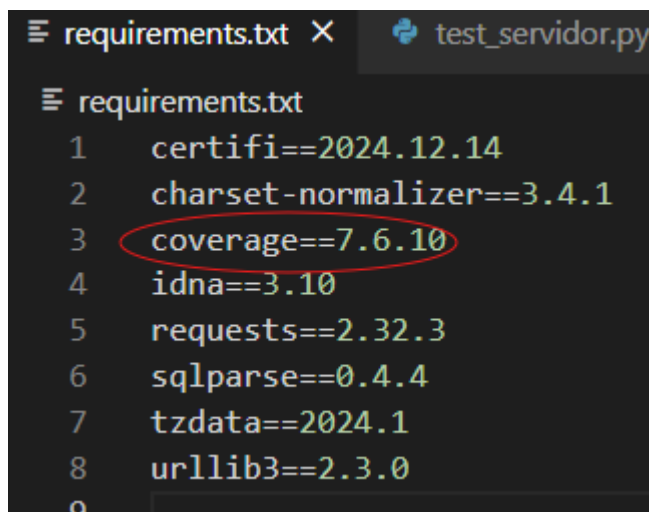
Secret *

769a2e3e [REDACTED] 25fca6e1be

Add secret

18. Para añadir la cobertura de test es necesario instalar la dependencia “coverage”.

- **pip install coverage**
- **pip freeze > requirements.txt**



```
requirements.txt X test_servidor.py
requirements.txt
1 certifi==2024.12.14
2 charset-normalizer==3.4.1
3 coverage==7.6.10
4 idna==3.10
5 requests==2.32.3
6 sqlparse==0.4.4
7 tzdata==2024.1
8 urllib3==2.3.0
9
```

19. Ahora será necesario añadir el paso de cobertura de test en el fichero **ci.yml**.

```
- name: Ejecutar pruebas
  run: |
    python -m coverage run test_servidor.py

- name: Generar informe de cobertura xml
  run: |
    python -m coverage xml

- name: Enviar reporte
  uses: codacy/codacy-coverage-reporter-action@v1
  with:
    project-token: ${ secrets.CODACY_PROJECT_TOKEN }
    coverage-reports: coverage.xml
```

Los dos primeros comandos se pueden ejecutar en local si se desea hacer un análisis de cobertura de test antes de committear.

20. Cuando finalice el paso anterior, suba los cambios (“**requirements.txt**” y “**ci.yml**” para poder validar el nuevo flujo de coberturas.

test

succeeded now in 12s

- > ☒ Set up job
- > ☒ Checkout código
- > ☒ Configurar Python
- > ☒ Instalar dependencias

✓ Ejecutar pruebas

```
1 ▶ Run python -m coverage run test_servidor.py
11 127.0.0.1 - - [20/Jan/2025 10:53:19] "GET / HTTP/1.1" 200 -
12 .
13 -----
14 Ran 1 test in 0.506s
15
16 OK
```

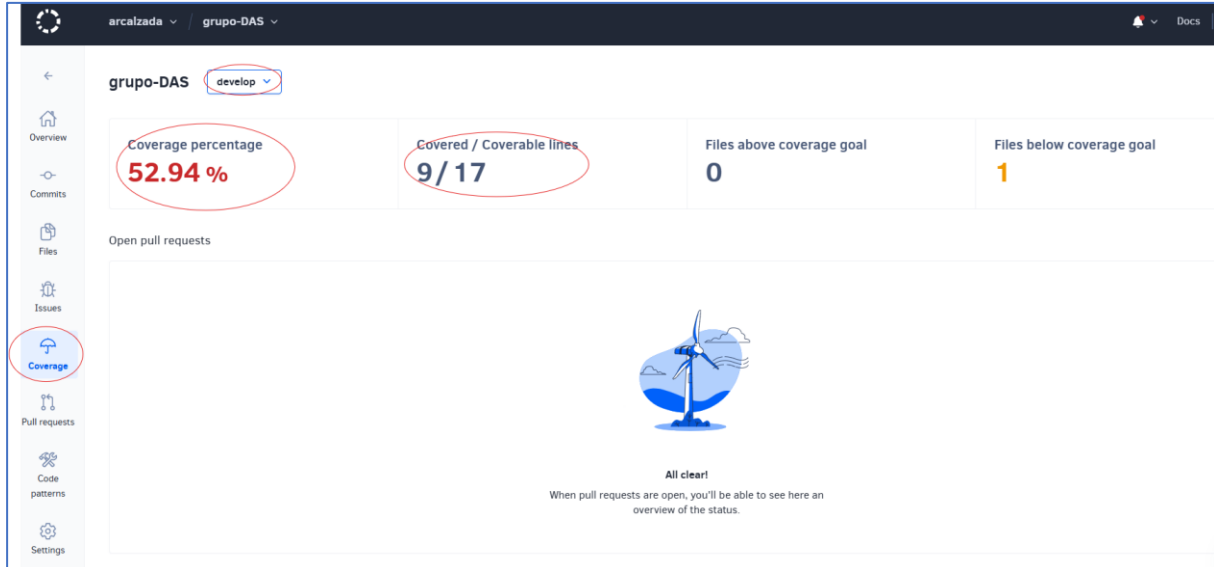
✓ Generar informe de cobertura xml

```
1 ▶ Run python -m coverage xml
11 Wrote XML report to coverage.xml
```

✓ Enviar reporte

```
1 ▶ Run codacy/codacy-coverage-reporter-action@v1
12 ▶ Run echo "ORGANIZATION_PROVIDER=$(if [[ $GITHUB_SERVER_URL == "https://github.com" ]]
24 ▶ Run set -eux
61 + report_list=
62 + '[' -n coverage.xml ']'
63 + report_list=coverage.xml
64 + IFS=,
65 + report_array=coverage.xml
66 + params=
67 + for report in $report_array
68 + '[' '!' -z coverage.xml ']'
69 + params=' -r coverage.xml'
70 + auth=
```

21. Por último, si accede de nuevo a [Codacy](#), al repositorio de la asignatura, y selecciona “coverage”, podrá ver el análisis que el CI le acaba de generar.



The screenshot displays the Codacy web interface for the 'grupo-DAS' repository. The top navigation bar shows 'arcalzada' and 'grupo-DAS'. The left sidebar contains icons for Overview, Commits, Files, Issues, Coverage (highlighted with a red circle), Pull requests, Code patterns, and Settings. The main content area shows the 'develop' branch selected. A summary table provides the following data:

Coverage percentage	Covered / Coverable lines	Files above coverage goal	Files below coverage goal
52.94 %	9 / 17	0	1

Below the table, the 'Open pull requests' section shows a message: 'All clear! When pull requests are open, you'll be able to see here an overview of the status.' accompanied by a wind turbine icon.