Boletín 1 – Control de Versiones & Desarrollo basado en Tests

Introducción....1
Objetivos....1
Pasos guiados....1
1. Instalación de Git....1
2. Creación de nuestro directorio de trabajo y repositorio git....3
3. Creación del entorno virtual Python....3
4. Creación del directorio para el código fuente....5
5. Primeros pasos con pytest y git....5
6. Juego del tres en raya....9
7. Trabajando con ramas y tags....17
8. Ejercicios opcionales....18

Introducción

En este boletín vamos a practicar con algunos conceptos relativos a la creación y gestión de un repositorio git. El proyecto a alojar en dicho repositorio permitirá desarrollar una serie de pruebas unitarias mediante `pytest`.

Objetivos

- Creación de un repositorio git de forma local y en GitHub mediante línea de comandos.
- Gestión de los ficheros alojados en dicho repositorio mediante comandos git.
- Creación y gestión de diferentes `branches` en el repositorio.
- Creación y ejecución de pruebas unitarias mediante `pytest`.

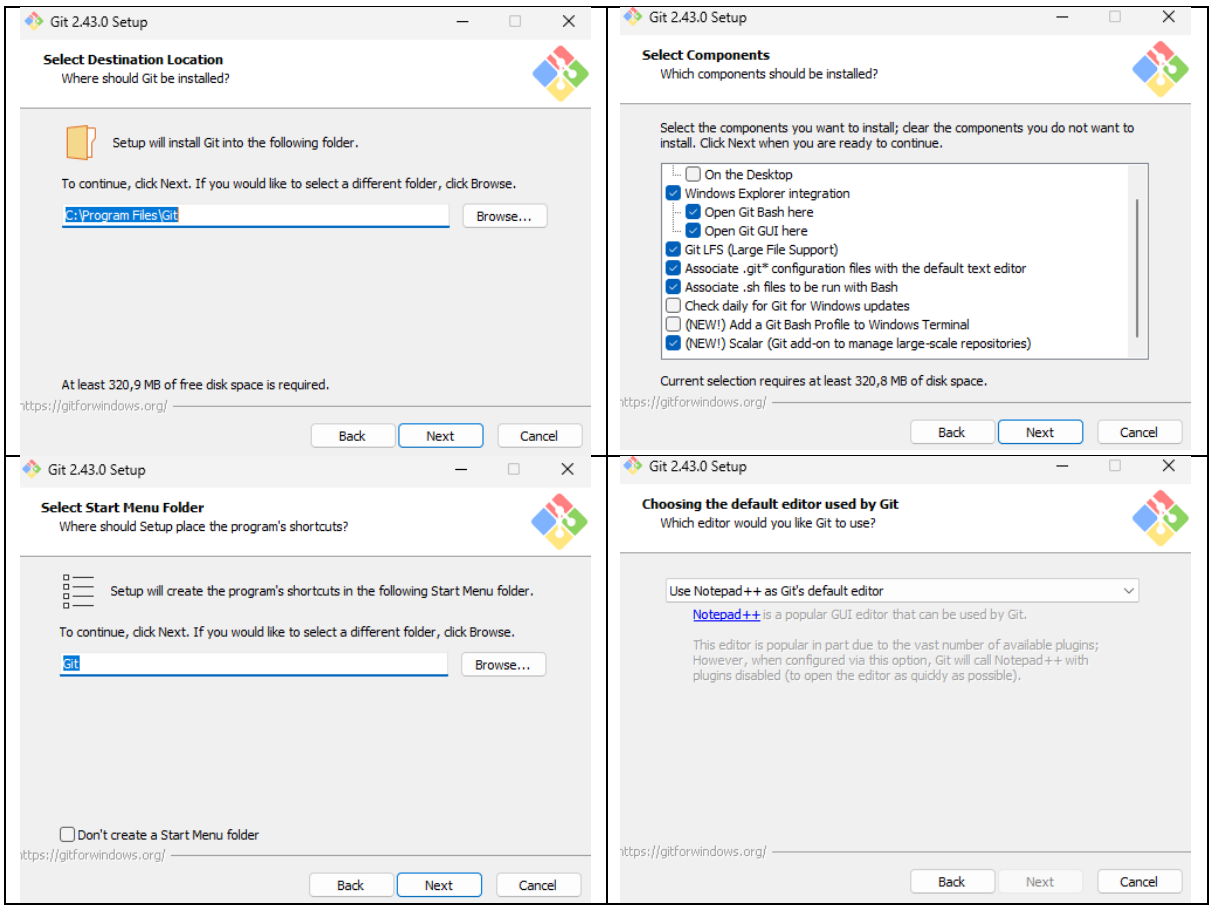
Pasos guiados

1. Instalación de Git

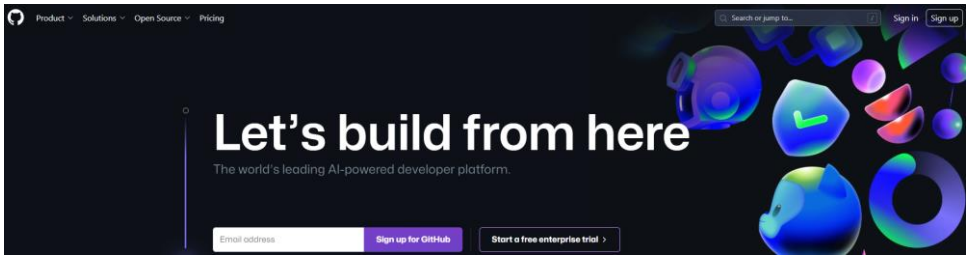
En este paso vamos a instalar git en nuestra máquina local y crear una cuenta en GitHub. Si ya lo tienes instalado git en tur ordenador y tienes creada una cuenta en GitHub puedes pasar a la sección B.

En primer lugar, vamos a descargarnos el instalador de https://git-scm.com/download/win y seleccionamos la opción Standalone Installer que se adapte a la arquitectura de nuestro ordenador.

A continuación, hacemos doble click sobre el fichero `Git...exe` descargado y comenzamos el proceso de instalación de git en nuestro ordenador. El proceso es bastante sencillo pues simplemente tenemos que mantener las opciones en todos los pasos SALVO cuando nos pida definir el editor git por defecto (ver imagen inferior derecha de la siguiente tabla). En dicho caso es mejor optar por `Notepad++` debido a su mayor facilidad de uso. Para ello deberemos instalar previamente dicho editor en nuestra máquina a partir del enlace (https://notepad-plus-plus.org/downloads/).



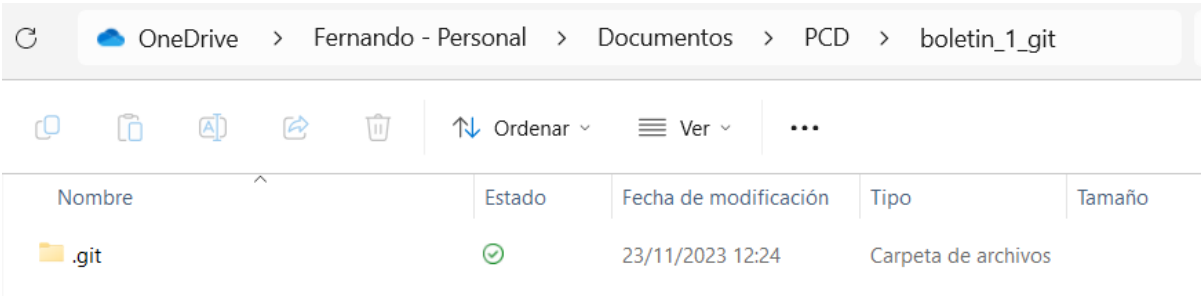
Una vez hemos completado la instalación, debemos crearnos una cuenta en el repositorio git online github, https://github.com/, pinchar sobre el botón Signup siguado arriba a la derecha y seguir las instrucciones indicadas



2. *Creación de nuestro directorio de trabajo y repositorio git*

Primero, crearemos un directorio llamado boletin_1_git en nuestra máquina local que corresponderá al directorio de trabajo en este boletín. A continuación, vamos a asociar un repositorio git a él ejecutando `git init` en la línea de comandos. Esto generará un directorio .git dentro de nuestro directorio de trabajo.

```
PS C:\Users\ferna\OneDrive\Documentos\PCD\boletin_1_git> git init
Initialized empty Git repository in C:/Users/ferna/OneDrive/Documentos/PCD/boletin_1_git/.git/
```



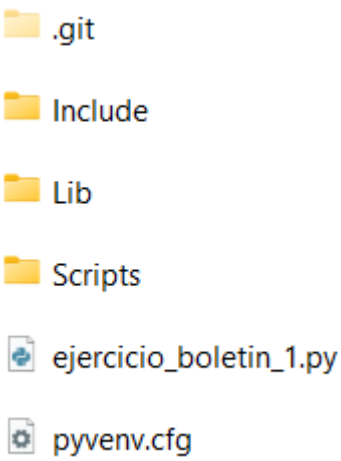
Ahora crearemos un fichero dentro del directorio que alojará el código fuente sobre el que trabajaremos. Vamos a llamar a dicho fichero, `ejercicio_boletin_1.py`

Ahora deberemos registrar, si no lo hemos hecho previamente, nuestras credenciales de GitHub en nuestro ordenador. Para ello ejecutamos los siguientes dos comandos:

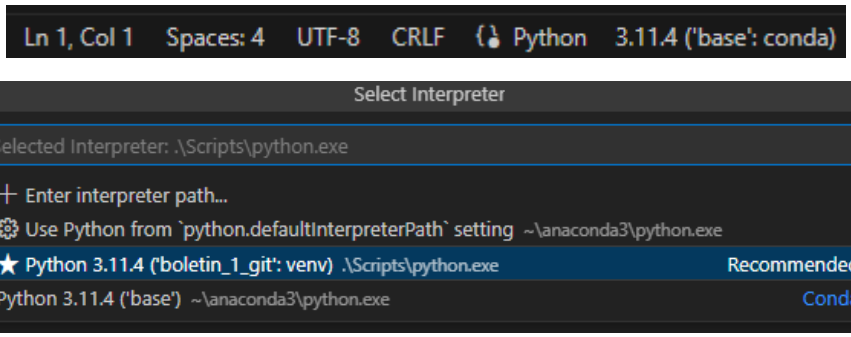
```
git config --global user.email "cuentaDeCorreoEnGitHub"
git config --global user.name "nombreDeUsuarioEnGitHub"
```

3. *Creación del entorno virtual Python*

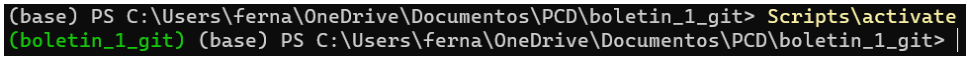
Creemos ahora el entorno virtual en Python sobre el que trabajaremos en este boletín. Para ello, nos situamos dentro del directorio `boletin_1_git` y ejecutamos el comando "`python -m venv .`" Esto generará una serie de carpetas en nuestro directorio de trabajo como podemos observar en las figuras de abajo



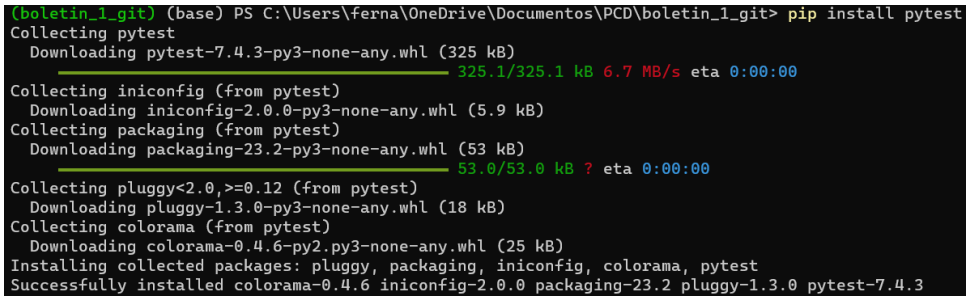
Ahora abrimos el directorio de trabajo en nuestra instalación de *Visual Studio Code* y lo enlazamos con el entorno virtual que acabamos de crear. Para ello, simplemente tenemos que pulsar sobre la versión de Python que aparece en la esquina inferior derecha de ventana principal de la aplicación y seleccionar el entorno `boletin_1_git` en el desplegable que nos aparece[1].



Ahora vamos a instalar `pytest` en nuestro ordenador. Para ello, primero nos aseguramos de que tenemos activado nuestro entorno virtual ejecutando `Scripts\activate`.



A continuación, ya podemos instalar la librería `pytest` dentro de nuestro entorno virtual. Para ello simplemente tenemos que ejecutar `pip install pytest` en la línea de comandos.



En este punto ya tenemos instalado pytest en nuestro entorno virtual.

[1] Set-ExecutionPolicy RemoteSigned -Scope LocalMachine

4. *Creación del directorio para el código fuente.*

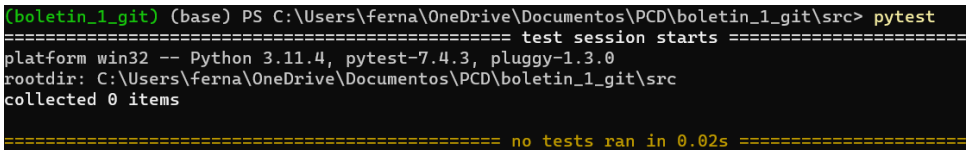
Vamos ya a empezar a trabajar sobre el código fuente del proyecto. Para ello y a fin de lograr una mejor organización del proyecto vamos a crear un directorio `src` dentro de `boletín_1_git`. Será sobre dicho directorio donde vayamos generando los diferentes ficheros de código fuente del boletín.

5. *Primeros pasos con pytest y git*

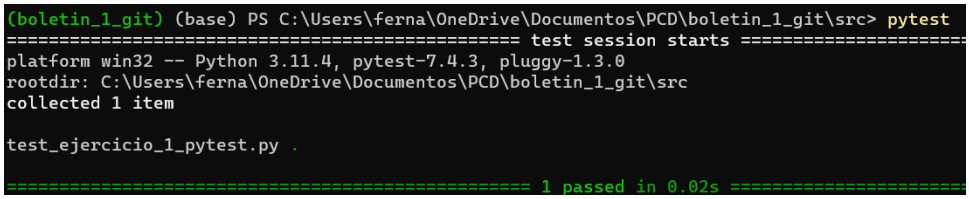
Vamos a crear un fichero llamado `ejercicio_1_pytest.py` y vamos a escribir el siguiente código en el mismo en el cual se crea una prueba unitaria para comprobar el funcionamiento de la función `capital_case`:

```
import pytest
def capital_case(x):
    return x.capitalize()
def test_capital_case():
    assert capital_case('semáforo') == 'Semáforo'
```

Si ahora desde la línea de comandos ejecutamos `pytest` veremos que no se nos devuelve ningún resultado asociado a ninguna prueba



¿Dónde está el fallo? En el nombre del fichero. Con `pytest`, todos los ficheros que queremos que sean procesados por `pytest` deben de incluir el prefijo test_. Por tanto, renombramos el fichero a `test_ejercicio_1_pytest.py` y volvemos a ejecutar el comando `pytest` en la línea de comandos y vemos que esta vez sí, hemos podido ejecutar el test unitario superándolo con éxito.

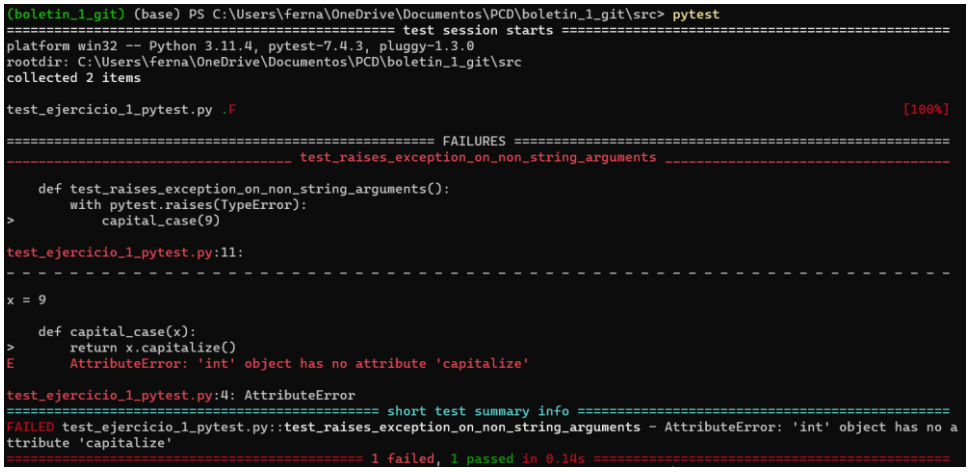


Vamos ahora a enriquecer nuestro panel de pruebas con un segundo test en donde se compruebe si nuestra función `capital_test` tiene alguna comprobación respecto al tipo de datos de entrada. Para ello añadimos la siguiente función:

```
def test_raises_exception_on_non_string_arguments():
```

```
    with pytest.raises(TypeError):
            capital_case(9)
```

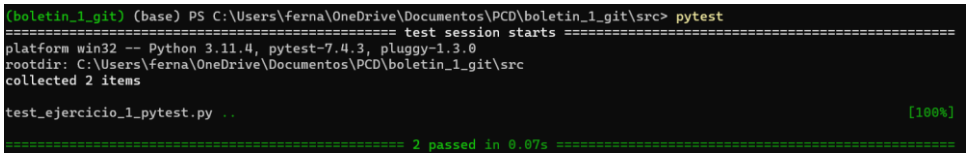
Si ahora ejecutamos de nuevo `pytest` en la línea de comandos vemos que nuestro código no supera el test que acabamos de definir:



Para solucionarlo, debemos de modificar la función `capital_test` para que haga una comprobación de tipos al comienzo de la función. Para ello, vamos a modificar su código como sigue:

```
def capital_case(x):
    if not isinstance(x, str):
     raise TypeError('Debes de proporcionar un string')
        return x.capitalize()
```

Si ahora ejecutamos el comando `pytest` vemos que nuestra pequeña función ya supera los dos *tests* unitarios.



En este punto ya estamos en disposición de añadir el fichero `test_ejercicio_1_pytest.py` a nuestro repositorio git mediante el comando `git add test_ejercicio_1_pytest.py`

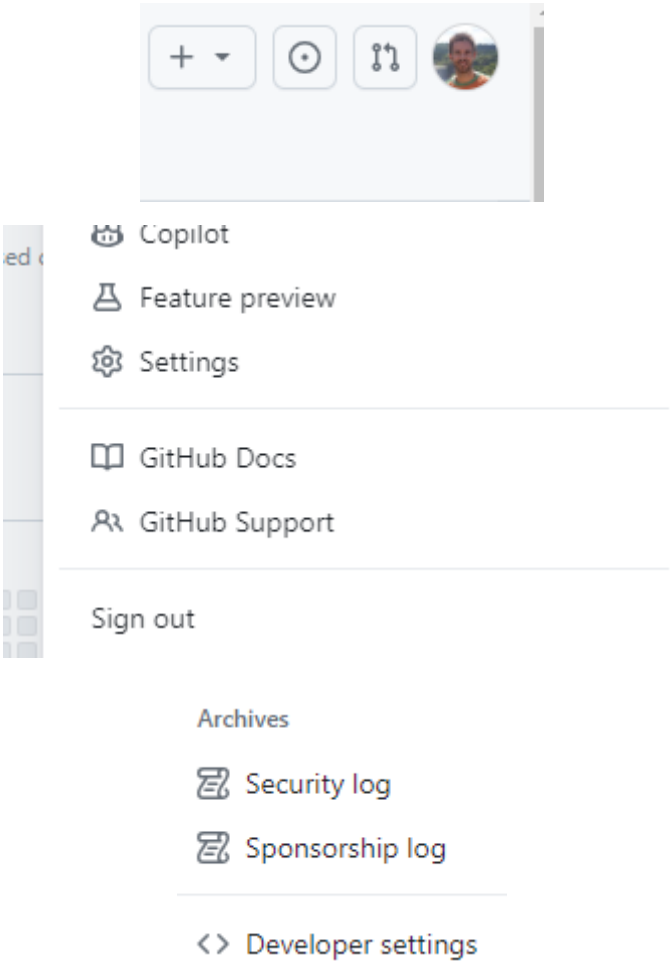
Si ahora ejecutamos el comando `git status` para ver el estado actual de nuestro repositorio veríamos que nuestro fichero .py está en el área preparatoria (*staging area*) antes del *commit*.

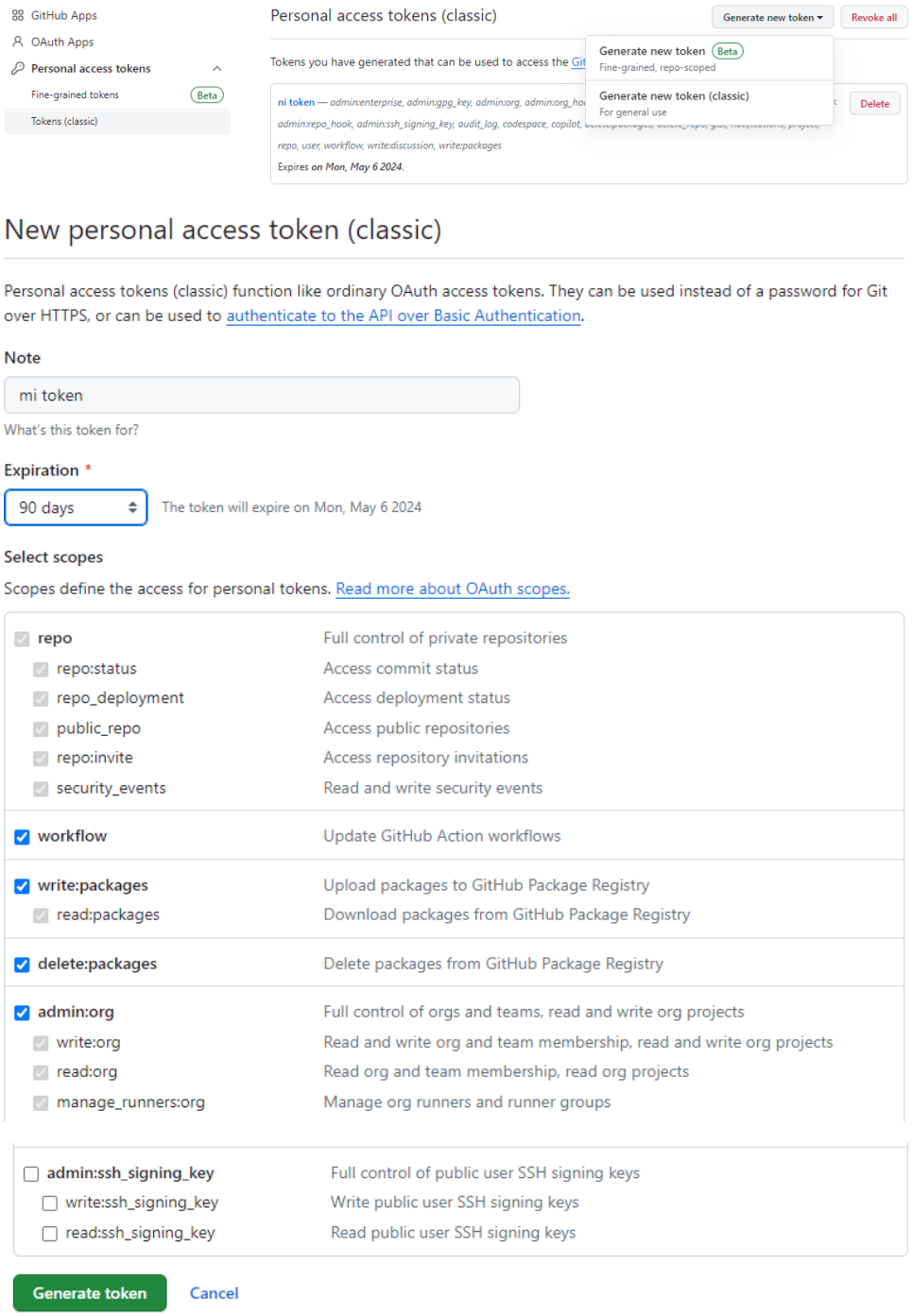
```
(boletin_1_git) (base) PS C:\Users\ferna\OneDrive\Documentos\PCD\boletin_1_git\src> git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   test_ejercicio_1_pytest.py

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        ../Lib/
        ../Scripts/
        ../pyvenv.cfg
        __pycache__/
```

Vamos por tanto a ejecutar el *commit* a fin de registrar dicho fichero en el directorio. Pare ello, ejecutamos el siguiente comando `git commit –m “Tests completados en el primer ejercicio de prueba”`

```
(boletin_1_git) (base) PS C:\Users\ferna\OneDrive\Documentos\PCD\boletin_1_git\src> git commit -m "Test completados en e
el primer ejemplo de prueba"
[master (root-commit) a764b2d] Test completados en eel primer ejemplo de prueba
 1 file changed, 14 insertions(+)
 create mode 100644 src/test_ejercicio_1_pytest.py
```

Una vez ya tenemos nuestro repositorio git local estable y con su primer commit realizado, vamos a crear un repositorio remoto en github que vamos a llamar `pcd_boletin1`. Una vez lo hayamos creado debermos generar un token de acceso para poder trabajar sobre él desde la línea de comandos. Para ello deberemos ir la opción Settings que se nos muestra al pulsar sobre nuestro usuario en la esquina superior derecha.





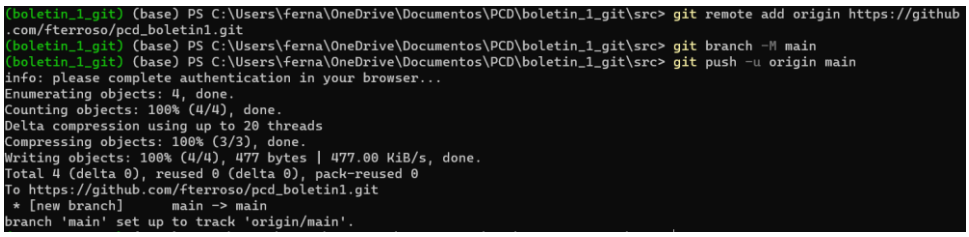
El toque que se nos genera deberemos usarlo en las instrucciones siguientes.

A continuación, vamos a subir nuestro repositorio local a dicho repositorio remoto. Para ello, debemos ejecutar los siguientes comandos git:

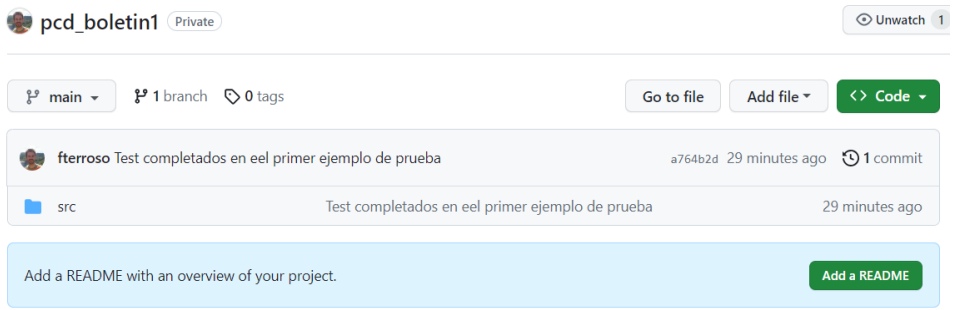
```
git remote add origin https://github.com/<usuario>/pcd_boletin1.git

git branch -M main

git remote set-url origin
https://<tokenquehemosgenerado>@github.com/<usuario>/pcd_boletin1
```

```
git push -u origin main
```

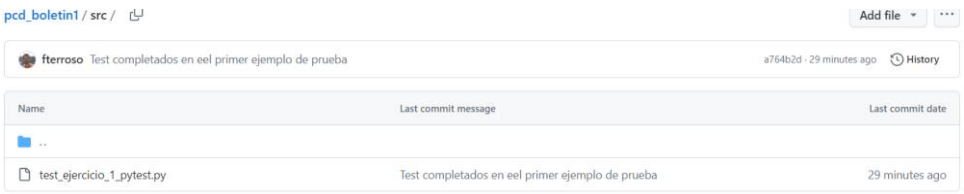
Con el primer comando (`git remote add origin`) asociamos a nuestro repositorio remoto en github el nombre `origin`. A continuación, con `git branch` renombramos la única rama de nuestro proyecto hasta ahora como `main`. Por último, con `git push` subimos al repositorio remoto nuestra recién renombrada rama `main`.



Si todo ha ido bien, nuestro repositorio en GitHub debe de mostrar la siguiente estructura:



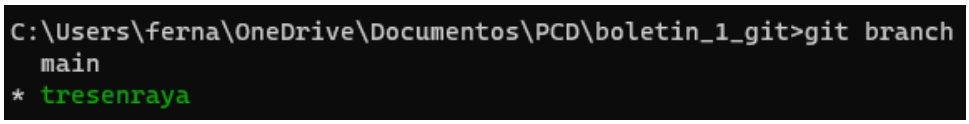
Y al hacer click sobre la carpeta `src` se nos debe de mostrar el siguiente contenido:



6. *Juego del tres en raya*

Vamos ahora a enriquecer nuestro repositorio programando el juego del 3 en raya. Para ello, primeros vamos a crear una nueva rama en nuestro repositorio llamada `tresenraya` con el comando `git branch tresenraya`

Podemos establecer ahora dicha rama como activa con `git checkout tresenraya`. Si ahora ejecutamos `git branch` vemos que efectivamente ahora tenemos dos ramas, `main` y `tresenraya`, siendo esta última en la que nos encontramos actualmente.



vamos en primer lugar a crear un fichero llamado `juego_3_en_raya.py` en nuestro directorio local `src`. Primero, implementaremos la función para mostrar el tablero por pantalla con una dimensión n y asumiendo que los jugadores han realizado movimientos (`movimientos_jugadores`).

```
fichas= ['o','x']
def generar_tablero(n, movimientos_jugadores):
    tablero=[]
    for i in range(n):
        fila=['_' for i in range(n)]
        for j in range(n):
            casilla_vacia = True
            for k in range(len(movimientos_jugadores)):
                movimientos_jugador= movimientos_jugadores[k]

                if i in movimientos_jugador:
                    if j in movimientos_jugador[i]:
                        fila[j]=fichas[k]
        tablero.append(fila)
    return tablero

t= generar_tablero(n, movimientos_jugadores)
print(t, len(t))
```

Podemos testear que dicho código genera correctamente el tablero con las dimensiones adecuadas con el siguiente código

```
import pytest
def  test_generar_tablero():
      mov_jugador_1 = {}
      mov_jugador_2 = {}
      movimientos_jugadores=[mov_jugador_1, mov_jugador_2]
      n=3
      t= generar_tablero(n, movimientos_jugadores)
```

```
    assert len(t)== n
    for f in t:
        assert len(f) == n
```

Ahora debemos forzar a `pytest para` que evalúe nuestro fichero `juego_3_en_raya.py`. Para ello, en la línea de comandos escribimos

```
pytest juego_3_en_raya.py -v
```

```
(boletin_1_git) PS C:\Users\ferna\OneDrive\Documentos\PCD\boletin_1_git\src> pytest .\juego_3_en_raya.py -v
============================================= test session starts =============================================
platform win32 -- Python 3.11.4, pytest-7.4.3, pluggy-1.3.0 -- C:\Users\ferna\OneDrive\Documentos\PCD\boletin_1_git\Scripts\python.ex
cachedir: .pytest_cache
rootdir: C:\Users\ferna\OneDrive\Documentos\PCD\boletin_1_git\src
collected 1 item

juego_3_en_raya.py::test_generar_tablero PASSED                                                      [100%]
```

La siguiente fase va a consistir en desarrollar un método que permita determinar si el movimiento de un jugador (colocar una ficha en la casilla con coordenadas (x,y)) es válido o no.

```
"""
Método que comprueba que un movimiento de un jugador es válido
    * x: fila donde el jugador quiere colocar la ficha.
    * y: columna donde el jugador quiere colocar su ficha.
    * movimientos_otro_jugador: listado con las celdas ocupadas por el otro jugador.
"""
def movimiento_valido(x, y, movimientos_otro_jugador):
    if x > n or y > n:
        return False
    if x in movimientos_otro_jugador:
        movimientos_en_columna= movimientos_otro_jugador[x]
        if y in movimientos_en_columna:
            return False
    return True
```

A partir de este método podemos diseñar varios test unitarios para comprobar su correcta funcionalidad:

```
def test_movimiento_columna_fuera_tablero():
```

```
    movimientos_otro_jugador={}
    x=  1
    y=  n+1
    assert False == movimiento_valido(x,y,movimientos_otro_jugador)


def test_movimiento_fila_y_columna_fuera_tablero():
    movimientos_otro_jugador={}
    x=  n+1
    y=  n+1
    assert False == movimiento_valido(x,y,movimientos_otro_jugador)


def test_movimiento_incorrecto():
    movimientos_otro_jugador={2:[3]}
    x=  2
    y=  3
    assert False == movimiento_valido(x,y,movimientos_otro_jugador)
```

De nuevo, ejecutando

```
pytest juego_3_en_raya.py -v
```

podemos re-evaluar todos los tests

```
============================================================ 2 passed in 0.02s ============================================================
(boletin_1_git) PS C:\Users\ferna\OneDrive\Documentos\PCD\boletin_1_git\src> pytest .\juego_3_en_raya.py -v
============================================================ test session starts ============================================================
platform win32 -- Python 3.11.4, pytest-7.4.3, pluggy-1.3.0 -- C:\Users\ferna\OneDrive\Documentos\PCD\boletin_1_git\Scripts\python.exe
cachedir: .pytest_cache
rootdir: C:\Users\ferna\OneDrive\Documentos\PCD\boletin_1_git\src
collected 5 items

juego_3_en_raya.py::test_tamaño_tablero PASSED                                                                                      [ 20%]
juego_3_en_raya.py::test_movimiento_fila_fuera_tablero PASSED                                                                       [ 40%]
juego_3_en_raya.py::test_movimiento_columna_fuera_tablero PASSED                                                                    [ 60%]
juego_3_en_raya.py::test_movimiento_fila_y_columna_fuera_tablero PASSED                                                             [ 80%]
juego_3_en_raya.py::test_movimiento_incorrecto PASSED                                                                               [100%]
```

Sin embargo, vemos que con la instrucción anterior se está volviendo a re-evaluar el test `test_tamaño_tablero`, lo cual es algo redundante. En este caso podemos limitar la ejecución sólo de aquellas pruebas que contengan la cadena `movimiento` en su título. Para ello ejecutamos el comando

```
pytest juego_3_en_raya.py -v –k movimiento.
```

```
============================================================ 5 passed in 0.02s ============================================================
(boletin_1_git) PS C:\Users\ferna\OneDrive\Documentos\PCD\boletin_1_git\src> pytest .\juego_3_en_raya.py -v -k movimiento
============================================================ test session starts ============================================================
platform win32 -- Python 3.11.4, pytest-7.4.3, pluggy-1.3.0 -- C:\Users\ferna\OneDrive\Documentos\PCD\boletin_1_git\Scripts\python.exe
cachedir: .pytest_cache
rootdir: C:\Users\ferna\OneDrive\Documentos\PCD\boletin_1_git\src
collected 5 items / 1 deselected / 4 selected

juego_3_en_raya.py::test_movimiento_fila_fuera_tablero PASSED                                                                      [ 25%]
juego_3_en_raya.py::test_movimiento_columna_fuera_tablero PASSED                                                                   [ 50%]
juego_3_en_raya.py::test_movimiento_fila_y_columna_fuera_tablero PASSED                                                            [ 75%]
juego_3_en_raya.py::test_movimiento_incorrecto PASSED                                                                              [100%]
```

En este punto ya podríamos subir pensar en subir la versión actual del juego a la rama `tresenraya` de nuestro repositorio git. Para ello, ejecutamos los siguientes comandos git en la línea de comandos:

```
git add juego_3_en_raya.py

git commit –m 'tablero y comprobacion de movimientos completada'
```

Si comprobamos nuestro repositorio en github veremos que la rama `tresenraya` no ha sido subida a github, para ello deberemos ejecutar el comando

```
git push –u origin tresenraya
```

```
PS C:\Users\ferna\OneDrive\Documentos\PCD\boletin_1_git\src> git push -u origin tresenraya
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 20 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 946 bytes | 946.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'tresenraya' on GitHub by visiting:
remote:      https://github.com/fterroso/pcd_boletin1/pull/new/tresenraya
remote:
To https://github.com/fterroso/pcd_boletin1.git
 * [new branch]      tresenraya -> tresenraya
branch 'tresenraya' set up to track 'origin/tresenraya'.
```

Vamos ahora a implementar el método que permita determinar si un nuevo movimiento de un jugador le permite ganar el juego

```
def jugada_ganadora(movimientos_jugador):
    """

    Método que permite determinar si los movimientos de un jugador le
permite ganar una partida.

    Parámetros:

        * movimientos_jugador: dict con el conjunto de movimientos de un
jugador
    """

    #Comprobamos si hay 3 fichas en una fila
```

```
    for fila in movimientos_jugador:
        movimientos_columna = movimientos_jugador[fila]
        if len(movimientos_columna)==3:
            return True
    return False
```

Sobre este método vamos a definir dos test unitarios diferentes

```
def test_no_ganador():
    movimientos_jugador={2:[2,3]}
    assert False == jugada_ganadora(movimientos_jugador)


def test_ganador():
    movimientos_jugador={2:[1,2,3]}
    assert True == jugada_ganadora(movimientos_jugador)
```

Que podemos ejecutar con el comando

```
pytest .juego_3_en_raya.py -v -k ganador
```

viendo que los dos test han sido superados satisfactoriamente.

```
(boletin_1_git) PS C:\Users\ferna\OneDrive\Documentos\PCD\boletin_1_git\src> pytest .\juego_3_en_raya.py -v -k ganador
============================================= test session starts =============================================
platform win32 -- Python 3.11.4, pytest-7.4.3, pluggy-1.3.0 -- C:\Users\ferna\OneDrive\Documentos\PCD\boletin_1_git\Scripts\pyt
hon.exe
cachedir: .pytest_cache
rootdir: C:\Users\ferna\OneDrive\Documentos\PCD\boletin_1_git\src
collected 7 items / 5 deselected / 2 selected

juego_3_en_raya.py::test_no_ganador PASSED                                                           [ 50%]
juego_3_en_raya.py::test_ganador PASSED                                                              [100%]

========================================== 2 passed, 5 deselected in 0.03s ==========================================
```

Vamos a hacer un nuevo `commit` con los cambios realizados, pero esta vez vamos a usar un tag (`v0.5`) para etiquetar dicho `commit` y subirlo al repositorio remoto.

```
git add juego_3_en_raya.py
git commit –m 'añadida comprobación de jugada ganadora'
git tag v0.5
git push origin tresenraya:tresenraya
```

En este punto vamos a añadir la lógica del juego que permita hacerlo interactivo. En primer lugar implementamos el método que permita mostrar el estado actual del tablero por pantalla:

```
def mostrar_tablero(tablero):
    """
    Método que muestra el estado actual del tablero

    Parámetros:
        * tablero: dict con el tablero a mostrar
    """
    for fila in tablero:
        for celda in fila:
            print(celda,end='')
        print('\n')
```

Y a continuación implementamos el bucle que permite a cada uno de los dos jugadores ir añadiendo movimientos:

```
#Pedimos el tamaño del tablero en que se va a realizar el juego
n=int(input('Introduce el tamaño del tablero cuadrado:'))

casillas_libres = n*n
jugador_activo = 0
movimientos_jugador_1 = {}
movimientos_jugador_2 = {}
movimientos_jugadores = [movimientos_jugador_1, movimientos_jugador_2]

tablero= generar_tablero(n,movimientos_jugadores)
mostrar_tablero(tablero)

while casillas_libres > 0:

    casilla_jugador = input(f"JUGADOR {jugador_activo+1}: Introduce
movimiento (x,y): ")

    casilla_jugador= casilla_jugador.strip()
    x= int(casilla_jugador.split(',')[0])-1
```

```
    y= int(casilla_jugador.split(',')[1])-1


    print(casilla_jugador,x,y)


    movimientos_jugador_activo= movimientos_jugadores[jugador_activo]
    movimientos_otro_jugador = movimientos_jugadores[(jugador_activo+1)%2]
    if movimiento_valido(x,y, movimientos_otro_jugador):
        mov_col= movimientos_jugador_activo.get(x,[])
        mov_col.append(y)
        movimientos_jugador_activo[x]= mov_col


        clear = lambda: os.system('cls')
        clear()
        tablero= generar_tablero(n,movimientos_jugadores)
        mostrar_tablero(tablero)
        if jugada_ganadora(movimientos_jugador_activo):
            print(F"ENHORABUENA EL JUGADOR {jugador_activo+1} HA GANADO")
            break
    else:
        frequency = 2000 # Set Frequency To 2500 Hertz
        duration = 1000  # Set Duration To 1000 ms == 1 second
        print('\a')
        print("Movimiento invalido. Turno para el siguiente jugador")


    casillas_libres= casillas_libres -1
    jugador_activo = (jugador_activo+1) % 2
```

Ahora podemos confirmar los cambios, crear un nuevo tag `v1.0` sobre dicho `commit` y subirlo todo al servidor remoto

```
git add juego_3_en_raya.py
git commit -m 'añadida primera version de la lógica del juego'
git tag v1.0
```

```
git push origin tresenraya:tresenraya
```

```
(boletin_1_git) PS C:\Users\ferna\OneDrive\Documentos\PCD\boletin_1_git\src> git add juego_3_en_raya.py
(boletin_1_git) PS C:\Users\ferna\OneDrive\Documentos\PCD\boletin_1_git\src> git commit -m 'añadida primera version de la lógica
 del juego'
[tresenraya 1d240da] añadida primera version de la lógica del juego
 1 file changed, 62 insertions(+), 3 deletions(-)
(boletin_1_git) PS C:\Users\ferna\OneDrive\Documentos\PCD\boletin_1_git\src> git tag v1.0
(boletin_1_git) PS C:\Users\ferna\OneDrive\Documentos\PCD\boletin_1_git\src> git push origin tresenraya:tresenraya
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 20 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 1.15 KiB | 1.15 MiB/s, done.
Total 4 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/fterroso/pcd_boletin1.git
   735a16c..1d240da  tresenraya -> tresenraya
```

7. Trabajando con ramas y tags

En este punto tenemos en nuestro repositorio dos ramas diferentes: `master` y `tresenraya`. Además, esta última contiene dos de sus commits etiquetados (`v0.5` y `v1.0`). Estos tags nos permiten movernos fácilmente entre las diferentes versiones del juego. Así ejecutando...

```
git checkout v0.5
```

...restauraremos el proyecto tal y como estaba al hacer dicho *commit* y podríamos crear una nueva rama `4enraya` a partir de dicho punto:

```
git branch cuatroenraya
git checkout cuatroenraya
```

```
(boletin_1_git) PS C:\Users\ferna\OneDrive\Documentos\PCD\boletin_1_git\src> git branch cuatroenraya
(boletin_1_git) PS C:\Users\ferna\OneDrive\Documentos\PCD\boletin_1_git\src> git branch
* (HEAD detached at v0.5)
  cuatroenraya
  main
  tresenraya
(boletin_1_git) PS C:\Users\ferna\OneDrive\Documentos\PCD\boletin_1_git\src> git checkout cuatroenraya
Switched to branch 'cuatroenraya'
(boletin_1_git) PS C:\Users\ferna\OneDrive\Documentos\PCD\boletin_1_git\src> git branch
* cuatroenraya
  main
  tresenraya
```

Posteriormente, podemos subir dicha rama al servidor remoto con

```
git push origin cuatroenraya
```

```
(boletin_1_git) PS C:\Users\ferna\OneDrive\Documentos\PCD\boletin_1_git\src> git push origin cuatroenraya
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'cuatroenraya' on GitHub by visiting:
remote:      https://github.com/fterroso/pcd_boletin1/pull/new/cuatroenraya
remote:
To https://github.com/fterroso/pcd_boletin1.git
 * [new branch]      cuatroenraya -> cuatroenraya
```

En este punto podríamos empezar a trabajar en un nuevo desarrollo para el juego del 4 en raya a partir del estado actual del proyecto.

Nosotros, sin embargo, vamos a volver al HEAD de la rama principal pues lo que vamoa a hacer es unirla con la de 3enraya con

```
git checkout main
```

```
(boletin_1_git) PS C:\Users\ferna\OneDrive\Documentos\PCD\boletin_1_git\src> git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
(boletin_1_git) PS C:\Users\ferna\OneDrive\Documentos\PCD\boletin_1_git\src> git branch
  cuatroenraya
* main
  tresenraya
```

Ahora podemos fusionar ambas ramas en una única `main` con

```
git merge tresenraya
```

```
(boletin_1_git) PS C:\Users\ferna\OneDrive\Documentos\PCD\boletin_1_git\src> git merge tresenraya
Updating a764b2d..1d240da
Fast-forward
 src/juego_3_en_raya.py | 153 +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 1 file changed, 153 insertions(+)
 create mode 100644 src/juego_3_en_raya.py
```

Subimos los cambios al servidor remoto con

```
git push origin main
```

```
(boletin_1_git) PS C:\Users\ferna\OneDrive\Documentos\PCD\boletin_1_git\src> git push origin
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/fterroso/pcd_boletin1.git
   a764b2d..1d240da  main -> main
```

Ahora borramos la rama `tresenraya` pues ya no nos hace falta

```
git branch –d tresenraya
```

```
(boletin_1_git) PS C:\Users\ferna\OneDrive\Documentos\PCD\boletin_1_git\src> git branch -d tresenraya
Deleted branch tresenraya (was 1d240da).
```

Finalmente borramos también dicha rama en nuestro servidor local con

```
git push origin –-delete tresenraya
```

```
(boletin_1_git) PS C:\Users\ferna\OneDrive\Documentos\PCD\boletin_1_git\src> git push origin --delete tresenraya
To https://github.com/fterroso/pcd_boletin1.git
 - [deleted]         tresenraya
```

8. *Ejercicios opcionales*

1. Como habrás comprobado, la función que comprueba si un jugador ha ganado una partida al 3 en raya, `jugada_ganadora`, está incompleta. Termina dicha función y sube los cambios a la rama main de tu repositorio.

2. Implementa la versión del 4 en raya a partir del *commit* inicial de la rama git `cuatroenraya`. Ejecuta los comandos necesarios para situarte en dicha rama.