GENERALITAT VALENCIANA
Conselleria d'Educació, Investigació, Cultura i Esport

Unión Europea
Fondo Social Europeo
El FSE invierte en tu futuro

I. E. S. MARE NOSTRUM
Beato Fco. Castelló Aleu s/n • ☎ 965936520
03008 ALICANTE
www.iesmarenostrum.com
correo@iesmarenostrum.com

Visual Studio 2022: Control de versiones con Git

1. Introducción

En esta práctica vamos a usar Git como software de control de versiones distribuido. Para ello usaremos GitHub, un servicio web que permite alojar repositorios de Git, de forma gratuita siempre que sean de software libre.

La práctica se realizará en grupos de 2 alumnos, debiendo realizar cada alumno una parte de la programación y su integración en el repositorio. En el caso de que no tengas la posibilidad de formar un grupo (educación semipresencial), se puede hacer de manera individual utilizando dos cuentas de correo diferentes. También es aconsejable, en este caso y para para evitar problemas, utilizar dos usuarios diferentes del equipo.

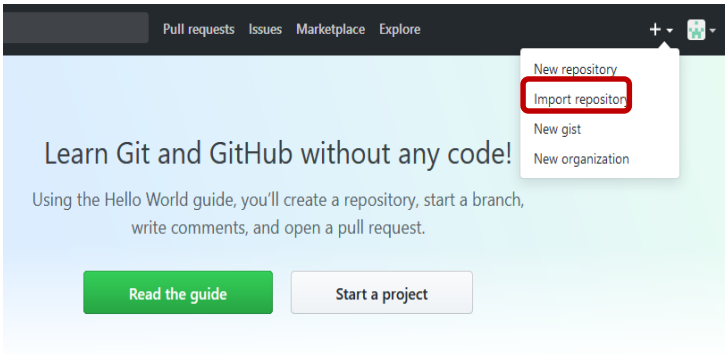
Al final de la práctica se deberá entregar una memoria con los pasos realizados para **trabajar con el repositorio**, tanto local como remoto (no de la programación), indicando los problemas surgidos y cómo se han solucionado, todo ello con capturas de pantalla. También se deberá indicar la **URL pública del repositorio**, para poder inspeccionar el código y ejecutar el programa, así como **autenticar la práctica** (historial de commits, código –comentarios o variables-, nombres de ramas, …) con **vuestras iniciales y el curso actual**.

2. Registro en GitHub y creación del repositorio

El primer paso a dar par esta práctica será crear una cuenta en GitHub.

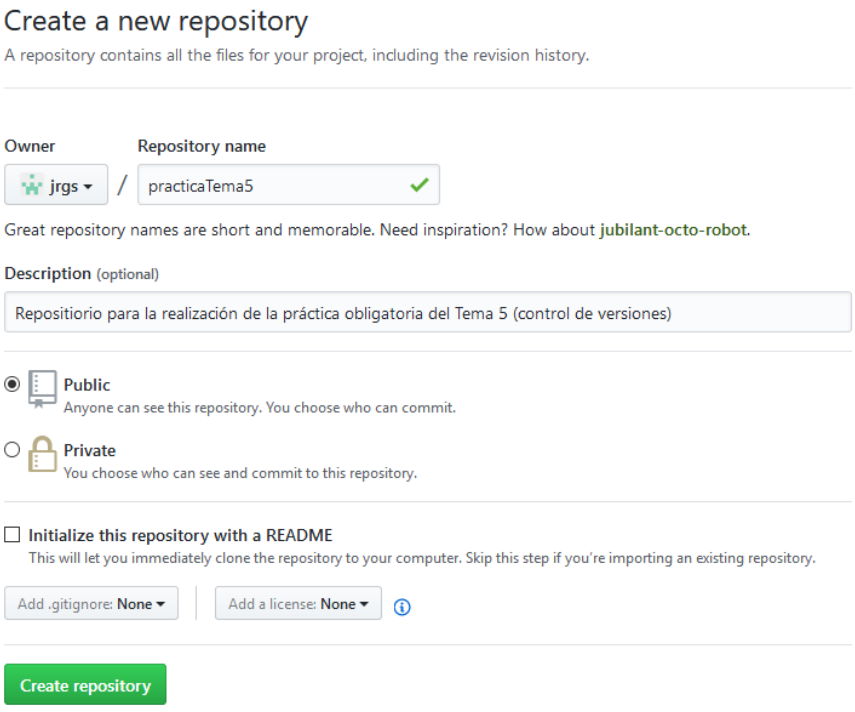
1. Accedemos a la web de GitHub (https://github.com/) y nos registramos (botón **Sign up for GitHub**). Hay que tener en cuenta que el campo **username** sólo admite caracteres alfanuméricos y guiones. Incluir vuestras iniciales y el curso en el nombre de usuario.

Lo normal para empezar es elegir la modalidad **Free**, que es gratuita,



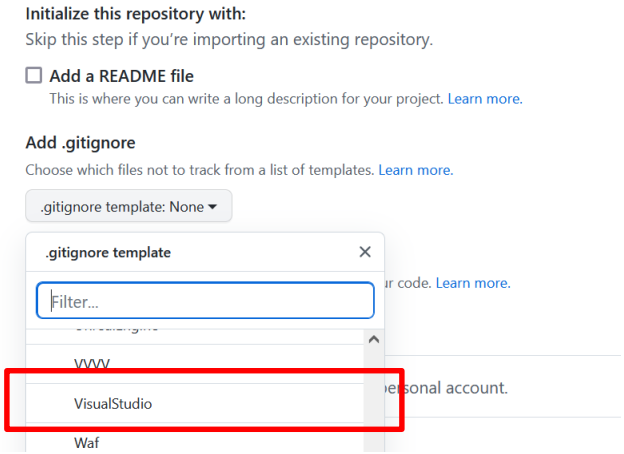
aunque no podemos tener repositorios privados. Esto para el caso de esta práctica no es ningún problema.

2. Una vez creada la cuenta, ya podemos crear nuestro repositorio:

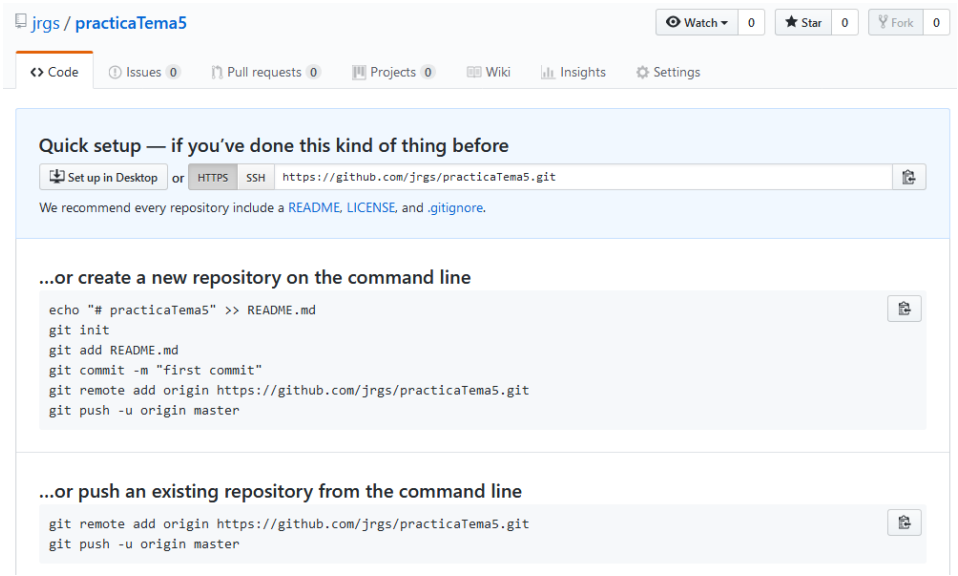


Este paso sólo lo tendrá que realizar un miembro del grupo, ya que el repositorio será compartido. Le ponemos un nombre significativo (por ejemplo *practicaGIT*) y una descripción, sin marcar la casilla de inicializar el repositorio.

Antes de crear el repositorio, es buena idea añadir un archivo .gitignore. Este archivo contiene una lista de ficheros que no se sincronizarán (generalmente, archivos temporales del IDE). Si se nos olvida, siempre podemos añadirlo más adelante.

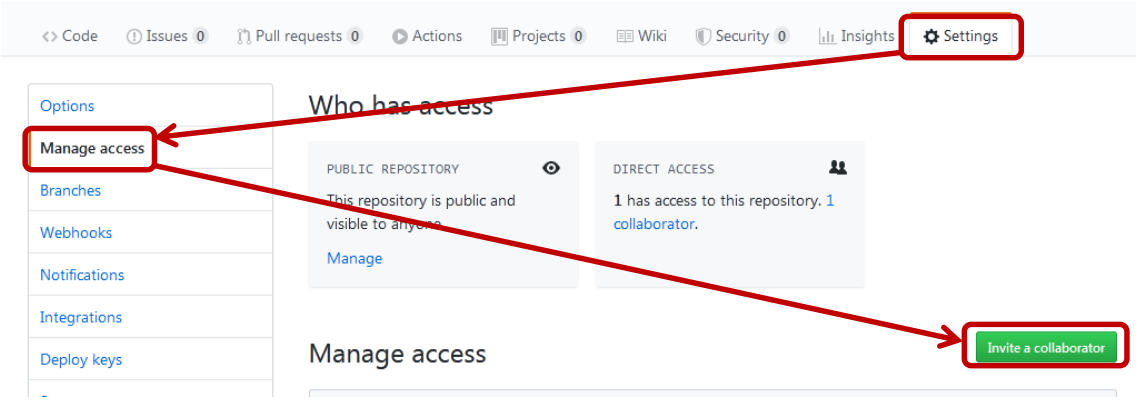


Y este es el aspecto que presenta nuestro repositorio vacío:



4. El siguiente paso será añadir al resto de usuarios del grupo. Hacemos click en **Settings** en la barra de opciones del repositorio, y en la siguiente

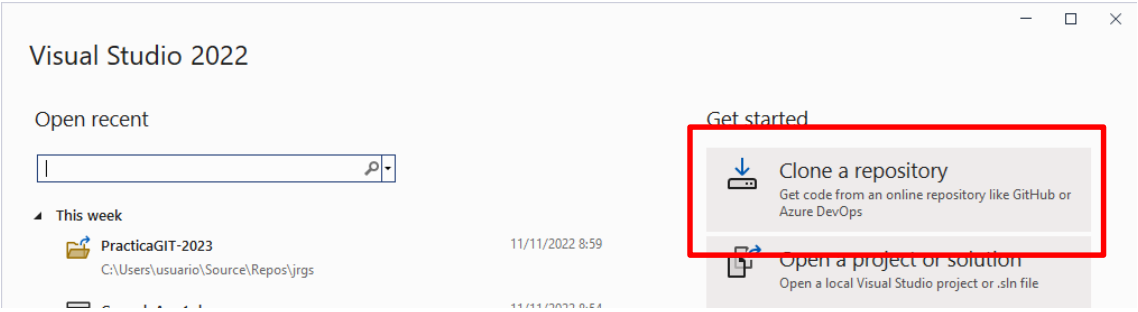
página en **Manage access**:

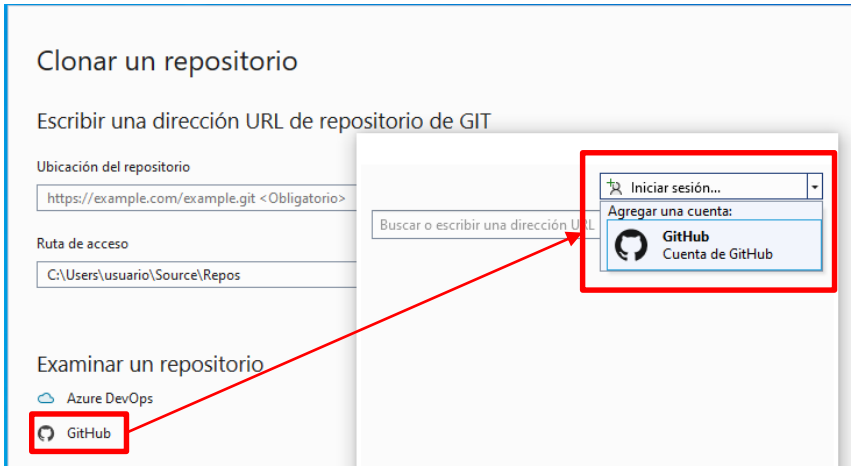


Aquí podemos buscar por el nombre de usuario (tiene que estar registrado). Una vez encontrados los añadimos (**Invite collaborator**) y se le enviará una invitación al colaborador (éste tendrá que confirmarla).

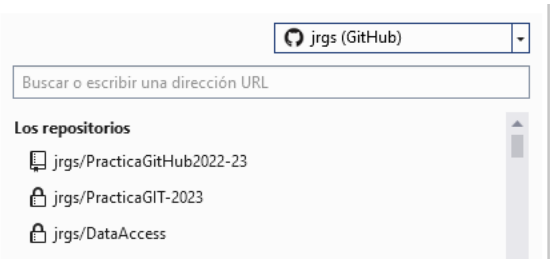
3. Creación de un proyecto GitHub con Visual Studio

La versión 2022 de Visual Studio ya trae incorporada toda la funcionalidad de GitHub, no siendo necesario instalar ninguna extensión, como ocurría con anteriores versiones. De hecho, una de las opciones que tendremos nada más arrancar el entorno será clonar un repositorio ya existente, simplemente proporcionando la URL del repositorio.

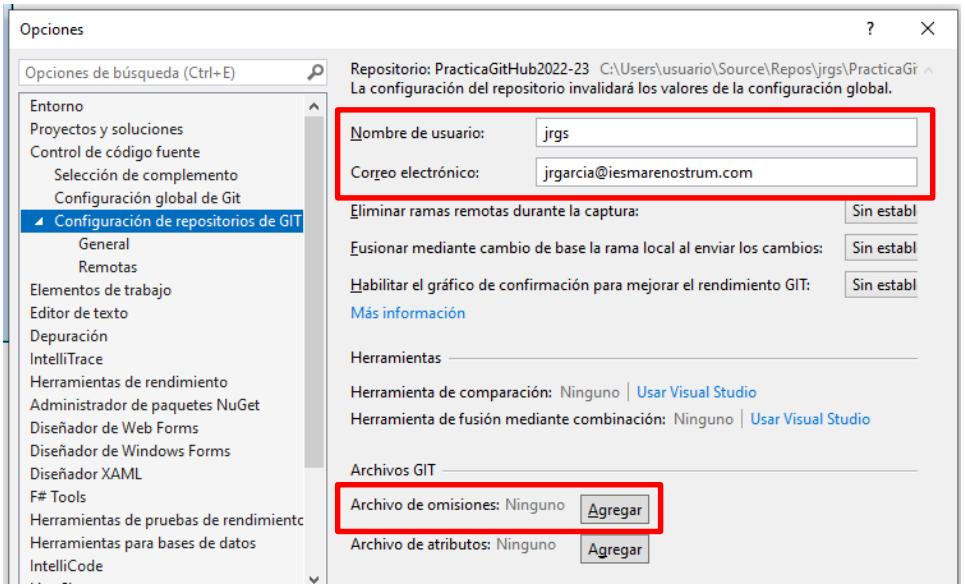




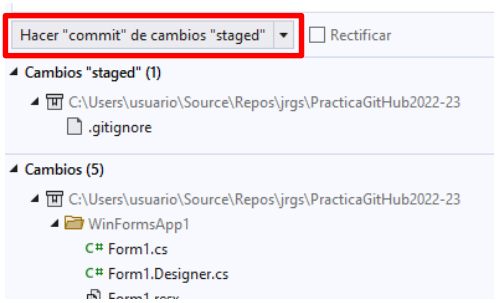
5. Una vez hayamos iniciado sesión (se abrirá una ventana del explorador), tendremos una lista con todos nuestros repositorios disponibles para ser clonados.

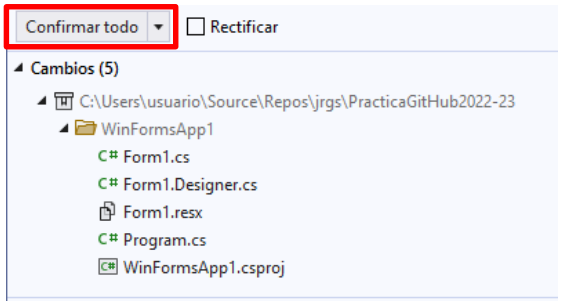


6. Una vez hayamos clonado el repositorio –que estará vacío-, deberemos de crear dentro del mismo una solución nueva de Aplicación de Windows Forms (.Net Framework)
7. En este punto, y para evitar posteriores problemas, es conveniente crear un archivo .gitignore (si no lo hemos hecho anteriormente). Este archivo le indica a Git qué tipos de ficheros se deben excluir de la sincronización. Normalmente se excluyen ficheros propios del IDE, que no suelen ser parte de nuestro proyecto. Para ello accederemos a la opción *Configuración/Configuración de repositorios*, dentro del plugin de Git, y crearemos automáticamente este fichero. También deberemos incluir (si no están ya) nuestro nombre de usuario y correo de GIT.

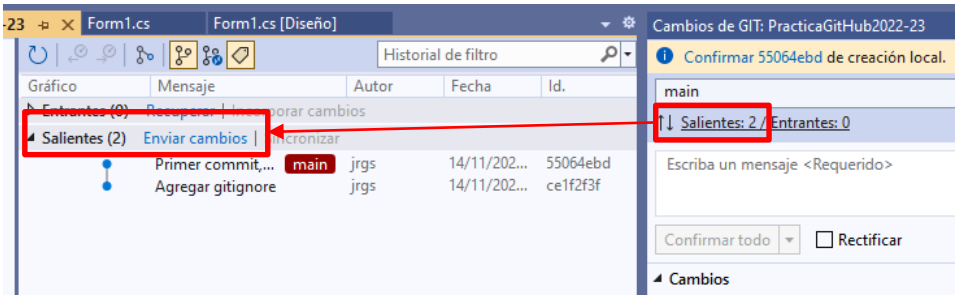


8. Si hemos añadido el archivo .gitignore nos pedirá confirmar el cambio 'staged' (temporal). Después, estaremos listos para hacer nuestro primer commit (con la solución vacía). No olvidéis antes de nada guardar también el archivo de solución (*.sln)

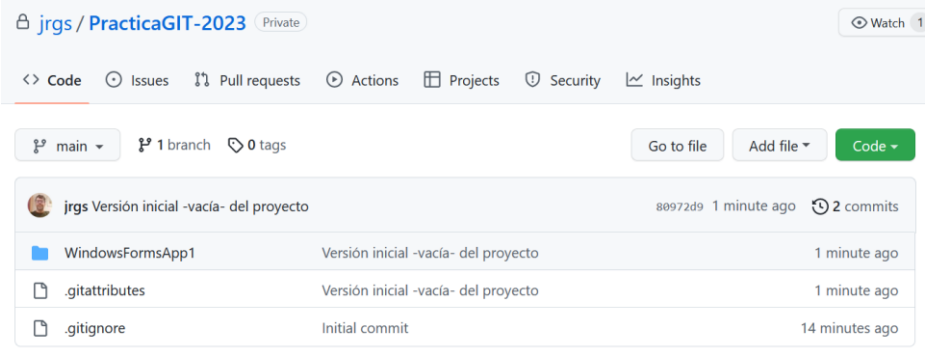




9. Al pulsar el botón *'Confirmar todo'* veremos que nos aparecen dos confirmaciones de salida pendientes de sincronizar en la ventana de cambios de Git. Procederemos a sincronizar (push).

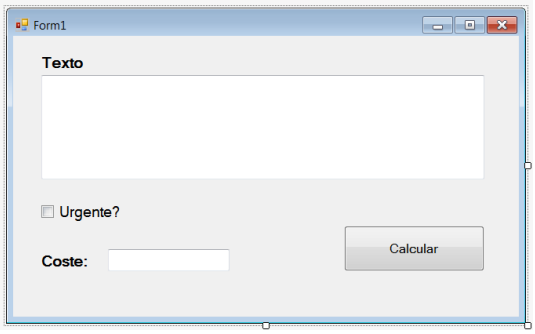


¡Ya está!. Podemos iniciar sesión en GitHub para comprobar que los cambios se han reflejado en el repositorio.

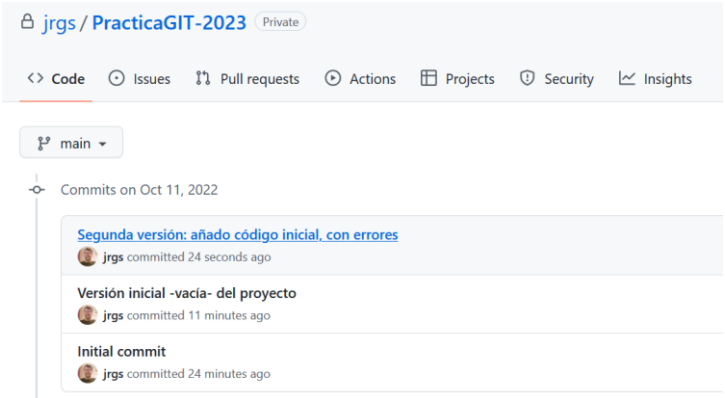


4. Simulación de un desarrollo en paralelo

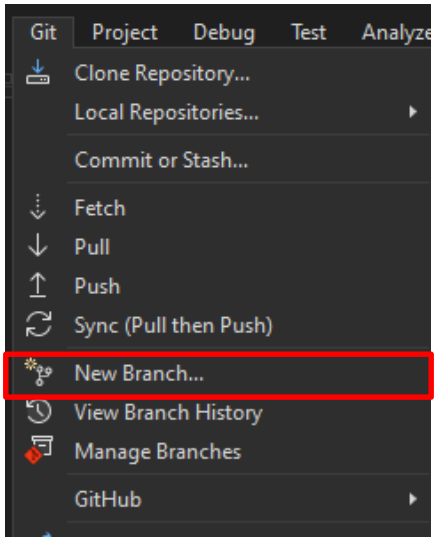
Una vez creado el repositorio, se agregará al proyecto el código que utilizamos en la práctica 2.3 (depuración), sin corregir ninguno de los errores que inicialmente había. También deberemos de crear el formulario.



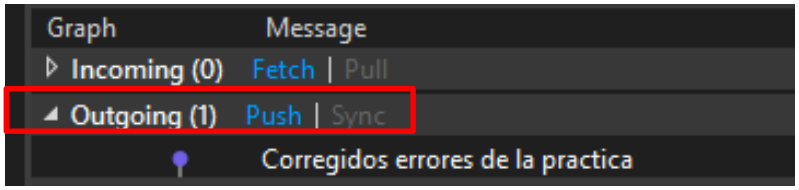
13. Cuando se haya comprobado que el código compila y ejecuta (aunque tenga errores), se deberá actualizar el repositorio remoto (commit + push).



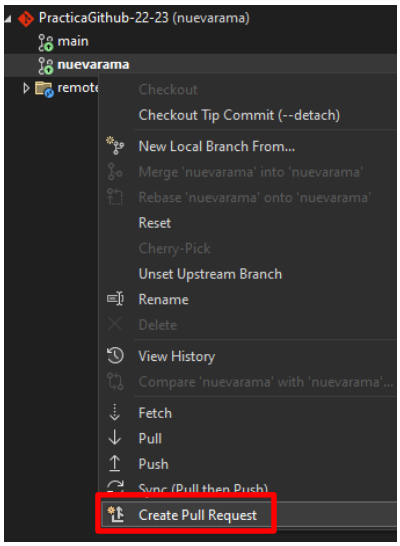
14. Una vez esté actualizado, el segundo miembro del grupo (o el mismo miembro, con una cuenta de correo diferente y en una máquina/usuario diferente) debe clonar el respositorio (para ello debe estar incluido como colaborador). Si utilizáis la misma máquina (caso de realizarlo en modalidad semipresencial), es fundamental que tengamos dos repositorios diferenciados, para poder simular adecuadamente el desarrollo en paralelo. Esto lo conseguiremos fácilmente usando dos usuarios distintos en la misma máquina.
15. El usuario 2 ahora creará una nueva rama del proyecto, haciendo su propia versión del mismo.



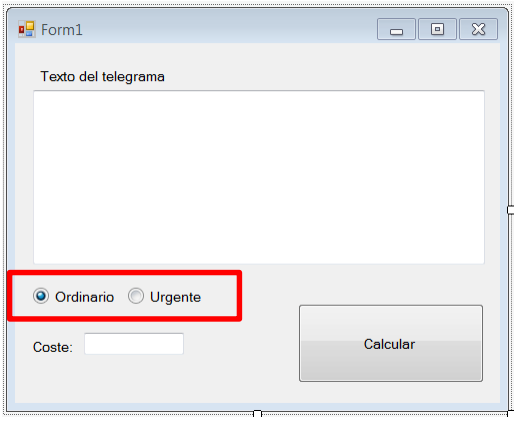
16. Una vez creada la nueva rama, modificaremos el código, corrigiendo los errores que se detectaron en su momento en la práctica 2.3.
17. Una vez hayamos comprobado que el código funciona correctamente (no hay errores lógicos) sincronizaremos los cambios con el repositorio remoto.



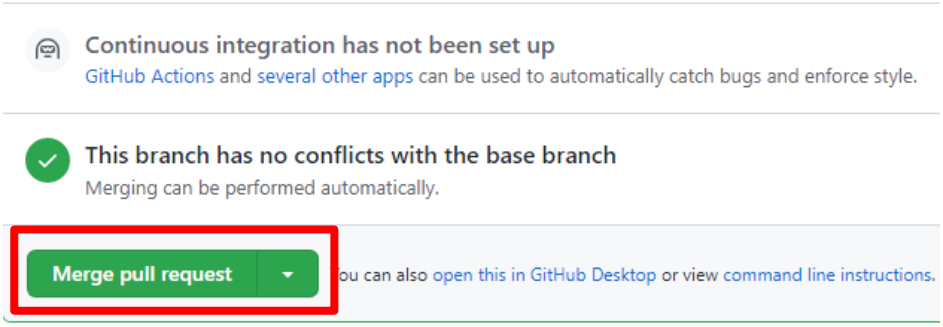
18. En este momento, en el flujo normal de trabajo, el usuario 2 debería informar al usuario 1 de que incorporara sus cambios a través de un 'Pull Request'. Si se omite este paso, el usuario 1 tendría un conflicto cuando intente incorporar sus cambios.



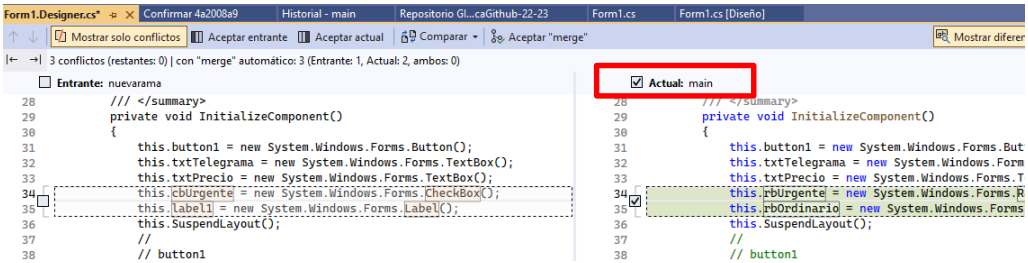
19. Ahora volvemos a cambiar al usuario1. Como paso previo, deberá comprobar las solicitudes de incorporación de cambios. Aquí comprobará que usuario2 ha realizado cambios que se deberían incorporar a la rama master. Vamos a ignorar esta incorporación de cambios, de momento. Lo que hará en su lugar el usuario1 será coger la rama master y cambiar el checkbox *cbUrgente* por un radiobutton (con dos opciones, urgente y ordinario). Así mismo, se deberá de modificar el código allí donde se requiera.



20. Una vez el código esté modificado y compile sin errores, haremos un commit local (sin hacer push). Antes de continuar, comprobaremos si tenemos alguna Pull Request pendiente (Git/GitHub/Ver Pull Request). Veremos que hay una Pull Request del usuario2, la incorporamos.



21. Si ahora le damos a sincronizar, veremos que nos aparecen los cambios entrantes y no nos deja automáticamente realizar la sincronización. Esto es debido a que la rama main ha incorporado los cambios de la Pull Request, pero nosotros todavía no los tenemos en local. Si intentamos enviar los cambios locales, nos dirá que hay que incorporar primero con Pull los remotos. Lo hacemos.
22. Si se produce un conflicto, deberemos proceder a solucionarlos. Para ello, elegimos la opción 'fusionar mediante combinación' en los archivos conflictivos y procedemos a indicar la versión con la que nos vamos a quedar.

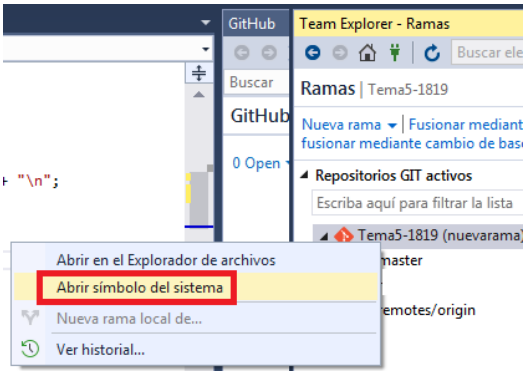


23. Una vez solucionados los posibles conflictos (no debería haber), debemos enviar todas los cambios locales al repositorio remoto (Push).
24. Para terminar la práctica, el usuario1 deberá hacer un Push, y el usuario2 deberá hacer un Pull, teniendo de esta manera reflejados los cambios.

El dossier deberá incluir un **pantallazo con el historial de commits**.

5. Trabajar con comandos de Git

A pesar de disponer de un complemento gráfico, también es posible trabajar desde Visual Studio con los comandos habituales de Git. Para ello, basta situarnos sobre el repositorio en cuestión con el que queremos trabajar y darle al botón derecho del ratón, eligiendo la opción *Abrir símbolo de sistema*.



Desde este símbolo de sistema se puede trabajar de la misma manera, utilizando los comandos de Git que hemos visto en la parte teórica. Os animo a ello.