

# Prácticas

## *Boletín Práctica 1*

# Introducción a Visual Studio y Azure DevOps

**Ingeniería del Software**

ETS Ingeniería Informática  
DSIC – UPV

**Curso 2020-2021**

## 1. Objetivo

El objetivo de la sesión de laboratorio es introducir al alumno a las principales funcionalidades de *Visual Studio 2019* y su relación con la herramienta de gestión de proyectos en equipo y control de versiones *Azure DevOps*<sup>1</sup>. El alumno tendrá que desarrollar obligatoriamente todas las actividades descritas hasta el punto 5 (inclusive). El punto 6 sobre ramificación y control de ramas es trabajo opcional.

**Nota:** Las capturas que aparecen en este boletín pueden corresponder a proyectos realizados en cursos diferentes.

## 2. Creación de un proyecto de equipo de *Azure DevOps* y de una solución inicial en el servidor

Para la realización de esta práctica es necesario haber seguido los pasos para la creación de un proyecto de equipo mediante *Azure DevOps* en la nube, haber dado de alta a los miembros del equipo de prácticas en ese proyecto y haber creado una solución en dicho proyecto. Es obligatorio utilizar el siguiente esquema de nombres para nombrar a la organización creada en servidor de *Azure DevOps*: 20120-upv-isw-<nombre grupo laboratorio>-<nombre equipo>. Por ejemplo, el servidor podría estar en **dev.azure.com/2020-upv-isw-3dl1-equipo01**

**Estas tareas previas las realizará un único miembro del equipo, el responsable del mismo o *Team Master*.**

La realización de estas actividades ha sido descrita el seminario del tema 3: *Visual Studio Integrado con DevOps y GIT*. Antes de pasar a los siguientes puntos se deben haber realizado esas actividades. Consulte el material existente en [Poliformat](#) relativo a este seminario y los videos disponibles en la pestaña vídeos del canal General ([Azure DevOps: Crear Organización](#) y [Visual Studio: Desarrollo del Proyecto SW](#)). Al finalizar estas tareas, el equipo de desarrollo tendrá un proyecto de equipo con una solución Visual Studio que contiene las carpetas de soluciones *Presentation*, *Library* (con carpeta para *BusinessLogic* y *Persistence*) y *Testing* (Ver Figura 1).

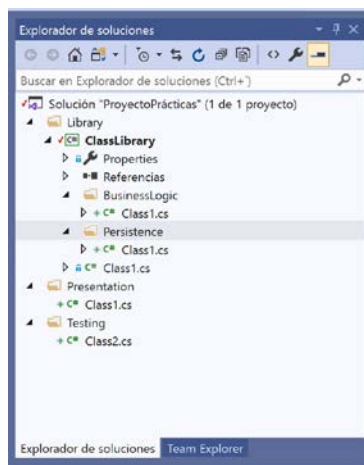


Figura 1. Solución inicial con las carpetas de soluciones

Recordar que todas las carpetas creadas deben tener al menos un elemento para que Git las sincronice adecuadamente en el repositorio remoto. Git no gestiona adecuadamente carpetas vacías, por lo que si subís al repositorio remoto una carpeta sin contenido, cuando el repositorio vuelve a clonarse, no es posible añadir contenido a estas carpetas debiendo ser eliminadas y vueltas a crear.

<sup>1</sup> Azure DevOps es la nueva nomenclatura de Microsoft para Team Services.

### 3. Conexión al proyecto existente desde Visual Studio

En este momento, cada miembro del equipo puede conectarse desde *Visual Studio* de forma **individual** al proyecto de *Azure DevOps*, descargar del servidor en la nube una copia de la solución y asignarla a un área de trabajo local (directorio local privado<sup>2</sup>) para trabajar con ella.

De esta manera cada alumno podrá realizar las siguientes actividades **de forma individual**. Para todo ello realice los siguientes pasos:

- Inicie la herramienta de desarrollo *Microsoft Visual Studio*.
- Aparecerá un diálogo, seleccione la opción *Continuar sin código*.

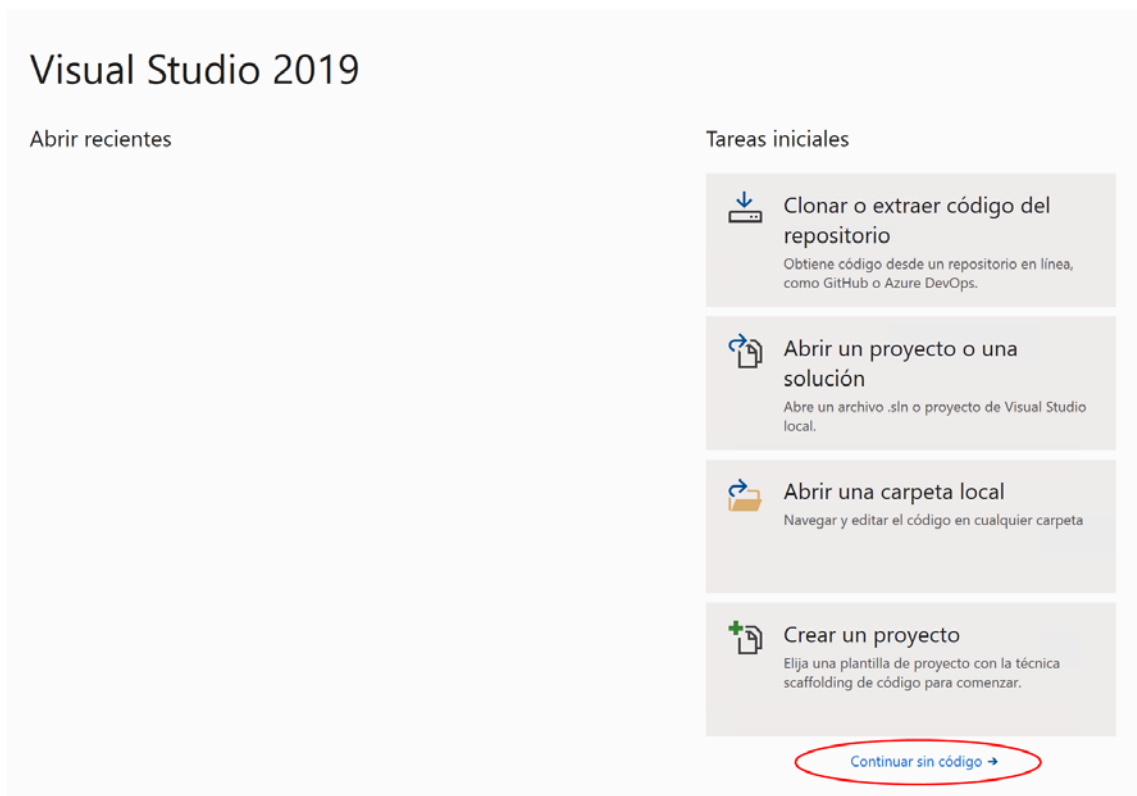


Figura 2. Seleccionar continuar sin código desde el diálogo inicial

<sup>2</sup> **Importante:** en los laboratorios del DSIC y accesos al entorno virtual, se **recomienda** utilizar el **directorio privado** de cada usuario a partir de la ruta C:\Users\nombreUsuario\... pero **no** utilizar la unidad W:, que es una unidad de red en la que puede perderse la conexión, produciendo fallos en la sincronización y errores con Visual Studio.

- c. Inicie sesión con la cuenta de usuario Microsoft asignada al proyecto:
- Archivo->Configuración de La cuenta->Iniciar Sesión

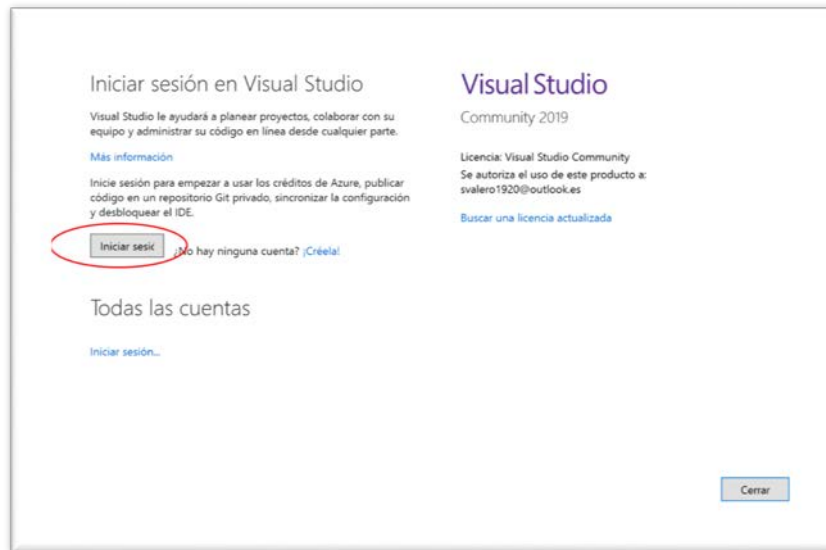


Figura 3. Iniciar sesión en Visual Studio

- d. Conéctese a su proyecto de equipo. Seleccione *Team Explorer* (ventana de la derecha) y a continuación seleccione el botón de *Administrar conexiones* (icono del enchufe de color verde). Vea la imagen Figura 3.

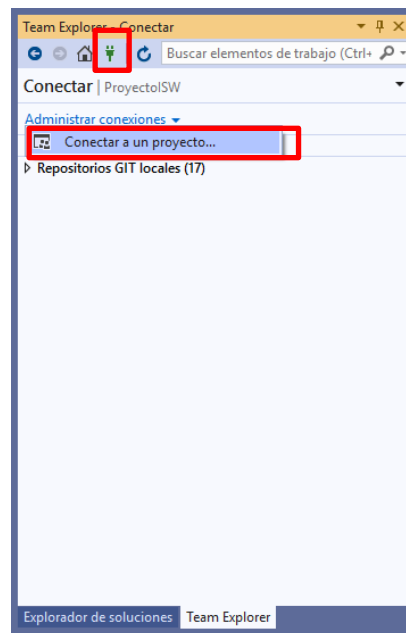


Figura 4. Conectarse a un proyecto desde Administrar Conexiones

- Una vez pulsado en el menú contextual de la figura anterior le aparecerá una ventana similar a la de la Figura 5. En la pestaña (Figura 5-1), aparece la cuenta Microsoft con la que habéis iniciado sesión. Bajo, aparece la relación de organizaciones a las que pertenecéis (en vuestro caso, es normal que solo aparezca una). Para cada organización, aparecen los proyectos que tiene (de nuevo, solo uno en vuestro caso). Seleccione el símbolo de git de vuestro proyecto (Figura 5-2), confirme la ruta en la

que desee guardar el proyecto (Figura 5-3 y pulse a clonar (Figura 5-4). Es posible que deba confirmar con que cuenta va a conectarse al repositorio remoto.

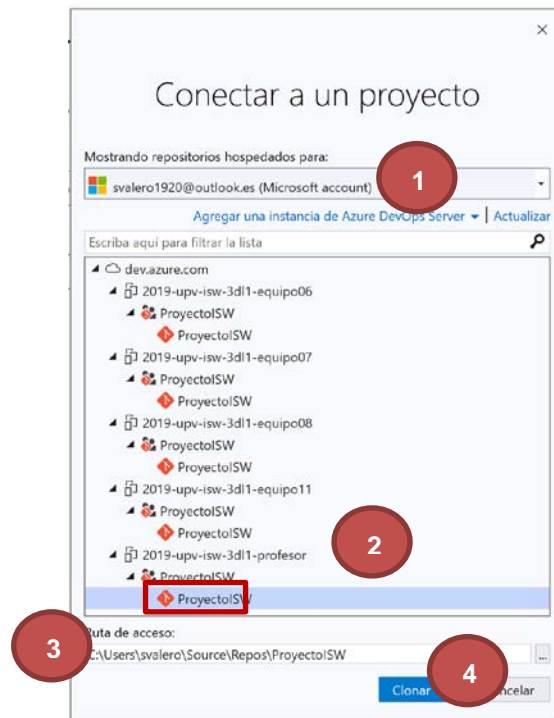


Figura 5. Conectarse a un proyecto de una organización

- e. Abra la solución creada por el *Team Master*. Una vez se ha conectado al proyecto de equipo, ha obtenido y asignado su área de trabajo local puede abrir la solución que el *Team Master*

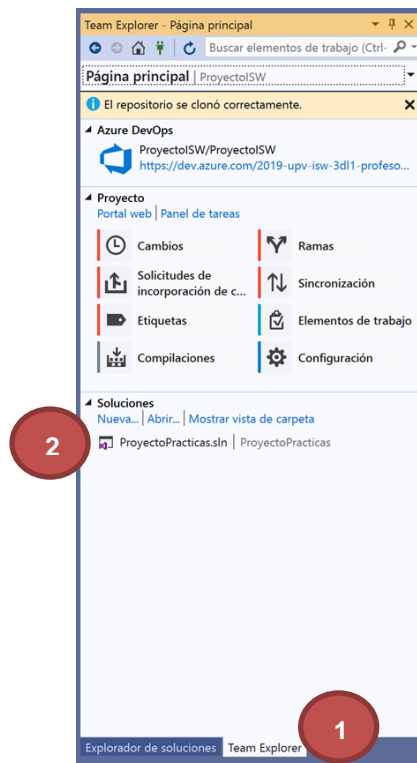


Figura 6. Abrir una solución desde el Team Explorer

ha creado previamente (en el punto 2 de esta guía). Desde el Team Explorer, haga doble click sobre la solución que se encuentra en la sección soluciones (Figura 6-2)

- f. Vaya al explorador de soluciones (pestaña “*Explorador de soluciones*” en la parte derecha inferior de *Visual Studio*) y compruebe que tiene una solución como la que el *Team Master* ha creado en el servidor originalmente, tal y como se mostraba en la Figura 1.

#### 4. Introducción a las herramientas de depuración de Visual Studio

Las herramientas de depuración de *Visual Studio* son un elemento fundamental para el desarrollo de aplicaciones en *Visual Studio* y la detección de *bugs* en el código. Estas herramientas permiten establecer puntos de ruptura en el código, ejecutar código paso a paso, inspeccionar valores de variables en tiempo de ejecución, etc.

##### 4.1 Creación de un proyecto de consola

Para iniciarnos en las labores de depuración vamos a **crear un proyecto de consola en la carpeta de soluciones *Testing***. Para ello:

- a. Cree una carpeta en *Testing* denominada *Lab1*
- b. Pulse el botón derecho del ratón en carpeta *Lab1* -> *Agregar* -> *Nuevo Proyecto*-> *Aplicación de consola(.Net Framework)* y llamar a la aplicación ***HelloWorldApp***

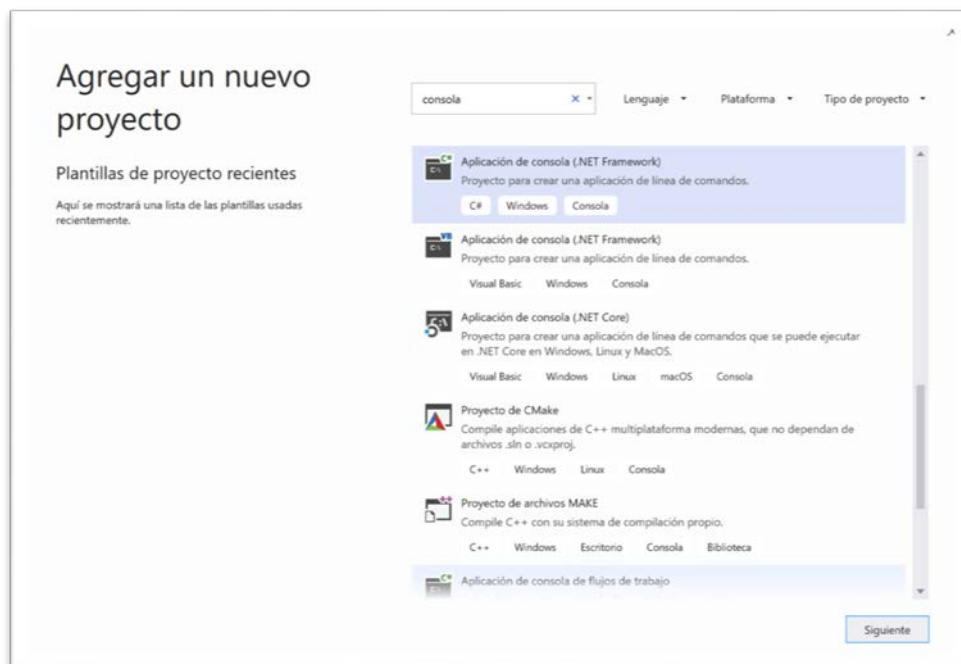


Figura 7. Creación aplicación de consola HelloWorldApp

- c. Una vez generada la aplicación observe las distintas ventanas que *Visual Studio* le proporciona (ver Figura 8): cuadro de herramientas (izquierda), explorador de soluciones (derecha arriba), propiedades (derecha abajo), editor de código (centro arriba), listas de errores y salida (centro abajo).
- d. En el caso de que el proyecto que acaba de crear no aparezca en el explorador de soluciones resaltado en negrita (proyecto de inicio) tendrá que indicarle a *Visual Studio* que este es su proyecto de inicio actual a efectos de compilación y ejecución. Para ello pulse el botón derecho del ratón sobre el proyecto *HelloWorldApp* y seleccione la opción de menú ***Establecer como proyecto de inicio***. Observe que el nombre del proyecto ha pasado a estar resaltado en negrita.

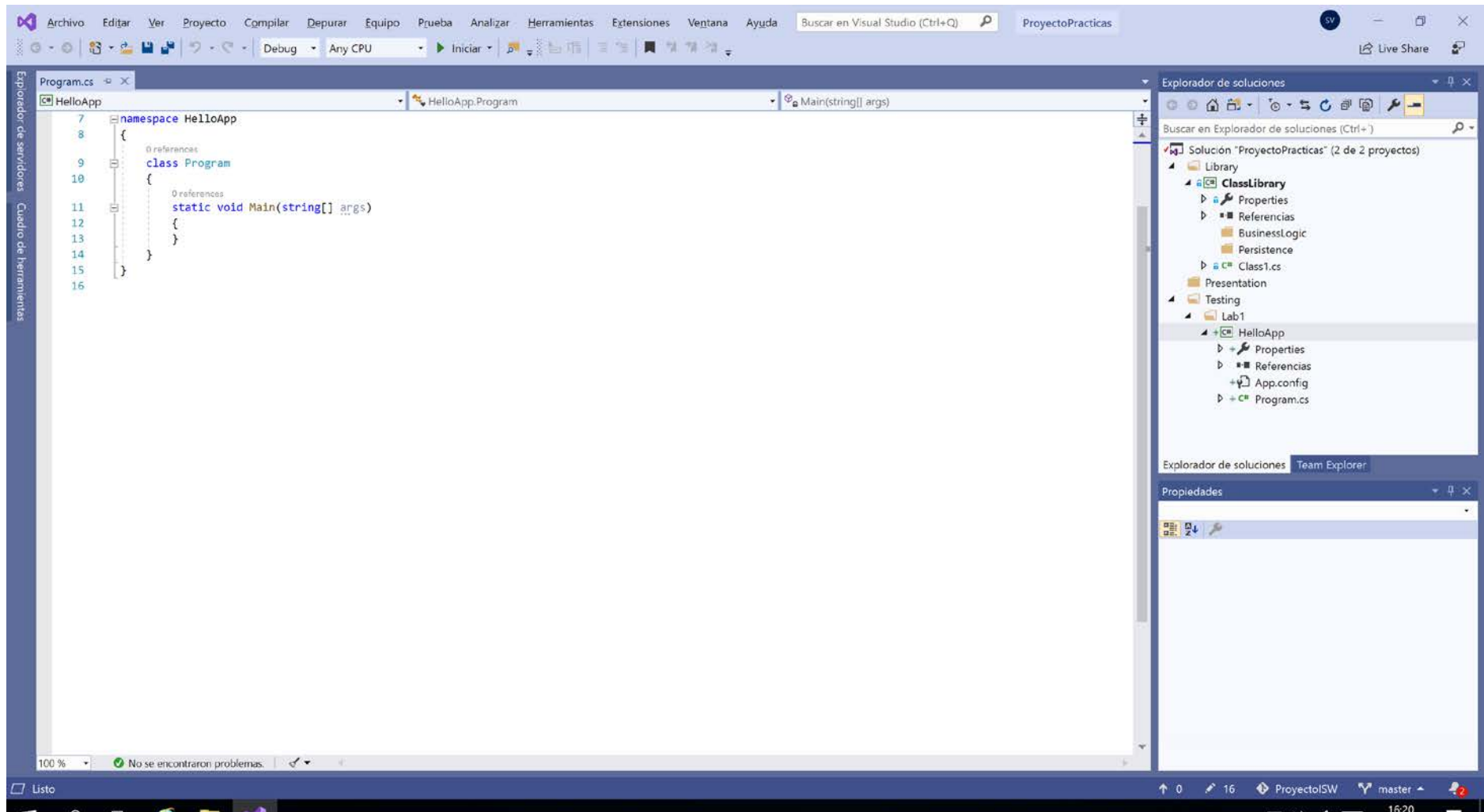
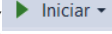


Figura 8. Ventanas típicas de trabajo de Visual Studio

## 4.2 Compilar y ejecutar

Para compilar y ejecutar esta aplicación pulse el triángulo verde () en la zona superior de la herramienta de desarrollo. Por la ventana de salida observará el resultado de la compilación y de la ejecución. Aunque el programa actual todavía no realiza ninguna acción, sí se podrá observar cómo se abre y se cierra una ventana de consola.

## 4.3 Exploración de errores de compilación

Abra el fichero *Program.cs* e introduzca un error artificialmente en el método *main*, por ejemplo, escribiendo la sentencia “*fadsfsde*”. Si vuelve a compilar su programa observará la lista de errores en la ventana correspondiente. Si hace click sobre el error, el cursor aparecerá en la zona de su programa donde se encuentra el mismo.

## 4.4 Creando una aplicación HelloWorld en C#

Para poder practicar las capacidades de depuración vamos a generar un sencillo programa de ejemplo. Abra el fichero *Program.cs* y modifique su contenido para que quede como sigue.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace HelloWorldApp
{
    class Program
    {
        private static List<string> nacionalities;
        private static void InitList()
        {
            nacionalities = new List<string>()
            {
                "Australian",
                "Mongolian",
                "Russian",
                "Austrian",
                "Brazilian"
            };
        }

        static void Main(string[] args)
        {
            InitList();
            nacionalities.Sort();
            foreach (string nationality in nacionalities)
            {
                Console.WriteLine(nationality);
            }
        }
    }
}
```

Observe que el programa anterior tiene unas sentencias *using* que actúan como directivas del compilador, haciendo que se puedan utilizar tipos definidos en el espacio de nombres importado sin necesidad de especificarlos de forma explícita en el código. A continuación, se define un espacio de nombres *HeLlOWorLdApp* y en él la clase *Program*. Esta clase contiene la definición de un atributo privado de tipo lista de strings (*List<string>*), un método estático de inicialización del mismo y el método estático *main* que es el que iniciará la aplicación. El código propuesto es relativamente sencillo de entender y comparte sintaxis con código Java. Concretamente se está



creando una lista de *strings*, se ordena alfabéticamente (método *Sort()*) y luego se recorre (bucle *foreach*) y se visualiza su contenido por pantalla. Este código será usado para practicar las capacidades de depuración. Compile y ejecute el programa. De nuevo verá que ha aparecido una ventana de consola con contenido, se ha ejecutado el programa rápidamente pero no ha podido observar la salida generada por el mismo.

#### 4.5 Ejecución de un programa paso a paso

En la mayoría de IDEs actuales es posible incluir puntos de ruptura donde se detiene la ejecución para poder, desde ese punto concreto, ejecutar sentencia a sentencia el código e ir observando su comportamiento de forma controlada. Para practicar esta característica realizaremos las siguientes acciones:

- Inserte un punto de ruptura en la línea de código *InitList()*. Para ello pulse sobre el ratón en la columna de color gris que aparece a la izquierda del código, a la altura de la línea donde desea interrumpir la ejecución (ver Figura 9)

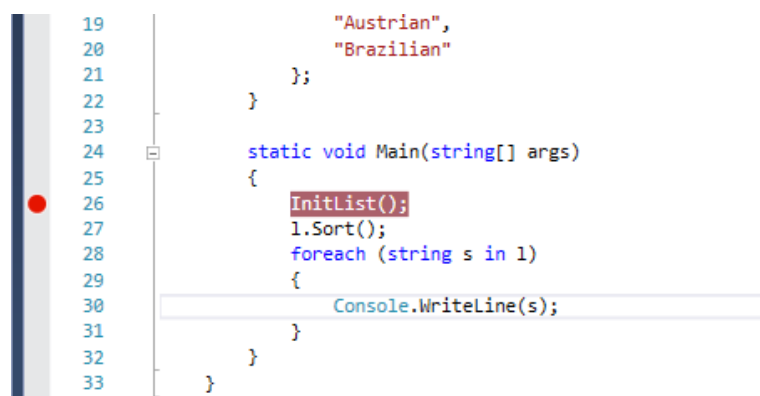


Figura 9. Inserción de punto de ruptura

- Compile e inicie la depuración del código (pulse el triángulo verde) y observe como la ejecución se detiene en la línea donde se introdujo el círculo rojo (aparece una flecha amarilla indicando donde se encuentra la ejecución y cuál será la siguiente instrucción a ejecutar). Si observa la ventana de consola creada, ésta aparecerá (de momento) vacía.
- Ejecute paso a paso la aplicación. Para ello utilice los controles de ejecución paso a paso (ver Figura 10). Puede detener la ejecución (cuadrado rojo), ejecutar la siguiente instrucción (Figura 10-1 o F11), ejecutar todo un procedimiento/método de forma completa sin entrar en él (Figura 10-2 o F10) o ejecutar todas las sentencias que hay en un procedimiento para salir de él (Figura 10-3 o Mayúsc+F11).

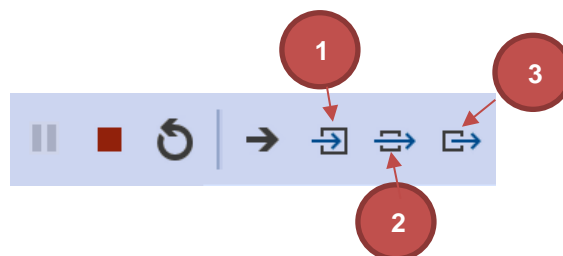


Figura 10. Ejecución paso a paso

#### 4.6 Observando los valores de datos del programa

Mientras el programa se encuentra en ejecución es posible observar los valores de los objetos, atributos, variables, etc. Esto es **extremadamente útil** cuando está depurando su programa. Para observar el valor de cualquier elemento sitúe el cursor sobre el mismo.

- Cuando su programa llegue a la ejecución de la sentencia `nacionalities.Sort()` ponga el cursor sobre la variable `nacionalities` y observe su contenido (ver Figura 11).

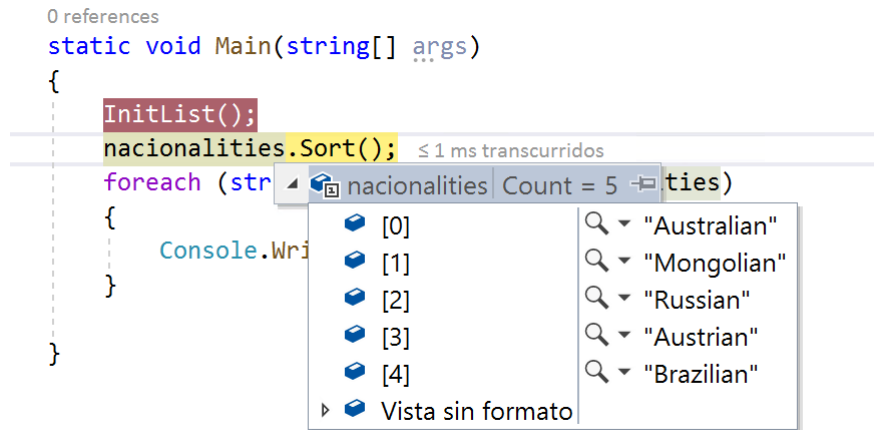


Figura 11. Inspeccionando valores de datos durante la ejecución

- Ejecute esa sentencia paso a paso (F11) y vuelva a observar la lista comprobando que se ha ordenado alfabéticamente.
- Siga ejecutando paso a paso su programa y observe cómo por la ventana de consola aparece el contenido de la lista (Figura 12).

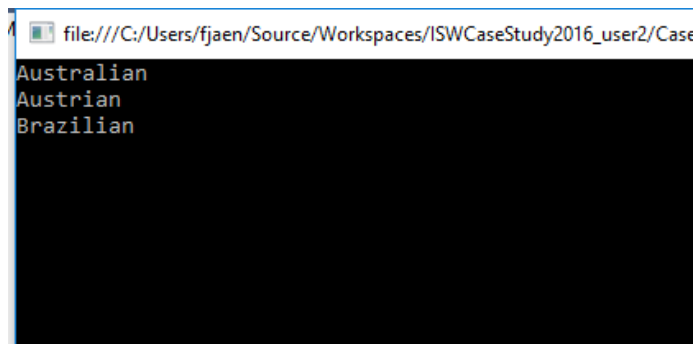


Figura 12. Salida por consola durante la ejecución

- Puede volver hacia atrás en la ejecución simplemente arrastrando la flecha amarilla hacia arriba a la línea de código desde donde desee repetir la ejecución de su programa. Esto puede resultar muy útil pues nos permite cambiar el flujo original de nuestro código en tiempo de ejecución.

#### 5. Operaciones básicas de control de versiones

Una de las ventajas de utilizar un entorno de desarrollo en equipo como *Azure DevOps* es la posibilidad de tener control y gestión sobre las distintas versiones del proyecto que se vayan generando. De hecho, cuando creó su proyecto indicó que el sistema de control de versiones a utilizar sería **Git**. Un sistema de control de versiones es una combinación de tecnologías y prácticas para controlar los cambios realizados en los distintos elementos que componen el proyecto, en particular en el código fuente, documentación, conjuntos de pruebas, etc.

Existe una nomenclatura específica a Git, que mencionaremos a continuación, dado que es el sistema de control de versiones que utilizaremos:

- **Repositorio remoto:** el repositorio que reside en el servidor de Microsoft Azure DevOps y en el cual todos los miembros del equipo pueden almacenar sus cambios.
- **Repositorio local:** contiene una copia del repositorio remoto donde un miembro del equipo puede trabajar localmente en su máquina, pudiendo no estar sincronizado en todo momento con el repositorio remoto.
- **Commit:** Almacena los cambios realizados localmente en el repositorio local. Guarda información sobre los archivos que han sido modificados, información sobre quién ha hecho el cambio y un mensaje (en este caso obligatorio) que describa la naturaleza del cambio.
- **Clone:** Obtiene una copia en local del Proyecto a partir de repositorio remoto. En la copia local es posible observar todos los cambios que se han llevado a cabo desde el inicio del proyecto.
- **Pull:** Obtiene los últimos commits desde el repositorio remoto y los almacena en el repositorio local.
- **Push:** Sube los commits almacenados en el repositorio local al repositorio remoto.
- **Merge:** Es el proceso mediante el cual diferentes commits se combinan para obtener una nueva versión del Proyecto.
- **Conflict:** Puede surgir cuando dos o más miembros del equipo han realizado cambios distintos en el mismo fragmento de código. Cuando los diferentes commits se combinan en un proceso de mezcla (merge), por ejemplo como parte de una operación pull, y se detecta que existen cambios en el mismo archivo, el sistema le pide al usuario con qué cambio se quedará para que sea almacenado. Como resultado de la resolución del conflicto obtendremos un nuevo commit y una nueva versión coherente del proyecto.

El control de versiones **Git** sigue un flujo de trabajo que la mayor parte de los desarrolladores siguen cuando escriben código y lo comparten con el equipo de desarrollo. Los pasos son los siguientes:

1. Obtener una copia en local del código del proyecto
2. Hacer cambios en el código
3. Una vez que se han terminado los cambios hacer público el código modificado al resto del equipo
4. Una vez aceptado, fusionar con el código en el repositorio del equipo.

Gráficamente el anterior flujo de trabajo puede representarse de la siguiente forma:

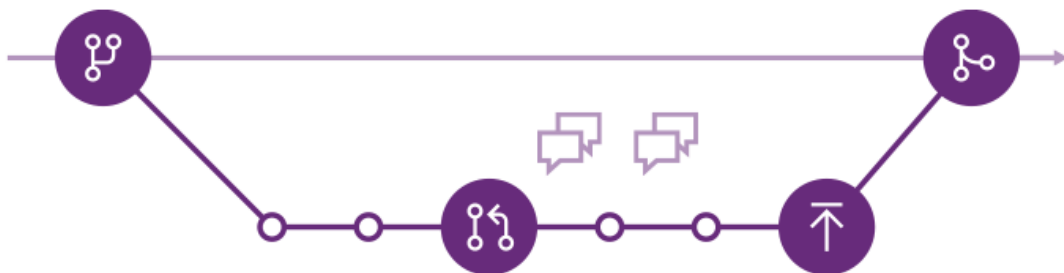


Figura 13 Flujo de trabajo de Git

A continuación introduciremos un conjunto de términos y comandos asociados como *repositories*, *branches*, *commits*, and *pull requests* que son comunes a diferentes gestores de control de

versiones como *Team Foundation Version Control* o *Subversion*. Hay que indicar que los comandos se comportan de manera ligeramente diferente en *Git*.

- Crear una rama. Las ramas se utilizan para los cambios que el desarrollador quiere llevar a cabo en local y llevan obligatoriamente un nombre.
- Commit cambios a tu rama local. En local podemos realizar todos los cambios que consideremos oportunos a nuestra rama.
- Push a la rama al repositorio remoto.
- Crear un pull request de forma que el resto del equipo pueda revisar tus cambios.
- Completar el pull request y resolver cualquier conflicto generado por cambios que otros han realizado sobre tu rama.

La creación de ramas la actualización y la mezcla entre las mismas se explica al final del documento.

A continuación simularemos un escenario típico de funcionamiento, sobre la copia local del proyecto cambiaremos el contenido de algún fichero, usaremos la opción cambios y posteriormente confirmaremos los mismos. Con esto nuestras modificaciones se incorporan al repositorio local. Utilizando la opción Sincronización subiremos los cambios al repositorio remoto y resolveremos los posibles conflictos que surjan.

Sobre el proyecto de depuración recientemente creado llevaremos a cabo las siguientes actividades:

- a) Modifique cualquier parte de su programa. Por ejemplo, la clase *Program* para que el primer elemento de la lista sea otro (cada miembro del equipo puede elegir una cadena diferente (*Italian*, *Spanish*, etc.))
- b) Vaya a Team Explorer -> Cambios (añada un comentario descriptivo de la nueva versión o modificación introducida).

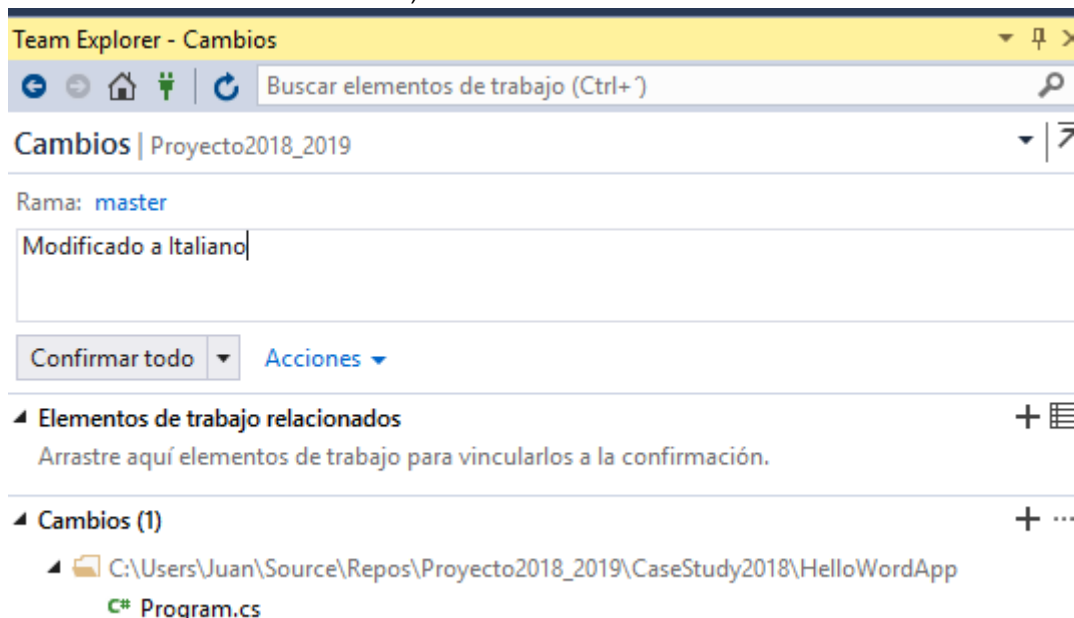


Figura 14 Ventana para subir modificaciones al repositorio central

Al pulsar **Confirmar todo** le aparecerá la siguiente ventana que aparece en la Figura 15:

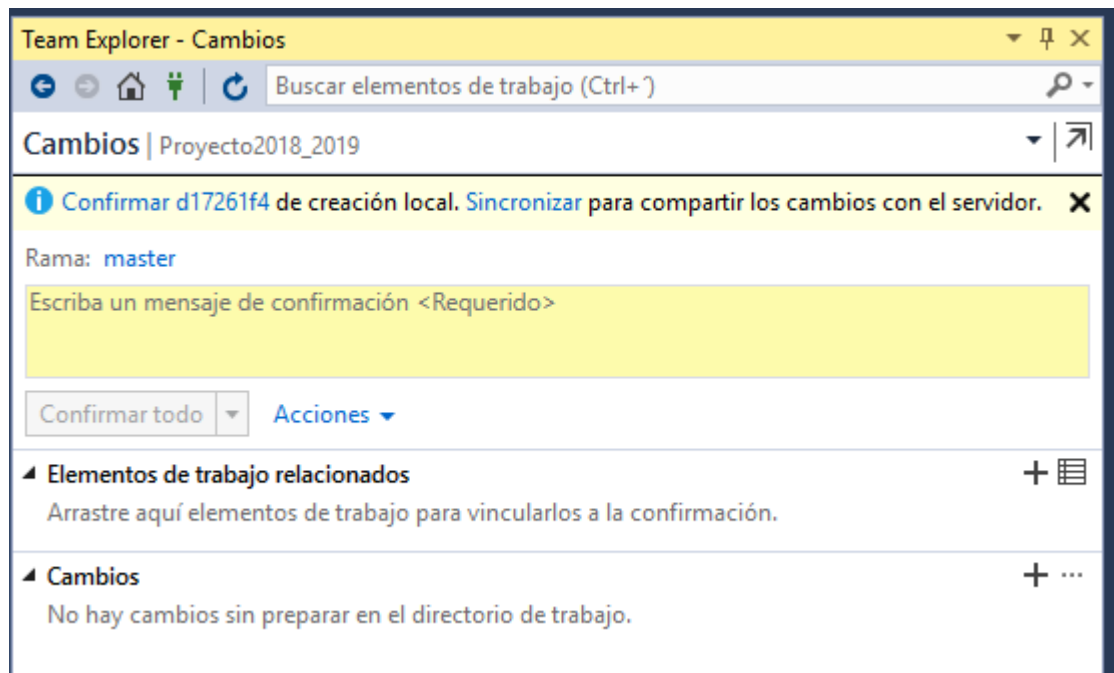


Figura 15 Ventana tras pulsar Confirmar todo

Pulse ahora sobre Sincronizar. El resultado de la operación se muestra en la Figura 16.

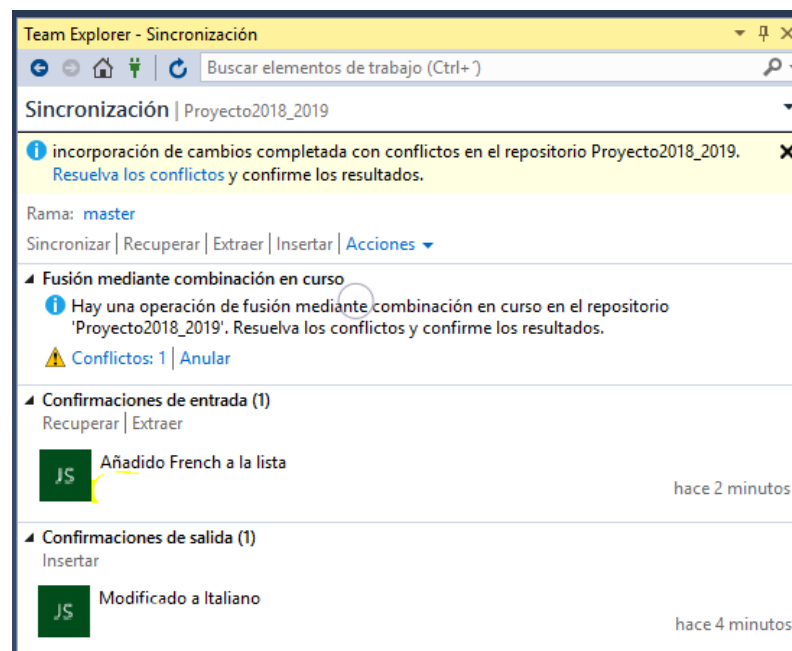


Figura 16 Conflictos al realizar push/pull (sincronizar)

- c) Resolución de conflictos: es posible que, si otro usuario generó una nueva versión antes que usted o bien después y usted no descargó la última versión existente en el repositorio, se produzcan conflictos que haya que resolver. La resolución de conflictos se puede hacer de forma manual decidiendo qué código conservar en la nueva versión a generar mediante la herramienta de resolución de conflictos (ver Figura 18). Pulse en Conflictos de la figura anterior

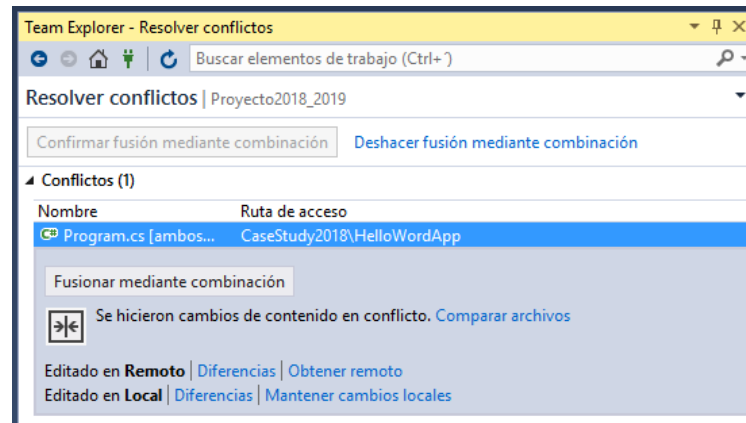


Figura 17 Resolución de conflictos

Pulsando en Comparar archivos de la ventana anterior podemos observar las diferencias entre el archivo remoto (servidor) y el archivo local.

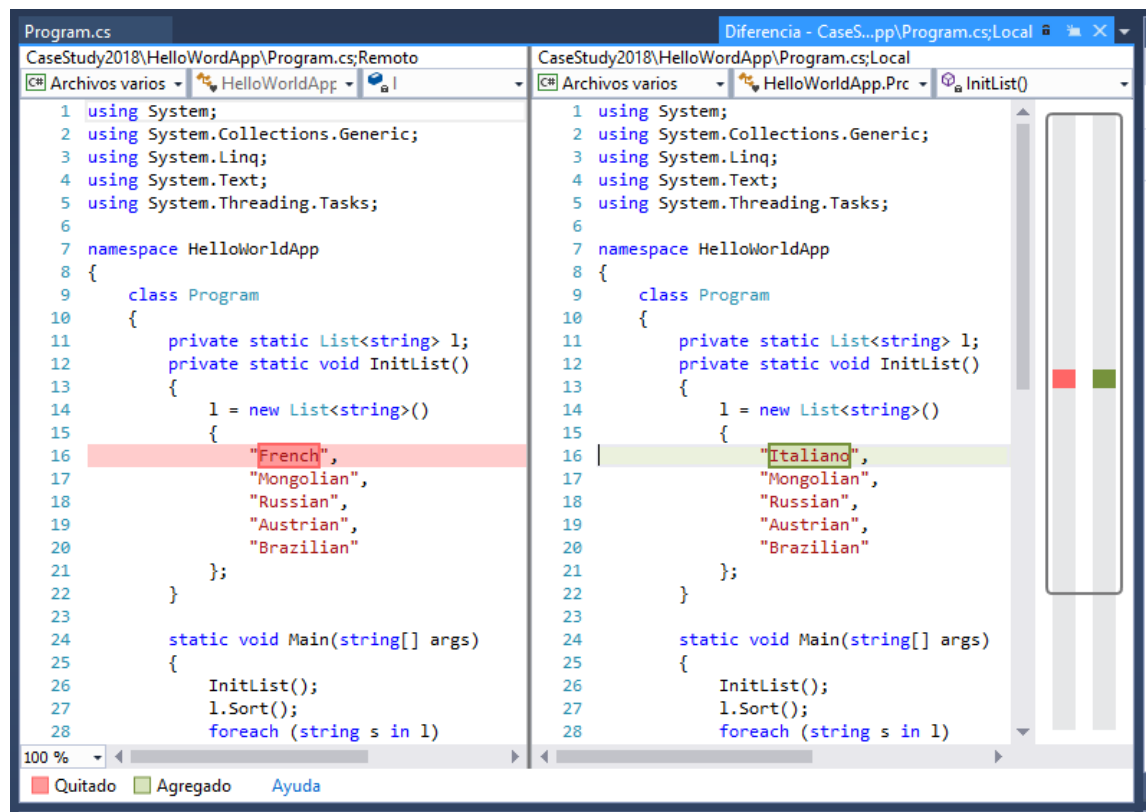


Figura 18 Ventana de resolución de conflictos

- e. Los conflictos surgen por la diferencia de contenido en la línea 16 del código. Se ha de decidir qué versión es la correcta. Como se puede observar en la Figura 18 tenemos una versión con un elemento en la lista ("Italiano"), y otra versión con un elemento ("French"). Seleccionamos como correcta una de las versiones y finalizamos la operación. Al hacer esto incorporaremos los cambios al repositorio central generando una nueva versión del proyecto. También podemos seleccionar Fusionar mediante combinación, en cuyo caso nos aparece una nueva ventana como la de la figura siguiente.

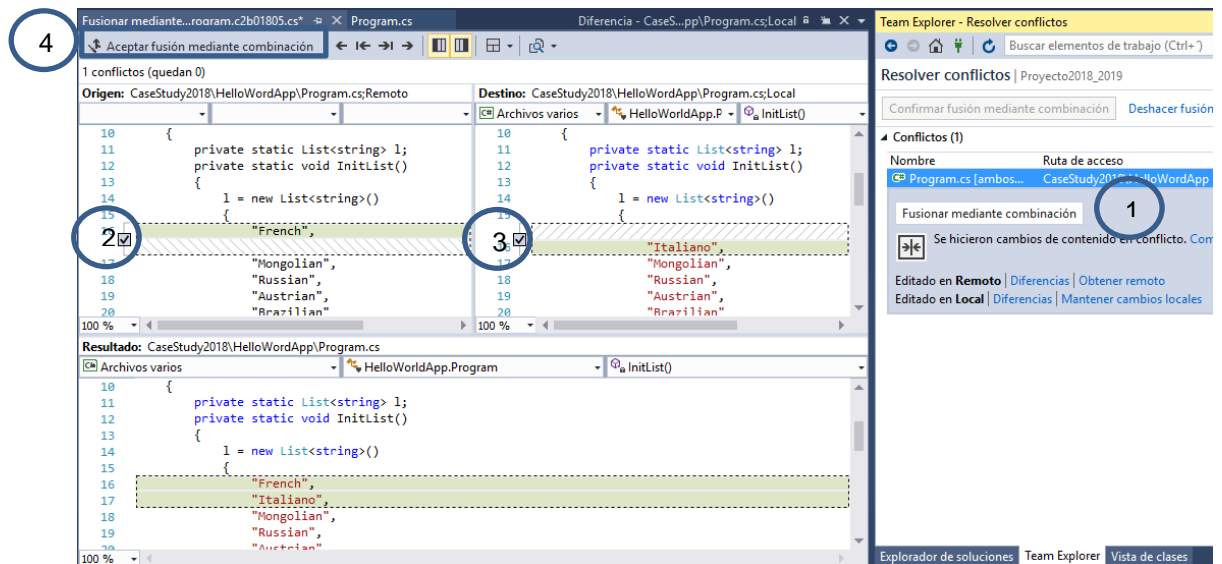


Figura 19. Resolución de conflictos en las versiones de código

- f. Hemos marcado en la figura precedente que deseamos mantener el cambio French y el cambio Italiano en la lista de países. El resultado del fichero combinado aparece en la parte inferior de la figura anterior. Para finalizar pulsamos sobre Aceptar fusión mediante combinación.
- g. Puede observar el historial de cambios de cualquier elemento de su proyecto mediante el Explorador de Versiones de código. Vaya a Explorador de soluciones -> Seleccione en el explorador el elemento a explorar (por ejemplo el fichero Program.cs) -> Botón derecho de ratón -> Ver historial
- h. Análogamente se pueden observar los diferentes conjuntos de cambios generados en la Web asociada a su proyecto de Azure DevOps. Para ello, seleccione la opción Repos>Commits. Seleccionando un commit, se muestra qué fichero y ficheros han sido modificados y cuáles son los cambios realizados.

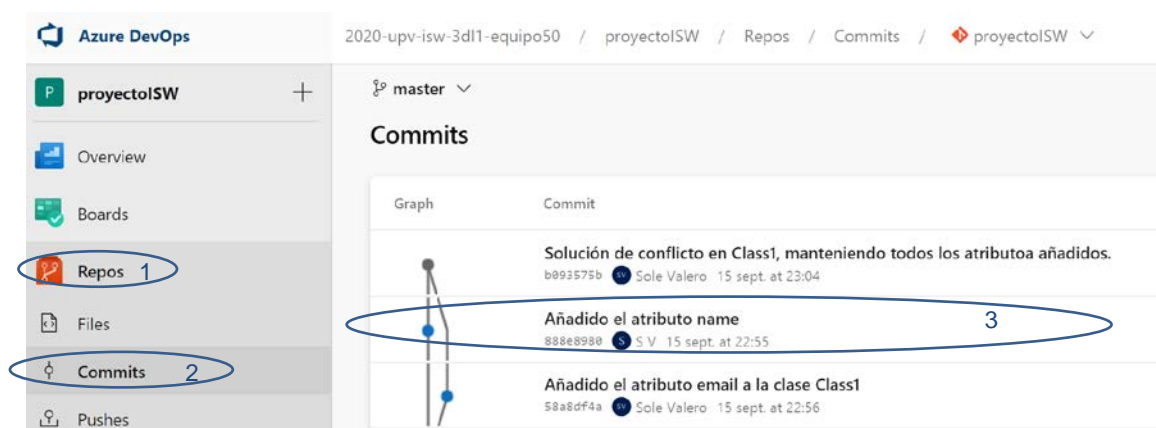


Figura 20 Visualización de la historia de cambios en el servidor

## 6. (Trabajo Opcional) Operaciones Avanzadas con el Explorador de control de versiones

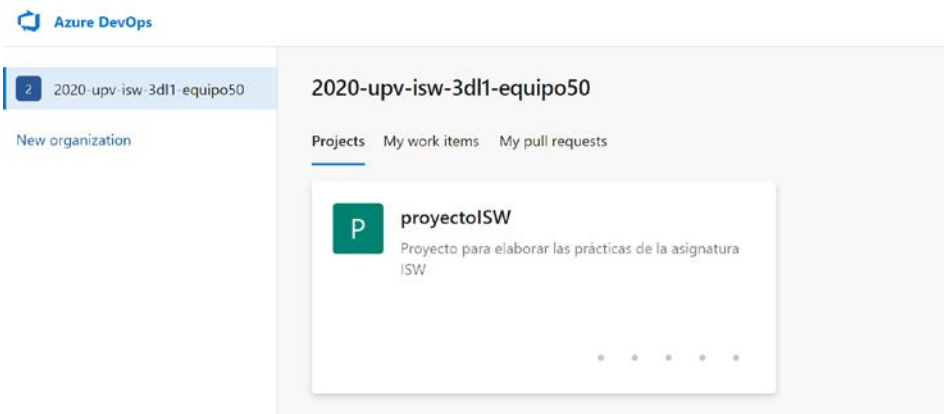
En esta sección practicaremos dos conceptos adicionales relativos al control de versiones:

- **rama (branch):** es una copia del proyecto, bajo el control de versiones, pero aislado, de forma que los cambios realizados en una rama no afecten al resto del proyecto y viceversa. Las ramas también son conocidas como "líneas de desarrollo".
- **merge:** mover un cambio de una rama a otra.

### 6.1 Generando una línea de desarrollo (rama)

Para generar una línea de desarrollo propia de forma que independice los cambios realizados por usted de los realizados por otros se puede proceder de la siguiente manera:

- a. El *Team Master* ha de conceder permisos de creación de ramas a los usuarios de su equipo que estén autorizados. Para ello, en la Web de gestión del proyecto de *Azure DevOps* irá al proyecto actual, pulse sobre el símbolo "P":



Y a continuación en "Project settings"> Repos> Repositories para añadir permisos específicos de acceso para un grupo o un miembro específico del equipo. Por defecto, los primeros 5 miembros invitados a una organización forman parte del grupo de desarrollo que crea Azure por defecto, que a su vez forma parte del grupo "Contributors". Todos los miembros de ese grupo pueden crear nuevas ramas (ver Figura 21).

- b. Si necesita, es posible asignar permisos de forma individual, aunque se recomienda la gestión de permisos a través de los grupos predefinidos. Para ello, seleccione al usuario indicando su cuenta en el campo de edición "Search for users or groups" de la Figura 21. En este ejemplo (ver Figura 22), se ha seleccionado al usuario [svalero1819@outlook.es](mailto:svalero1819@outlook.es), que ya estaba agregado con anterioridad al equipo de desarrollo del proyecto. Aparecerán los permisos asignados al usuario. Estos permisos pueden cambiarse pulsando en el panel derecho. Tenga en cuenta que algunos se heredan y no hace falta modificarlos (aparecen marcados como *inherited*). Conceda permisos de gestión de ramas a los usuarios que desee



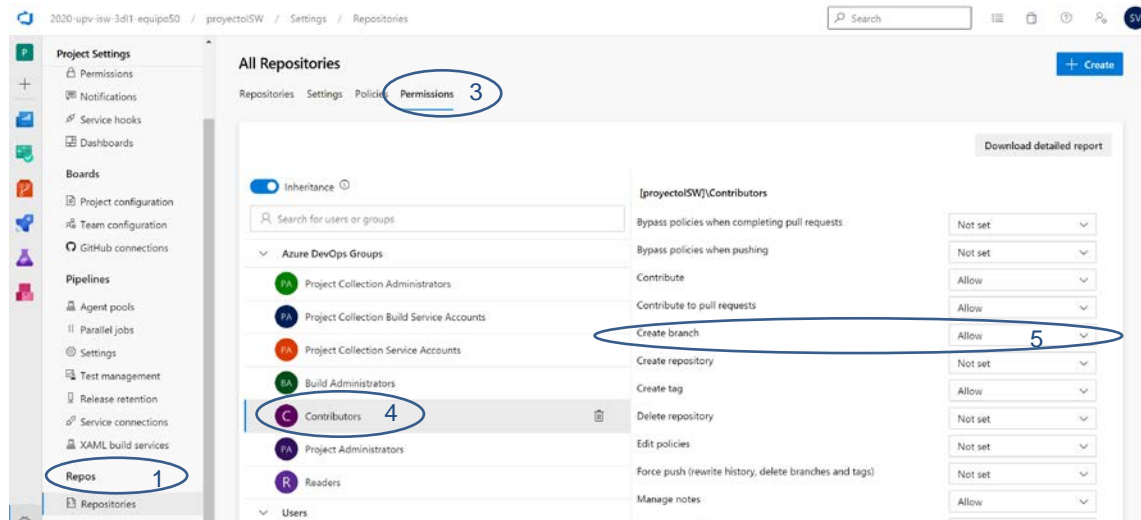


Figura 21. Concesión de permisos de gestión de ramas a un grupo de usuarios

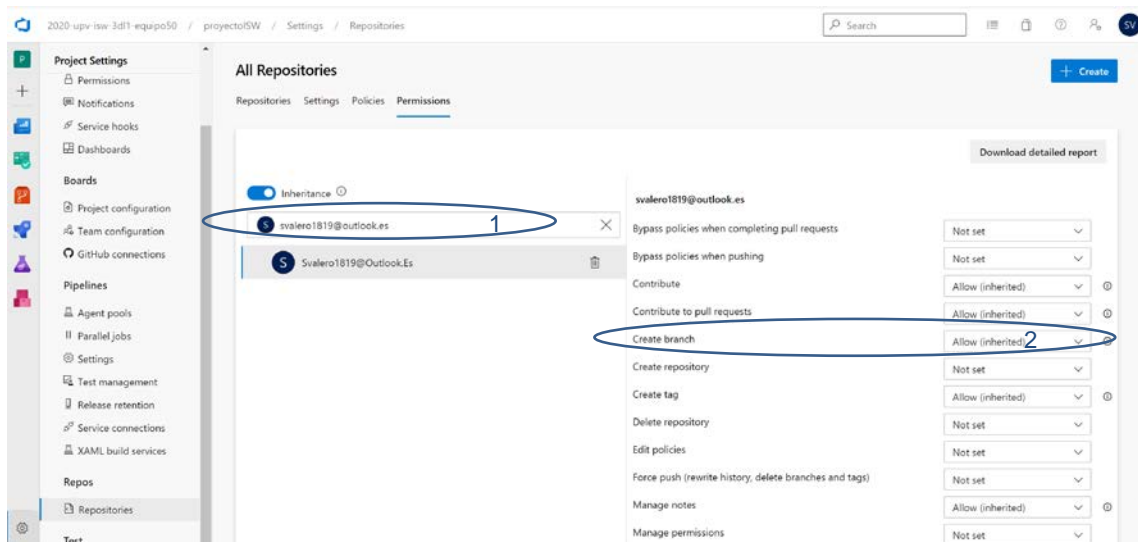


Figura 22. Concesión de permisos en el repositorio a un usuario

- c. Una vez concedidos estos permisos, el usuario en cuestión puede crear ramas. En *Visual Studio* debe seleccionar primero la carpeta del proyecto (dentro de la solución) a partir de la cual se genera la ramificación y pulsar en el menú emergente Nueva rama.

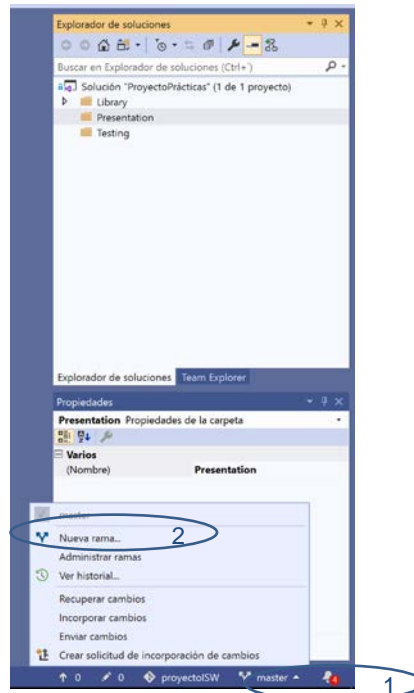


Figura 23. Abriendo el explorador de control de código fuente

- d. En nuestro caso generaremos la nueva rama tomando como versión de código la que tenemos en nuestra área de trabajo local (ver Figura 24)

**NOTA:** En esta práctica cada miembro del equipo puede crear una rama diferente con nombres "...-Feature2", "...-Feature3", etc.

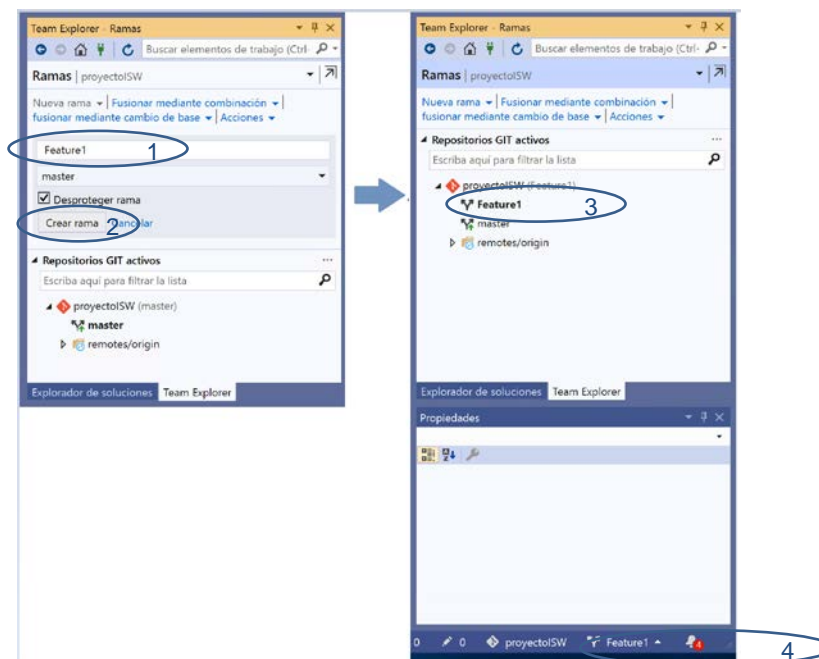


Figura 24. Creación de rama a partir de la versión de código en área de trabajo local

Para ello, debemos introducir el nombre de la nueva rama (Figura 24-1) y pulsar el botón *Crear Rama* (Figura 24-2). Tras esto se obtiene una nueva ramificación (Figura 24-3), convirtiéndose en la rama actual de trabajo (Figura 24-4). Las dos ramas, la principal (master) y la recién creada están en el repositorio local mientras que la principal lo está en el remoto.

## 6.2 Actualizando una rama

Como ya se ha visto anteriormente, en cualquier momento se puede crear una rama local asociada a una rama principal, de manera que se genere un conjunto de cambios y que se puedan subir las nuevas versiones al control de versiones, generando una nueva versión del producto.

- Modifique de nuevo la clase *Program* para que el algún elemento de la lista sea otro (cada miembro del equipo puede elegir una vez más una cadena diferente para los elementos de la lista)
- En este punto vamos a actualizar los cambios en la nueva rama local (ver Figura 25).

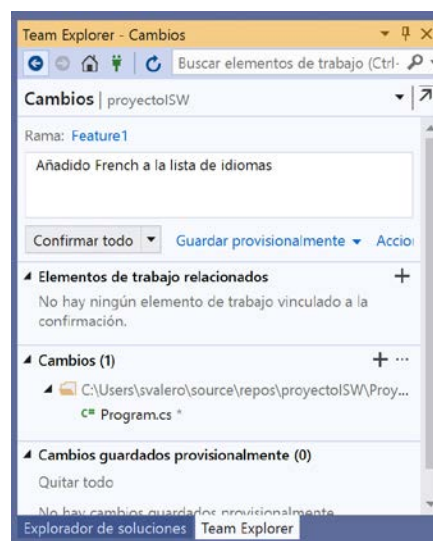


Figura 25. Cambios de la solución a partir de una nueva rama

- Pulse en Confirmar todo de la figura anterior, en la nueva ventana pulse ahora Sincronizar (ver Figura 26). Para que los cambios se suban al servidor pulse en Insertar. Con esto se suben los cambios de la nueva rama al servidor.

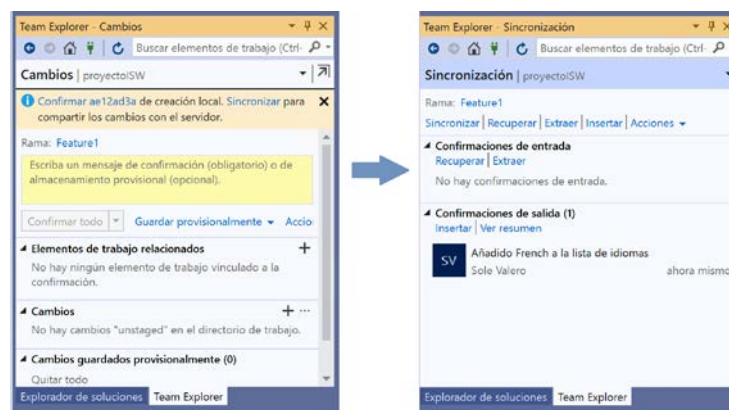


Figura 26 Paso de sincronización de cambios al servidor

- d. Puede observar las dos ramas existentes en el servidor a través de la aplicación Web de *Azure DevOps* y comparar las dos versiones de la clase *Program* en las dos ramas creadas.

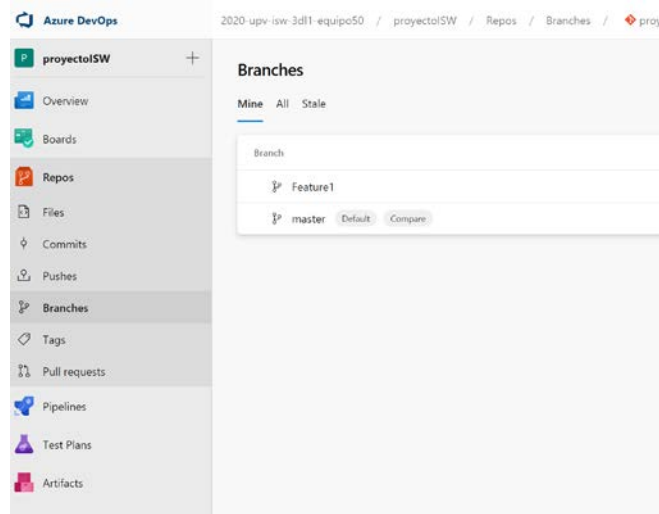


Figura 27 Ramas en el servidor: la principal y la creada en local

- e. Se puede observar las ramas existentes en este momento seleccionando en Visual Studio -> Ramas. Seleccione del repositorio local la rama recién creada y con el menú contextual utilice Ver historial

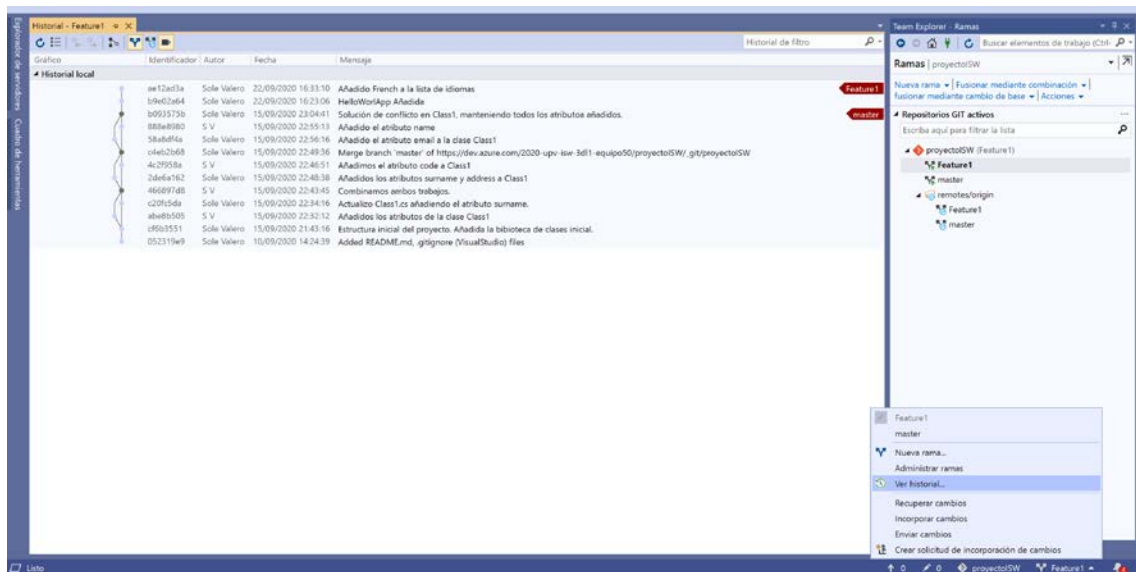


Figura 28. Explorando las ramas

**Nota:** La captura del historial muestra algunos cambios realizados en el proyecto y no descritos en este documento.

### 6.3 Combinación de ramas (merging)

Es posible combinar dos ramas de manera que los cambios existentes en las mismas se combinen. Esto es útil por ejemplo para incorporar los cambios de una rama de desarrollo a una rama de *release* o para combinar dos ramas de desarrollo en una. En este caso vamos a combinar (fusionar) la rama principal (master) con la rama Feature1. Para ello:

- Desde la opción *Ramas* del menú principal del Team Explorer de *Visual Studio*, seleccione la rama de origen a combinar y pulse el botón derecho de ratón, a continuación, seleccione la opción de menú *Fusionar mediante combinación de...* (ver Figura 29).

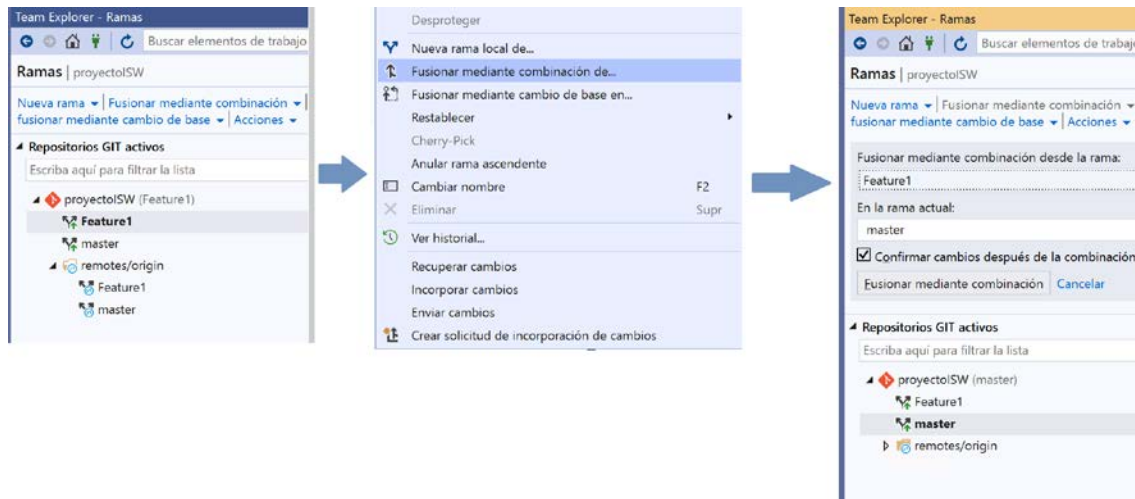


Figura 29. Combinación de dos ramas

- Pulse en el botón *Fusionar mediante combinación*. Al realizar esta operación pueden aparecer conflictos. El conflicto se podrá resolver automáticamente o mediante la herramienta de resolución de conflictos como vimos anteriormente.
- Recuerde que debe confirmar los cambios y sincronizar los nuevos *commits* que se hayan creado para que la fusión de ambas ramas quede reflejada también en el repositorio remoto.
- Se puede comprobar mediante la aplicación Web que en el repositorio central ambas ramas contienen ahora la misma versión de la clase *Program*.

Para más información sobre el modelo de control de versiones en forma de ramas puede consultar la información disponible en:

<https://www.visualstudio.com/nl-nl/docs/tfvc/use-branches-isolate-risk-team-foundation-version-control>

<https://docs.microsoft.com/en-us/azure/devops/repos/git/git-branching-guidance?view=azure-devops>

**NOTA:** El equipo de desarrollo tendrá que decidir cuántas ramas tener, solo una principal, una de desarrollo y otra principal, etc. Se recomienda leer el siguiente documento <https://www.visualstudio.com/nl-nl/docs/tfvc/branch-strategically> para decidir una adecuada estrategia de ramificación.