

# LABORATORIO 2: ADMINISTRACIÓN DE VERSIONES Y SOLUCIÓN DE CONFLICTOS

---

## Contenido

1	Objetivo.....	2
2	Fecha de Entrega.....	2
3	Preparación.....	2
3.1	Instalar extensiones de VS Code.....	2
4	Trabajo Propuesto.....	2
4.1	bifurcar el proyecto en su organización.....	2
4.2	Clonar el proyecto en el computador.....	3
4.3	Distribuir el trabajo del equipo.....	3
4.4	Crear ramas para cada estudiante.....	4
4.5	Copiar las ramas de cada estudiante.....	8
4.6	A (Estudiante-1) Modificar código en la rama Est-1.....	9
4.6	B (Estudiante-2) Modificar código en la rama Est-2.....	11
4.6	C (Estudiante-3) Modificar código en la rama Est-3.....	13
4.7	Modificar el menú de la vista en <i>main</i> .....	15
4.8	Integrar ramas del proyecto.....	15
4.9	A (Estudiante-1) integrar rama Est-3 con main.....	18
4.9	B (Estudiante-2) integrar rama Est-1 con main.....	20
4.9	C (Estudiante-3) integrar rama Est-2 con main.....	22
4.10	Finalizar actualizaciones en la rama principal.....	24
5	Entrega.....	25
5.1	Confirmar cambios finales.....	25
5.2	Compartir los resultados con los evaluadores.....	25

# 1 Objetivo

Aprender a administrar versiones de código y solucionar conflictos en el ambiente de desarrollo colaborativo. Al finalizar este laboratorio el estudiante estará en capacidad de:

1. Entender los conceptos generales para administrar versiones de código en el ambiente de trabajo distribuido (Repositorio GIT, IDE VS Code, Python 3.7 o superior).
2. Administrar las versiones y ramas de código en un repositorio GIT (GitHub).
3. Administrar un repositorio GIT por medio de la consola y línea de comandos.
4. Comprender la organización de una aplicación con el esquema Modelo-Vista-Controlador (MVC por sus siglas en inglés).

## 2 Fecha de Entrega

Recuerde que durante la sesión de laboratorio todos los miembros del grupo deben completar las instrucciones de la sección **4.5 Copiar las ramas de cada estudiante**.

La entrega completa del laboratorio será el martes **22 de agosto** antes de la media noche (11:59 p.m.).

## 3 Preparación

### 3.1 Instalar extensiones de VS Code

Considere las siguientes extensiones de VS Code son fundamentales para el desarrollo del laboratorio.

- a. Extensión para integrar **GitHub** y **VS Code** por *GitHub* en [GitHub Pull Requests and Issues](#)
- b. Extensión para visualizar el historial de **GitHub** en VS Code por *mhutchie* en [Git Graph](#)
- c. Marcador de tareas pendientes dentro del código por *Gruntfuggly* en [TODO Tree](#)

Si no las ha instalado, recuerde los pasos de la practica anterior para adicionarlos a su IDE.

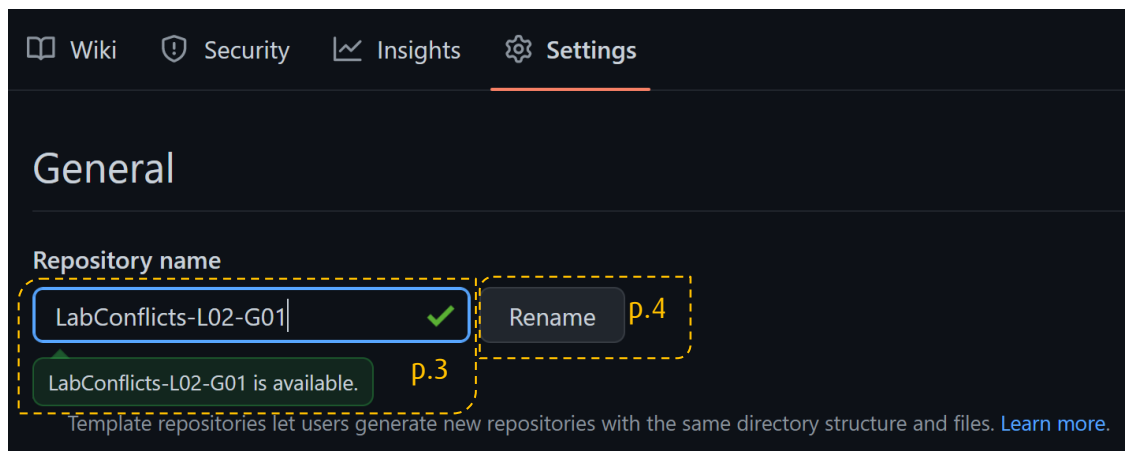
## 4 Trabajo Propuesto

### 4.1 Bifurcar el proyecto en su organización

Bifurque/ejecute el **Fork** del repositorio en su organización utilizando el procedimiento aprendido en la práctica anterior. Recuerde que el repositorio para el segundo laboratorio es [SampleConflicts](#)

Los siguientes pasos los debe completar solo un miembro de la organización:

- p.1. Abra el enlace [SampleConflicts](#)
- p.2. Bifurque el repositorio privadamente como lo aprendió en la práctica pasada.
- p.3. Cambie el nombre del repositorio como lo muestra la Ilustración 1 siguiendo el esquema **LabConflicts-L<<XX>>-G<<YY>>** donde **XX** es el numero de la semana de la práctica que se está haciendo y **YY** el número del grupo de trabajo (ej.: LabConflicts-L02-G01 para el segundo laboratorio del primer grupo).
- p.4. Confirme los cambios seleccionando la opción de **Renombrar (Rename)** de la Ilustración 1.



*Ilustración 1. Ejemplo de cómo renombrar un repositorio existente en GitHub.*

## 4.2 Clonar el proyecto en el computador

Ahora, cada integrante del grupo debe **clonar (clone)** el proyecto desde la organización siguiendo el procedimiento aprendido en la práctica anterior. Recuerda seguir las instrucciones del video dispuesto por el equipo del curso:

- a. Video Uniandes de como clonar un proyecto de GitHub desde VS Code titulado [Como clonar un repositorio GIT](#)

Ahora, repita los pasos utilizando el enlace del repositorio privado de la organización creado a partir del **ISIS1225-SampleMVC**.

- p.5. Ingrese al portal **GitHub** de su organización.
- p.6. Diríjase al repositorio del primer laboratorio.
- p.7. Copie el URL GIT del repositorio.
- p.8. Diríjase a VS Code.
- p.9. Inicie el comando **Clonar (Clone)** desde la paleta de comandos.
- p.10. Pegue el URL en el campo despegado
- p.11. Seleccione un lugar apropiado en su directorio local para guardar los archivos del proyecto.
- p.12. Espere a que descarguen los archivos y comience a trabajar<sup>1</sup>.

**IMPORTANTE:** aunque los repositorios y sus archivos dentro de una organización son compartidos, solo un miembro del grupo necesita grupo **bifurcar (fork)** y **clonar (clone)** el repositorio.

## 4.3 Distribuir el trabajo del equipo

En este laboratorio practicamos integrar diferentes versiones de código dentro de un grupo de trabajo. Por lo tanto, las modificaciones del código se distribuyen entre los miembros del grupo.

Para ello cada uno de los integrantes debe tener un papel asignado siguiendo estas instrucciones:

- p.13. Decidir entre todos quien de los miembros tomara el papel de del **Estudiante 1**, el **Estudiante 2** y el **Estudiante 3**.

<sup>1</sup> Este proceso puede completarse por diferentes medios, utilizando la terminal/console de Windows o MacOS, y también utilizando aplicaciones independientes como [GitHub Desktop](#) y [Sourcetree](#)

- p.14. Modifique el archivo **README.md** del repositorio 2 de acuerdo con su decisión como lo indica Ilustración 2.
- p.15. Ejecute un **Commit** en el repositorio con el comentario “Cambios README laboratorio 2”(git commit -m “Cambios README laboratorio 2”).
- p.16. Ejecute un **Push** de los cambios del repositorio hacia GitHub.

```
30 <!-- GROUP MEMBERS -->
31 ## Members
32 |
33 The students edit this section to add their names, Uniandes emails, and specify
   which project functionality of the project they will implement.
34 -----
35 1. Student-1, Melissa Carolina Castañeda Prieto, <mc.castanedap@uniandes.edu.co>
36 1. Student-2, Daniel Alejandro Angel Fuertes, <da.angelf@uniandes.edu.co>.
37 1. Student-3, Juan Andres Ariza Gacharna, <ja.arizag@uniandes.edu.co>.
38
39 [Back to top](#sample-conflicts)
40 -----
41 <!-- ABOUT THE PROJECT -->
42 ## About The Project
```

Ilustración 2. README con la distribución de trabajo para el segundo laboratorio.

## 4.4 Crear ramas para cada estudiante

Las *Ramas* o **Branch** se utilizan en los repositorios para mantener versiones alternativas del mismo proyecto que eventualmente se integrarán en una rama principal denominada **main**.

Los miembros del equipo utilizan estas ramas/**Branch** para trabajar al mismo tiempo en diferentes partes del código, esta práctica evita conflictos de versiones y previene introducir errores en el código que ya esté funcionando. Por ejemplo, se puede utilizar una rama para desarrollar una nueva función o para corregir un error descubierto.

Las ramas/**Branch** están aisladas entre sí y esto nos permite utilizarlas para experimentar de forma segura modificaciones dentro del código. Para aprender a utilizar estas ramas hemos dispuesto el siguiente video del curso:

- Video Uniandes de cómo crear ramas desde VS Code titulado [Crear ramas en GitHub desde VSCode](#)

En este laboratorio cada uno de los estudiantes creará una rama de la cual estará a cargo. Luego, modificará el código. Y finalmente integrará sus cambios con las ramas de sus compañeros.

Siguiendo los pasos descritos a continuación cada estudiante creará una rama en el proyecto denominadas *Est-1*, *Est-2* y *Est-3* respectivamente:

- p.17. Seleccione la extensión para control de versiones en VS Code (**Ctrl+Shift+G**) como se ve en Ilustración 3.
- p.18. Despliegue el menú de comandos presionando el icono de opciones según lo indica Ilustración 3 (comandos en VS Code: **Ctrl+Shift+P**).
- p.19. Seleccione la opción **Checkout to...** como lo muestra la Ilustración 3 (comandos en VS Code: **git: checkout to**).

- p.20. Seleccione crear una nueva rama (comandos VS Code **Create new Branch...**) como lo indica Ilustración 4.
- p.21. Escriba el nombre de la rama que le corresponde en el campo desplegado (ej.: **Est-1** para el **Estudiante-1**).<sup>2</sup>
- p.22. Publique la rama creada localmente con el menú de VS Code (Branch -> Publish Branch) como lo indica Ilustración 5.<sup>3</sup>

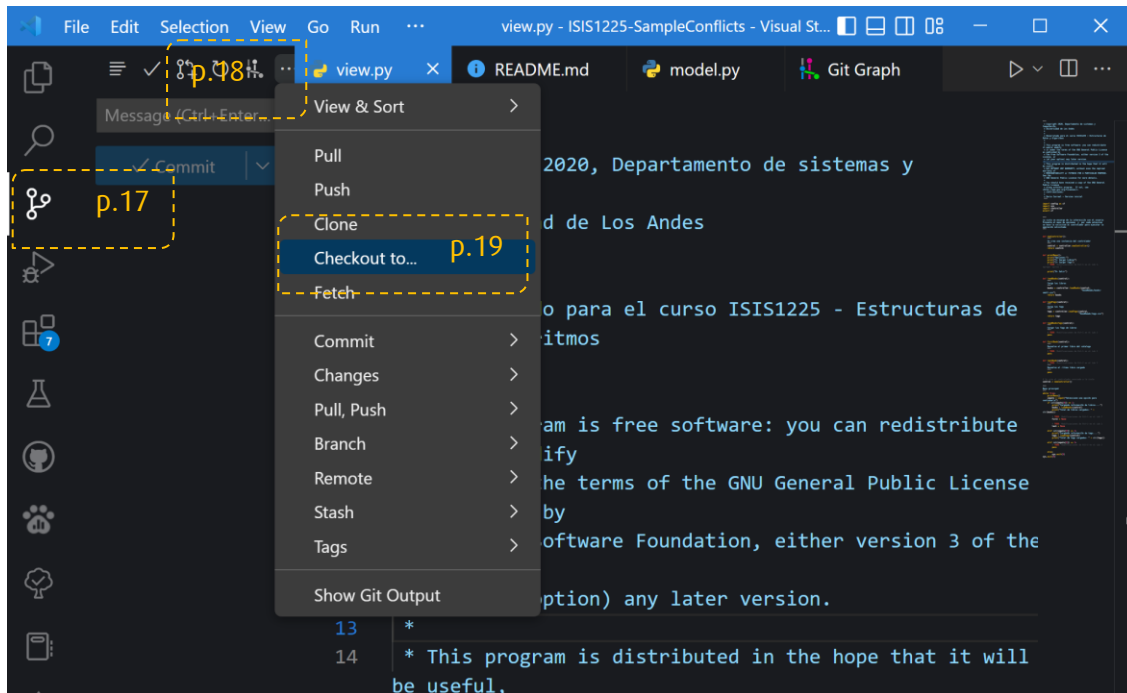


Ilustración 3. Secuencia de comandos en VS Code para crear una nueva rama con GIT.

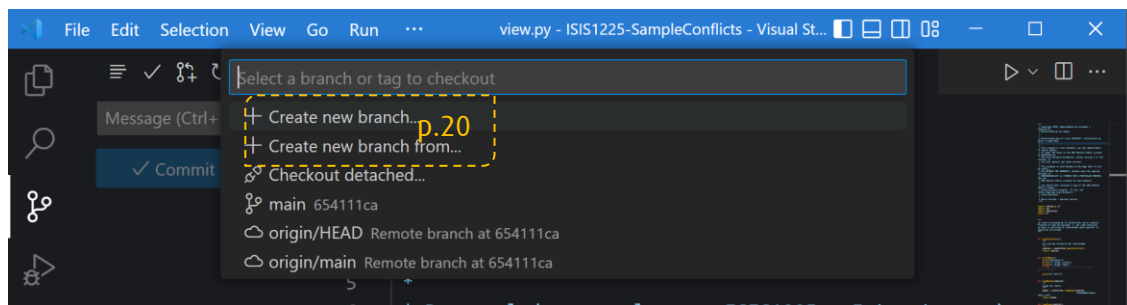


Ilustración 4. Menú de comandos VS Code para crear una nueva rama GIT.

<sup>2</sup> Puede lograr el mismo resultado utilizando el comando GIT Bash por consola **"git checkout -b <<nombre de la rama>>"**.

<sup>3</sup> Puede lograr el mismo resultado utilizando el comando GIT Bash por consola **"git push --set-upstream origin <nombre de la rama>"**.

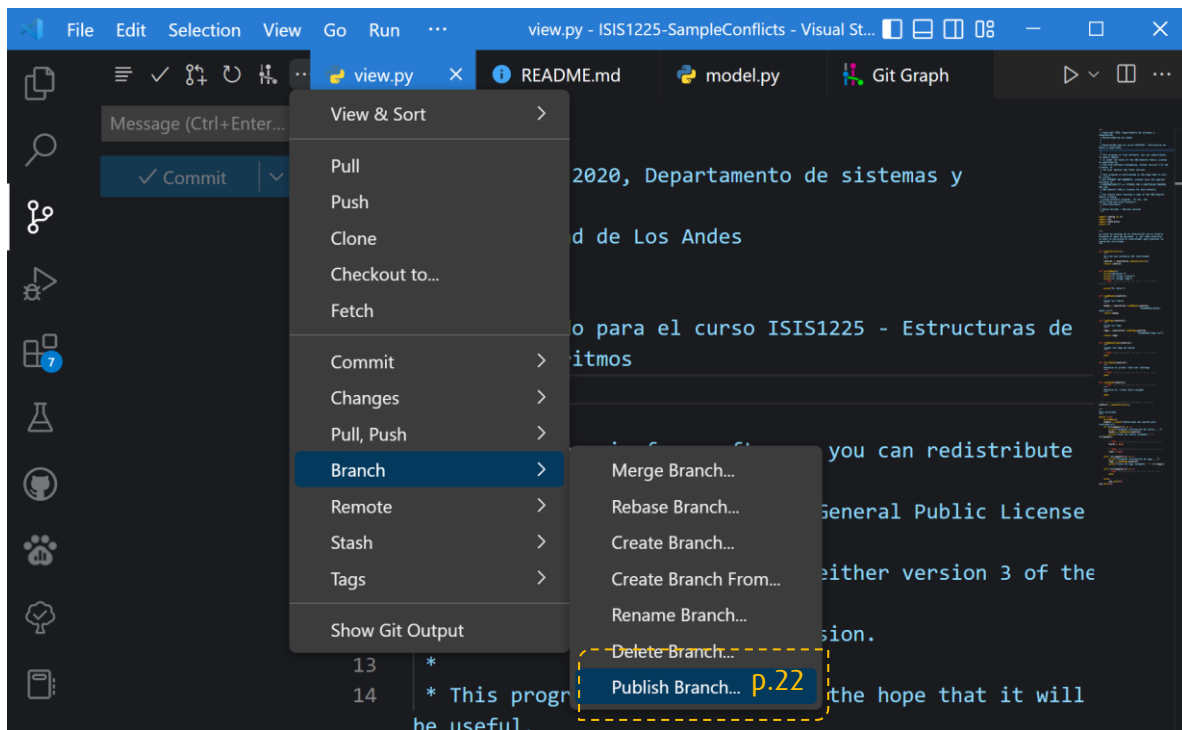


Ilustración 5. Menú de comandos VS Code para publicar una rama GIT.

Al completar el procedimiento confirme que las ramas **Est-1**, **Est-2** y **Est-3** fueron creadas exitosamente. Esto se puede lograr con el comando *git branch -a* ejecutable por la consola de comando o la interfaz de VS Code con el comando *Git: Checkout to...*

Por ejemplo, el caso particular del **Estudiante 1**, la consola de la Ilustración 6 muestra primero las ramas locales, la marca verde resalta la rama de trabajo seleccionada. Después la lista de ramas en rojo identifica las ramas existentes en el servidor remoto (GitHub) y la marca **origin/main** señala la rama principal que trabaja el servidor remoto.

Esto significa que por ejemplo el **Estudiante 1** no podrá ver todas las ramas remotas a menos que el **Estudiante 2** y **Estudiante 3** las publiquen adecuadamente. Este efecto se observa en la Ilustración 7 y la Ilustración 8 donde el **Estudiante 1** y el **Estudiante 2** tienen perspectivas incompletas del proyecto desarrollado. Esto dependerá del código que hallan sincronizado con su repositorio remoto y su máquina local.

**IMPORTANTE:** en caso de que el grupo este conformado por dos estudiantes uno de los dos integrantes (**Estudiante 1** o **Estudiante 2**) deberá tomar el papel del **Estudiante 3**.

```
C:\Users\Felipe\OneDrive\Documents\GitHub\phillipus85\ISIS1225-SampleConflicts>git branch -a
Est-1
Est-2
* Est-3
main
remotes/origin/HEAD -> origin/main
remotes/origin/main

C:\Users\Felipe\OneDrive\Documents\GitHub\phillipus85\ISIS1225-SampleConflicts>git branch -a
Est-1
Est-2
* Est-3
main
remotes/origin/Est-1
remotes/origin/HEAD -> origin/main
remotes/origin/main

C:\Users\Felipe\OneDrive\Documents\GitHub\phillipus85\ISIS1225-SampleConflicts>git branch -a
Est-1
Est-2
* Est-3
main
remotes/origin/Est-1
remotes/origin/Est-2
remotes/origin/HEAD -> origin/main
remotes/origin/main

C:\Users\Felipe\OneDrive\Documents\GitHub\phillipus85\ISIS1225-SampleConflicts>git branch -a
Est-1
Est-2
* Est-3
main
remotes/origin/Est-1
remotes/origin/Est-2
remotes/origin/Est-3
remotes/origin/HEAD -> origin/main
remotes/origin/main

C:\Users\Felipe\OneDrive\Documents\GitHub\phillipus85\ISIS1225-SampleConflicts>
```

Ilustración 6. Ejemplo de consola revisando el estado de las ramas locales y remotas.

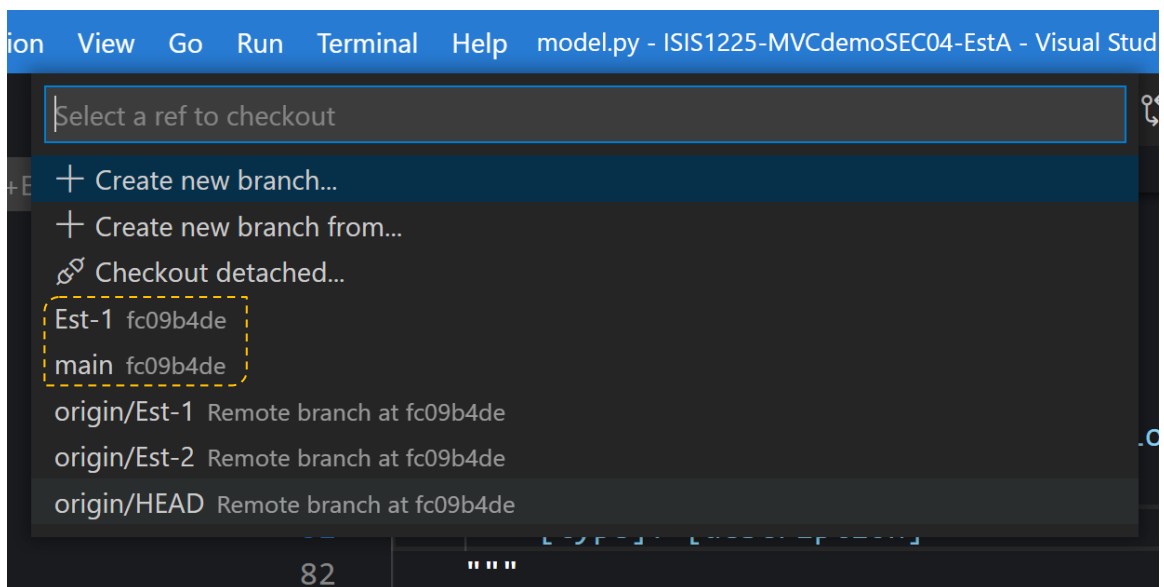


Ilustración 7. Perspectiva del código sincronizado entre el servidor de GitHub y el Estudiante 1.

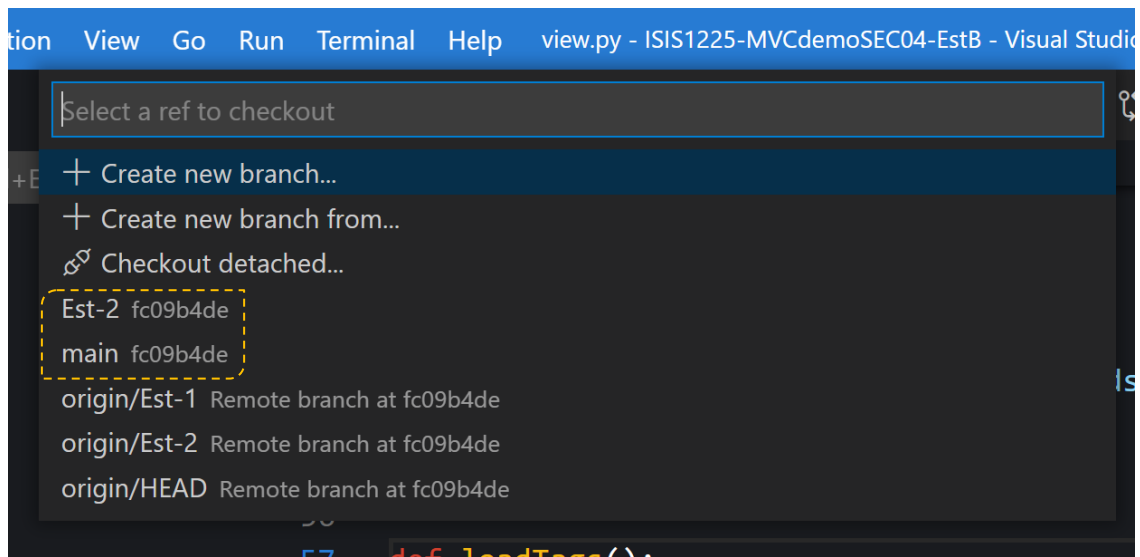


Ilustración 8. Perspectiva del código sincronizado entre el servidor de GitHub y el Estudiante 2.

## 4.5 Copiar las ramas de cada estudiante

En este paso los estudiantes copian localmente las ramas de sus compañeros para preparar la división de tareas y permitir que cada uno de ellos implemente los cambios recomendados en la guía.

El objetivo es que cada uno de los integrantes trabaje sobre sus propios cambios (ej.: el **Estudiante 3** implementa las modificaciones de la rama **Est-3**) y genere una nueva versión en su rama. Esto creará tres (3) ramas con tres (3) versiones diferentes de código para cada uno de los estudiantes.

Al final cada uno de los estudiantes deberán integrar (**Merge**) los cambios de sus respectivas ramas en la principal (**main**). Recuerde que las secciones **4.6 A (Estudiante-1) Modificar código en la rama Est-1**, **4.6 B (Estudiante-2) Modificar código en la rama Est-2** y **4.6 C (Estudiante-3) Modificar código en la rama Est-3** describen los pasos que deben completar cada uno de los estudiantes.

Es **IMPORTANTE** recordar que al completar de los pasos de las secciones **4.6 A**, **4.6 B** y **4.6 C** crearán **tres versiones diferentes y conflictivas de código**. esto es un acto deliberado para que los estudiantes resuelvan el problema de integración **MANUALMENTE**.

Si lo necesita, puede recordar cómo debe confirmar los cambios localmente (**Commit**) y empujarlos (**Push**) al servidor GitHub por medio del siguiente video:

- a. Video Uniandes titulado [Como administrar desarrollo de código desde VS Code](#)

Para traer estas ramas remotas a la maquina local cada uno de los estudiantes deberá completar los siguientes pasos:

- p.23. Seleccionar el menú de control de versiones (Source Control).
- p.24. Seguir el menú a la opción Pull, Push.
- p.25. Seleccionar la opción de Fetch From all Remotes.
- p.26. Abrir la paleta de comandos en VS Code (**Ctrl+Shift+P**).
- p.27. Escribir **git: checkout** en el espacio disponible como lo muestra Ilustración 9.



- p.28. Seleccionar la rama remota que desea copiar como se ve en la Ilustración 10. Recuerde que tiene el mismo nombre que la rama local con el prefijo **origin\<<nombre de la rama>>**.<sup>4</sup>
- p.29. Confirme que copió la rama remota en su máquina local escribiendo **git: checkout** como lo ilustra la Ilustración 10. Recuerde que la rama local **NO** tiene ningún prefijo.<sup>5</sup>

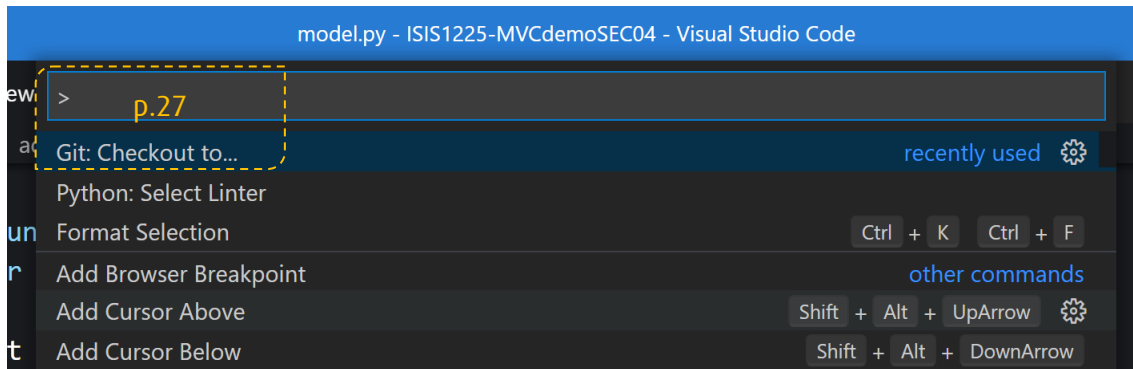


Ilustración 9. Paleta de comandos en VS Code con comandos GIT para seleccionar una rama por medio de Checkout.

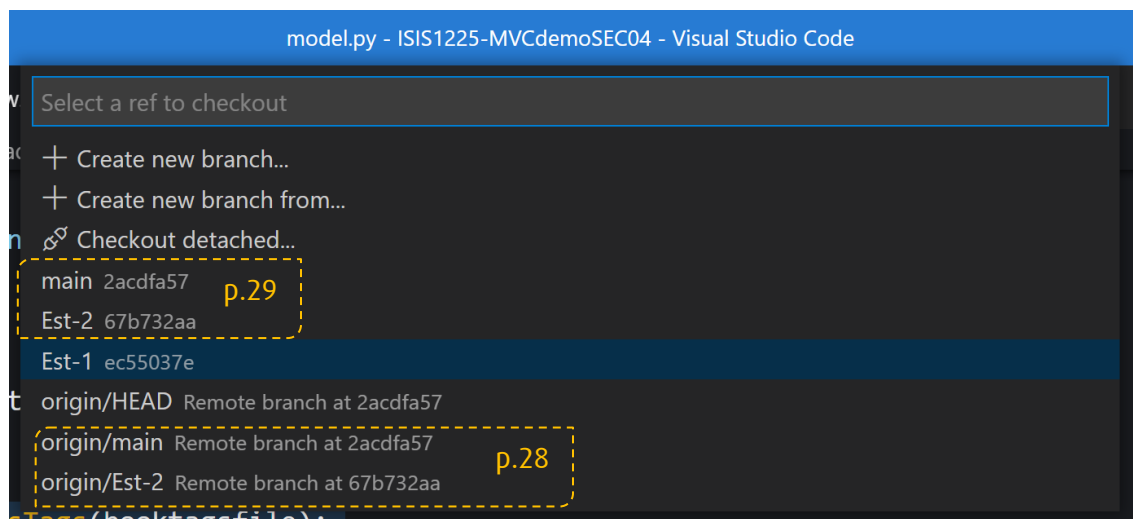


Ilustración 10. Paleta de comandos en VS Code con el estado de las ramas locales y remotas del repositorio.

## 4.6 A (Estudiante-1) Modificar código en la rama Est-1

Ahora como **Estudiante-1** busque los comentarios “# **TODO: Mods de Est-1...**” en el proyecto para implementar los cambios indicados en la rama **Est-1**.

Para ello diríjase al **view.py** y complete las siguientes indicaciones.

- p.a.1. Modifique la definición de la función **printMenu()** como lo indica Segmento 1.
- p.a.2. Modifique la definición de la función **LoadBooksTags()** como lo indica Segmento 2.
- p.a.3. Modifique la definición de la función **firstBook()** como lo indica Segmento 3.

<sup>4</sup> Puede lograr el mismo resultado utilizando el comando GIT Bash por consola “**git checkout <<nombre de la rama>>**”.

<sup>5</sup> Puede lograr el mismo resultado utilizando el comando GIT Bash por consola “**git switch -c origin/<<nombre de la rama>>**”.

p.a.4. Modifique la invocación de la función **firstBook()** como lo indica Segmento 4.

Ahora, busque los comentarios “# **TODO: Mods de Est-1, Est-2 y Est-3 en el Lab 2**” dentro del **controller.py** y complete las siguientes modificaciones:

p.a.5. Modifique la definición de la función **LoadBooksTags()** como lo especifica Segmento 5.

```
def printMenu():
    print("Opciones:")
    print("1- Cargar Libros")
    print("2- Cargar Tags")
    # TODO: Mods de Est-1, Est-2 y Est-3 en Lab 2
    print("3- Cargar Tags de Libros")
    print("0- Salir")
```

*Segmento 1. modificaciones Est-1, view.py, definición de la función printMenu().*

```
def loadBooksTags(control):
    """
    Cargar los Tags de libros
    """
    # TODO: Mods de Est-1 en el Lab 2
    booktags = controller.loadBooksTags(control,
                                         "GoodReads/book_tags-small.csv")
    return booktags
```

*Segmento 2. modificaciones Est-1, view.py, definición de la función loadBooksTags().*

```
def firstBook(control):
    """
    Devuelve el primer libro del catalogo
    """
    # TODO: Mods de Est-2 en el Lab 2
    first = controller.firstBook(control)
    return first
```

*Segmento 3. modificaciones Est-1, view.py, definición de la función firstBook().*

```
# TODO: Mods de Est-1 en el Lab 2
first = firstBook(control)
print("Primer libro cargado:\n" + str(first) + "\n")
```

*Segmento 4. modificaciones Est-1, view.py, invocación de la función loadBooksTags().*

```
def loadBooksTags(control, filename):
    """
    Carga la información que asocia tags con libros.
    """
    # TODO: Mods de Est-1, Est-2 y Est-3 en el Lab 2
    catalog = control["model"]
    booksfile = os.path.join(cf.data_dir, filename)
    catalog = model.addBookTags(catalog, booksfile)
    return model.bookTagSize(catalog)
```

*Segmento 5. modificaciones Est-1, controller.py, definición de la función loadBooksTags().*

Por último, en el **model.py** busque los comentarios “# **TODO: Mods de Est-1, Est-2 y Est-3 en el Lab 2**” y modifique el código como se indica a continuación:

p.a.6. Modifique la definición de la función **addBooksTags()** como lo indica Segmento 6.

p.a.7. Modifique la definición de la función **addBookTag()** como lo indica Segmento 7.

- p.a.8. Modifique la definición de la función **createBookTagList()** como lo indica Segmento 8.
- p.a.9. Adjunte los cambios en su máquina local con el comando **Stage**.
- p.a.10. Confirme los cambios con el comando **Commit** y el mensaje **“cambios del Estudiante 1”**.
- p.a.11. Envíe los cambios al repositorio remoto en GitHub con **Push**.

```
def addBookTags(catalog, booktagsfile):
    """
    Esta función guardar los booktags provenientes del archivo CSV.
    """
    # TODO: Mods de Est-1, Est-2 y Est-3 en el Lab 2
    catalog["book_tags"] = lt.newList(datastructure="ARRAY_LIST",
                                      filename=booktagsfile)
    return catalog
```

*Segmento 6. modificaciones Est-1, model.py, definición de la función addBooksTags().*

```
def addBookTag(catalog, booktag):
    """
    Esta función agrega un elemento a lista de booktags.
    """
    # TODO: Mods de Est-1, Est-2 y Est-3 en el Lab 2
    lt.addLast(catalog["book_tags"], booktag)
    return catalog
```

*Segmento 7. modificaciones Est-1, model.py, definición de la función addBookTag().*

```
def createBookTagList(catalog):
    """
    Esta función crea una lista vacía para booktags.
    """
    # TODO: Mods de Est-1, Est-2 y Est-3 en el Lab 2
    catalog["book_tags"] = lt.newList(datastructure="ARRAY_LIST")
    return catalog
```

*Segmento 8. modificaciones Est-1, model.py, definición de la función createBookTagList().*

## 4.6 B (Estudiante-2) Modificar código en la rama Est-2

Ahora como **Estudiante-2** busque los comentarios **“# TODO: Mods de Est-2...”** en el proyecto para implementar los cambios indicados en la rama **Est-2**.

Para ello diríjase al **view.py** y complete las siguientes indicaciones.

- p.b.1. Modifique la definición de la función **printMenu()** como lo indica Segmento 9.
- p.b.2. Modifique la definición de la función **LastBook()** como lo indica Segmento 10.
- p.b.3. Modifique la invocación de la función **LastBook()** como lo indica Segmento 11.

```
def printMenu():
    print("Opciones:")
    print("1- Cargar Libros")
    print("2- Cargar Tags")
    # TODO: Mods de Est-1, Est-2 y Est-3 en Lab 2
    print("3- Cargar Booktags")
    print("0- Salir")
```

*Segmento 9. modificaciones Est-2, view.py, definición de la función printMenu().*

```
def lastBook(control):
    # TODO: Mods de Est-3 en el Lab 2
    """
    Devuelve el último libro cargado
    """
    last = controller.lastBook(control)
    return last
```

*Segmento 10. modificaciones Est-2, view.py, definición de la función lastBook().*

```
# TODO: Mods de Est-2 en el Lab 2
last = lastBook(control)
print("Último libro cargado:\n" + str(last) + "\n")
```

*Segmento 11. modificaciones Est-2, view.py, invocación de la función lastBook().*

Ahora, busque los comentarios “**# TODO: Mods de Est-1, Est-2 y Est-3 en el Lab 2**” dentro del **controller.py** y complete las siguientes modificaciones:

p.b.1. Modifique la definición de la función **LoadBooksTags()** como lo especifica Segmento 12.

```
def loadBooksTags(control, filename):
    """
    Carga la información que asocia tags con libros.
    """
    # TODO: Mods de Est-1 y Est-2, Est-3 en el Lab 2
    tf = os.path.join(cf.data_dir, filename)
    input_file = csv.DictReader(open(tf, encoding="utf-8"))
    control["model"] = model.createBookTagList(control["model"])
    for booktag in input_file:
        model.addBookTag(control["model"], booktag)
    return model.bookTagSize(control["model"])
```

*Segmento 12. modificaciones Est-2, controller.py, definición de la función loadBooksTags().*

Por último, en el **model.py** busque los comentarios “**# TODO: Mods de Est-1, Est-2 y Est-3 en el Lab 2**” y modifique el código como se indica a continuación:

p.b.2. Modifique la definición de la función **addbookstags()** como lo indica el Segmento 13.

p.b.3. Adjunte los cambios en su máquina local con el comando **Stage**.

p.b.4. Confirme los cambios con el comando **Commit** y el mensaje “**cambios del Estudiante 2**”.

p.b.5. Envíe los cambios al repositorio remoto en GitHub con **Push**.

```
def addBookTags(catalog, booktagsfile):
    """
    Esta función guardar los booktags provenientes del archivo CSV.
    """
    # TODO: Mods de Est-1 y Est-2, Est-3 en el Lab 2
    bt = lt.newList(datastructure="SINGLE_LINKED", filename=booktagsfile)
    catalog["book_tags"] = bt
    return catalog
```

*Segmento 13. modificaciones Est-2, view.py, definición de la función loadBooksTags().*

## 4.6 C (Estudiante-3) Modificar código en la rama Est-3

Ahora como **Estudiante-3** busque los comentarios “*# TODO: Mods de Est-3...*” en el proyecto para implementar los cambios indicados en la rama **Est-3**.

Para ello diríjase al **view.py** y complete las siguientes indicaciones.

**p.c.1.** Modifique la definición de la función **printMenu()** como lo indica el Segmento 14.

**p.c.2.** Modifique la opción 3 del menú principal y la invocación de la función **LastBook()** como lo indica el Segmento 15

Ahora, busque los comentarios “*# TODO: Mods de Est-1, Est-2 y Est-3 en el Lab 2*” dentro del **controller.py** y complete las siguientes modificaciones:

**p.c.3.** Modifique la definición de la función **LoadBooksTags()** como lo especifica el Segmento 16.

**p.c.4.** Modifique la definición de las funciones **firstBook()** y **LastBook()** como lo especifica el Segmento 17.

```
def printMenu():
    print("Opciones:")
    print("1- Cargar Libros")
    print("2- Cargar Tags")
    # TODO: Mods de Est-1, Est-2 y Est-3 en Lab 2
    print("3- Cargar Book-Tags!!!...")
    print("0- Salir")
```

*Segmento 14. modificaciones Est-3, view.py, definición de la función printMenu().*

```
elif int(inputs[0]) == 3:
    # TODO: Mods de Est-3 en el Lab 2
    print("Cargando información de Book-Tags...")
    booktags = loadBooksTags(control)
    print("Total de Book-Tags cargados: " + str(booktags))
```

*Segmento 15. modificaciones Est-3, view.py, opción 3 del menú y la ejecución de la función loadBooksTags().*

```
def loadBooksTags(control, filename):
    """
    Cargo los tags de los libros del archivo
    """
    # TODO: Mods de Est-1 y Est-2, Est-3 en el Lab 2
    booktagfile = os.path.join(cf.data_dir, filename)
    catalog = control["model"]
    input_file = csv.DictReader(open(booktagfile, encoding="utf-8"))
    catalog = model.createBookTagList(catalog)
    for booktag in input_file:
        model.addBookTag(catalog, booktag)
    return model.bookTagSize(catalog)
```

*Segmento 16. modificaciones Est-3, controller.py, definición de la función loadBooksTags().*

```
def firstBook(control):
    """
    Devuelve el primer libro del catalogo
    """
    # TODO: Mods Est-3 en el Lab 2
    return model.firstBook(control["model"])
```

```
def lastBook(control):
    """
    Devuelve el último libro del catalogo
    """
    # TODO: Mods Est-3 en el Lab 2
    return model.lastBook(control["model"])
```

*Segmento 17. modificaciones Est-3, controller.py, definición de la función firstBook() y lastBook().*

Por último, en el **model.py** busque los comentarios “# **TODO: Mods de Est-1, Est-2 y Est-3 en el Lab 2**” y modifique el código como se indica a continuación:

p.c.5. Modifique la definición de la función **addBookTag()** como lo especifica el Segmento 18.

p.c.6. Modifique la definición de las funciones **bookSize()**, **tagSize()**, **bookTagSize()**, **firstBook()** y **lastBook()** como lo especifica el Segmento 19.

p.c.7. Adjunte los cambios en su máquina local con el comando **Stage**.

p.c.8. Confirme los cambios con el comando **Commit** y el mensaje “**cambios del Estudiante 3**”.

p.c.9. Envíe los cambios al repositorio remoto en GitHub con **Push**.

```
def addBookTag(catalog, booktag):
    """
    Esta función agrega un elemento a lista de booktags.
    """
    # TODO: Mods de Est-1 y Est-2, Est-3 en el Lab 2
    lt.addLast(catalog["book_tags"], booktag)
    return catalog
```

*Segmento 18. modificaciones Est-3, model.py, definición de la función addBookTag ().*

```
def bookSize(catalog):
    # TODO Mods de Est-3 en el Lab 2
    return lt.size(catalog["books"])

def tagSize(catalog):
    # TODO Mods de Est-3 en el Lab 2
    return lt.size(catalog["tags"])

def bookTagSize(catalog):
    # TODO Mods de Est-3 en el Lab 2
    return lt.size(catalog["book_tags"])

def firstBook(catalog):
    # TODO Mods de Est-3 en el Lab 2
    return lt.firstElement(catalog["books"])

def lastBook(catalog):
    # TODO Mods de Est-3 en el Lab 2
    return lt.lastElement(catalog["books"])
```

*Segmento 19. modificaciones Est-3, model.py, definiciones para las funciones bookSize(), tagSize(), bookTagSize(), firstBook() y lastBook().*

## 4.7 Modificar el menú de la vista en *main*

Aquí uno de los tres estudiantes (no importa quien sea) deberá cambiar el código sobre la rama *main*.

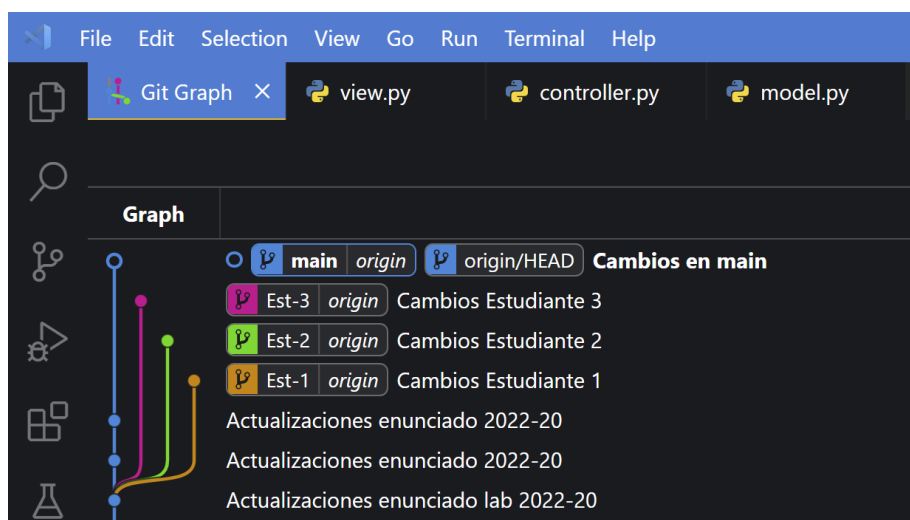
Para ello diríjase al `view.py`, busque el comentario “# *TODO: Mods de Est-1, Est-2 y Est-3 en el Lab 2*” y complete los siguientes pasos:

- p.17. Modifique la definición de la función `printMenu()` como lo indica el Segmento 20.
- p.18. Adjunte los cambios en su máquina local con el comando **Stage**.
- p.19. Confirme los cambios con el comando **Commit** y el mensaje “*cambios en main*”.
- p.20. Envíe los cambios al repositorio remoto en GitHub con **Push**.

```
def printMenu():  
    print("Opciones:")  
    print("1- Cargar Libros")  
    print("2- Cargar Tags")  
    # TODO: Mods de Est-1 en el Lab 2, agregar opción 3  
    print("3- Cargar los Tags de Libros")  
    print("0- Salir")
```

*Segmento 20. modificaciones main, view.py, definición de la función printMenu().*

Al finalizar estos pasos su historial de GitHub debe mostrar la existencia de las ramas *Est-1*, *Est-2*, *Est-3* y la *main* con diferentes estados en el servidor GitHub como se ve en la Ilustración 11. Recomendamos utilizar la extensión [Git Graph](#) para renderizar fácilmente el historial de versiones de código de sus proyectos.



*Ilustración 11. Historial de cambios de las ramas activas creadas por los miembros del grupo.*

## 4.8 Integrar ramas del proyecto

En este momento el grupo tiene cuatro versiones diferentes del código dentro de cada una de las ramas (*main*, *Est-1*, *Est-2* y *Est-3*). Ahora deben integrar el código utilizando el comando **Merge** el cual generará una versión definitiva del código e intentará solucionar los conflictos que aparezcan entre las ramas.

Este proceso de Merge o integración de las versiones se hace secuencialmente, así que las ramas deben integrarse una después de la otra. En este caso el *Estudiante 1* integra primero la rama *Est-3*

con **main**, luego el **Estudiante 2** integrará la rama **Est-1** con la actualización de **main**. Y, por último, el **Estudiante 3** integrará la rama **Est-2** con la versión resultante entre **main** y **Est-1**.

Este procedimiento contraintuitivo es un escenario apropiado para usar los comandos GIT de **Pull**, **Push**, **Checkout**, **Merge**, entre otros. Estas instrucciones serán de gran utilidad para los grupos de trabajo en donde se deben integrar diferentes funcionalidades y estados del código a una versión final.

Entender el proceso para manejar las versiones, las ramas y los conflictos de código ayuda a mejorar la calidad del código implementado.

**IMPORTANTE** recuerde que cambiar el orden de estos pasos o desviarse de alguna manera del procedimiento generará resultados diferentes a los ilustrados en el documento.

Antes de iniciar el proceso estudie las instrucciones del video dispuesto por el equipo del curso:

- a. Video Uniandes de como integrar versiones de código desde VS Code titulado [Como hacer Merge en VSCode](#).

Antes de iniciar el **Merge** cada uno de los estudiantes deberá actualizar sus ramas locales desde el repositorio GitHub. Para ello complete los siguientes pasos:

- p.17. Abrir la paleta de comandos en VS Code (Ctrl+Shift+P).
- p.18. Seleccionar el menú de control de versiones (Source Control) como se ve en Ilustración 12.
- p.19. Seguir el menú a la opción **PULL**, **PUSH** como se ve en Ilustración 12.
- p.20. Seleccionar la opción de **Fetch From all Remotes** como lo muestra Ilustración 12.<sup>6</sup>
- p.21. Escribir **git: checkout** en el espacio disponible como lo
- p.22. Envíe los cambios al repositorio remoto en GitHub con **Push**.

Este procedimiento permitirá a cada uno de los estudiantes ver las referencias a todas las ramas remotas. Para obtener una copia local de las ramas de GitHub los estudiantes deben ejecutar los siguientes pasos por consola GIT:

- p.23. El **Estudiante 1** debe ejecutar los siguientes comandos:
  1. *Git checkout Est-2*
  2. *Git checkout Est-3*
  3. *Git Checkout main*
- p.24. El **Estudiante 2** debe ejecutar los siguientes comandos:
  1. *Git checkout Est-1*
  2. *Git checkout Est-3*
  3. *Git Checkout main*
- p.25. El **Estudiante 3** debe ejecutar los siguientes comandos:
  1. *Git checkout Est-1*
  2. *Git checkout Est-2*
  3. *Git Checkout main*

Esto creará una copia local de las ramas remotas. Para comprobar que el procedimiento fue exitoso puede escribir en la paleta de comandos (**Ctrl+Shift+P**) la instrucción **>git: checkout to...** como se ve en Ilustración 13, esto desplegará la lista de ramas locales y remotas disponibles. O en la consola de GIT Bash ejecutar el comando **git Branch -a** como se muestra en la Ilustración 14.

---

<sup>6</sup> Puede lograr el mismo resultado utilizando el comando GIT Bash por consola **"git fetch --all"**.



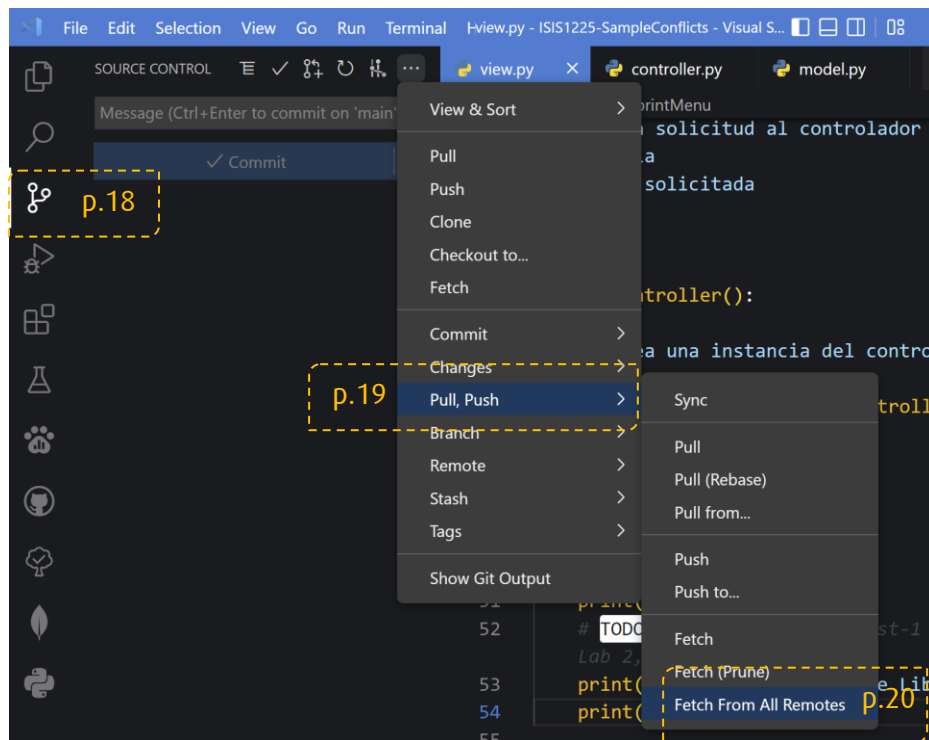


Ilustración 12. Secuencia de operaciones para copiar las ramas activas en el menú de control de versiones de VS Code.

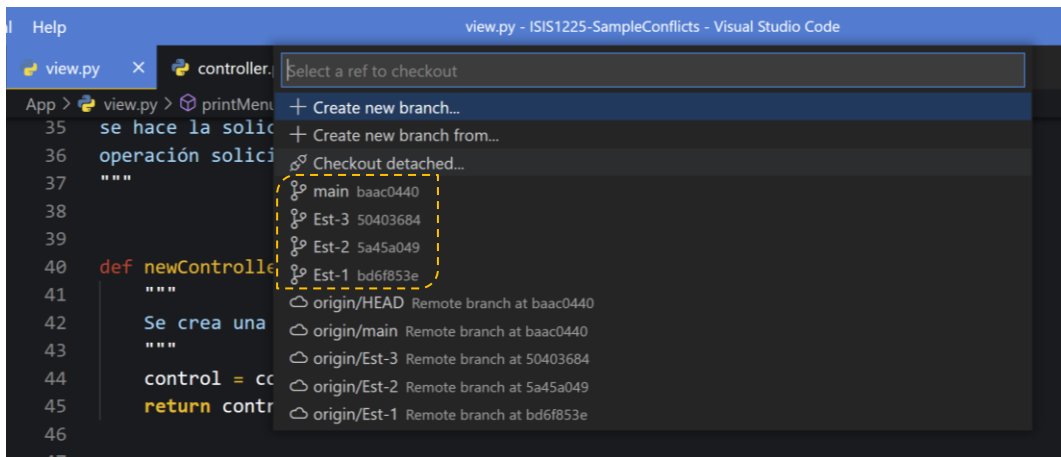


Ilustración 13. Listado de ramas locales activas en GIT desde VS Code.

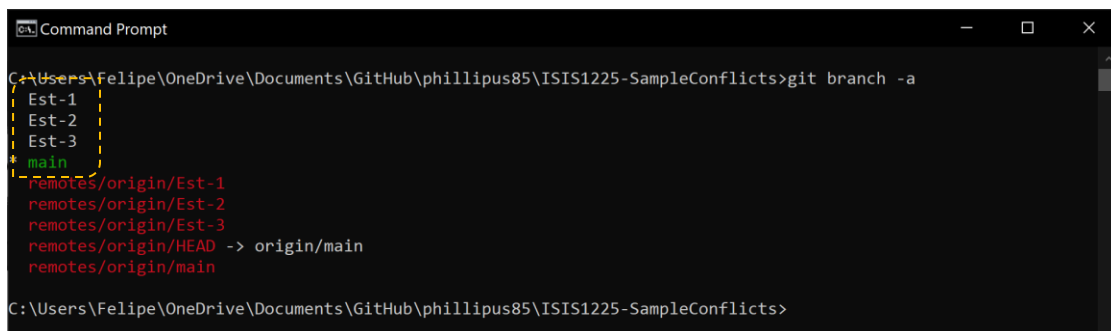


Ilustración 14. Listado de ramas locales activas en GIT desde consola.

## 4.9 A (Estudiante-1) integrar rama Est-3 con main

Recuerde sincronizar los cambios de la copia local del código antes de iniciar la integración utilizando los comandos **Fetch all** y **Pull**.

Antes de iniciar el proceso estudie las instrucciones del video dispuesto por el equipo del curso:

- a. Video Uniandes de como tutorial de como integrar versiones de código desde VS Code titulado [Tutorial de cómo hacer Git Merge](#).

El **estudiante 1** deberá cumplir con los siguientes pasos para integrar la rama **Est-3** con la **main**:

- p.a.1. Ubíquese en la rama **main** (comando GIT Bash: **git checkout main**).
- p.a.2. Abrir la paleta de comandos en VS Code (Ctrl+Shift+P).
- p.a.3. Busque el comando **git merge Branch** como se ve en Ilustración 15.
- p.a.4. Seleccione la copia local de la rama **Est-3** como lo indica Ilustración 16. Esto iniciara el proceso de integración.

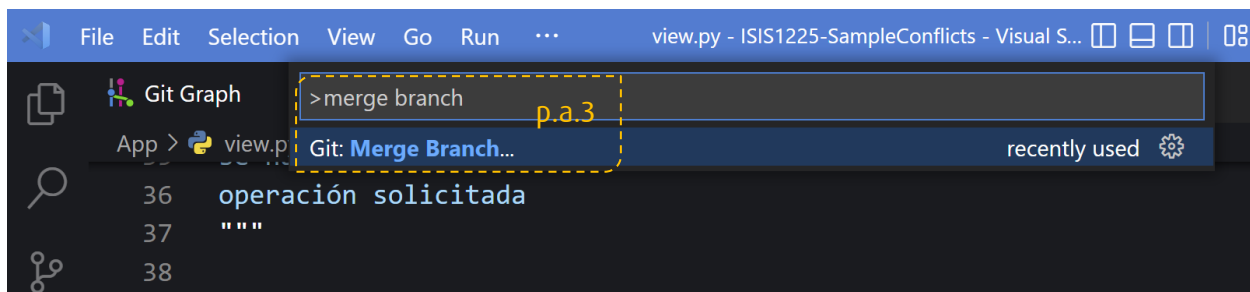


Ilustración 15. Comando GIT Merge dentro del panel de operaciones de VS Code para el estudiante 1.

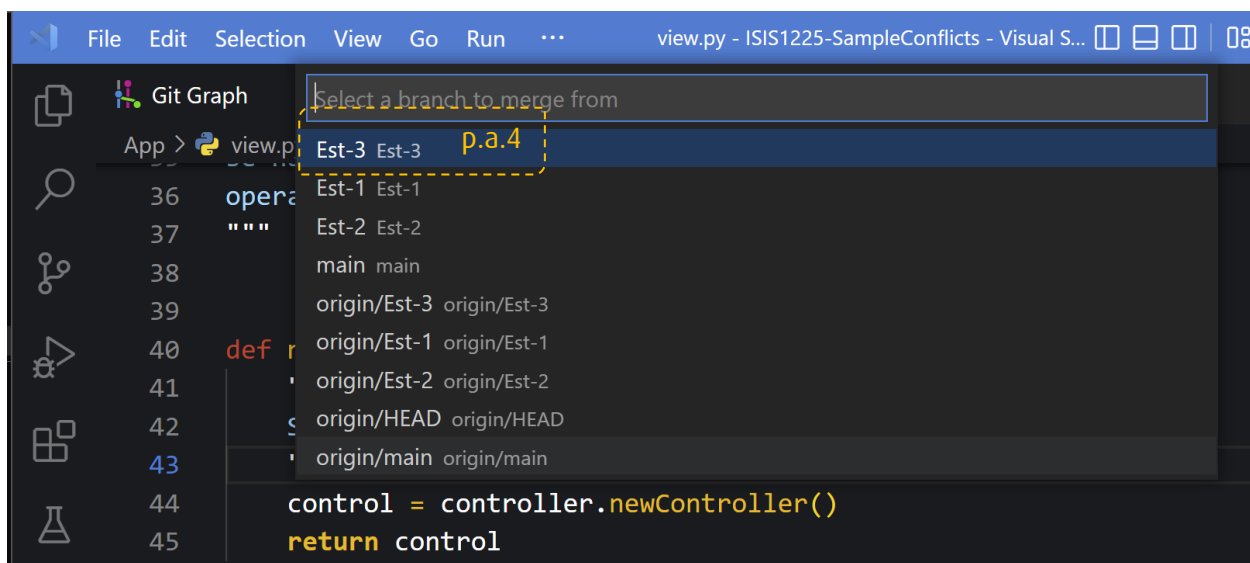


Ilustración 16. Selección de Est-3 dentro del conjunto de ramas locales y remotas activas desde VS Code.

Los bloques de códigos modificados entrarán en conflicto durante la integración. **VS Code** señalará estas inconsistencias en el editor de código. Pero para completar el **Merge** el **Estudiante 1** deberá elegir **MANUALMENTE** la versión que desea mantener en cada uno de los conflictos descritos por el IDE.

p.a.5. Adjunte los cambios del **model.py**, **view.py**, y **controller.py** dentro de la máquina local con el comando **Stage**.

p.a.6. Confirme los cambios con el comando **Commit** (puede dejar el mensaje generado GIT).

p.a.7. Envíe los cambios al repositorio remoto en GitHub con **Push**.

Git Graph

App > controller.py > ...

Theirs origin/Est-3, Est-3

16ac7ad ...

Yours Est-2, origin/Est-2

15fb3a9 ...

```

76 def loadBooksTags(control, filename):
77     # TODO: Modificación de Est-1 y Est-2 en el lab 2
78     tf = os.path.join(cf.data_dir, filename)
79     input_file = csv.DictReader(open(tf, encoding="utf-8"))
80     control["model"] = model.createBookTagList(control["model"])
81     for booktag in input_file:
82         model.addBookTag(control["model"], booktag)
83     return model.bookTagSize(control["model"])
84

```

```

72 for tag in input_file:
73     model.addTag(catalog, tag)
74     return model.tagSize(catalog)
75
76 def loadBooksTags(control, filename):
77     # TODO: Modificación de Est-1 y Est-2 en el lab 2
78     catalog = control["model"]
79     btfile = os.path.join(cf.data_dir, filename)
80     catalog = model.addBooks(catalog, btfile)
81     return model.bookTagSize(catalog)
82

```

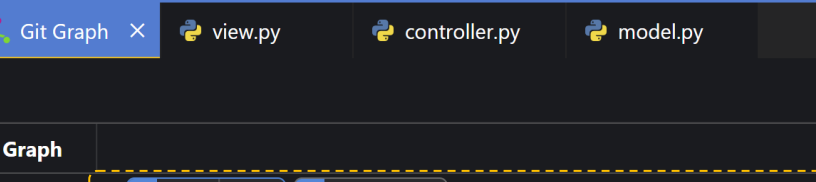
Result App/controller.py

```

72 for tag in input_file:
73     model.addTag(catalog, tag)
74     return model.tagSize(catalog)
75
76 def loadBooksTags(control, filename):
77     # TODO: Modificación de Est-1 y Est-2 en el lab 2
78     tf = os.path.join(cf.data_dir, filename)
79     input_file = csv.DictReader(open(tf, encoding="utf-8"))
80     control["model"] = model.createBookTagList(control["model"])
81     for booktag in input_file:
82         model.addBookTag(control["model"], booktag)
83     return model.bookTagSize(control["model"])
84

```

0 Conflicts Remaining



The screenshot shows the Git GUI interface. The top bar contains the menu items: File, Edit, Selection, View, Go, Run, Terminal, and Help. Below the menu bar, there are tabs for 'Git Graph', 'view.py', 'controller.py', and 'model.py'. The 'Git Graph' tab is active, and the 'Branches' section is visible on the right. The 'Graph' section shows a commit history graph. The 'main' branch is highlighted with a dashed yellow box. The commit history shows a sequence of commits on the 'main' branch, with the most recent commit being 'Merge remote-tracking branch 'origin/Est-3''. The commit messages are: 'Cambios en main', 'Cambios Estudiante 3', 'Cambios Estudiante 2', and 'Cambios Estudiante 1'. The commit dates are 'Actualizaciones enunciado 2022-20' and 'Actualizaciones enunciado lab 2022-20'.

pg. 19

## 4.9 B (Estudiante-2) integrar rama Est-1 con main

Recuerde sincronizar los cambios de la copia local del código antes de iniciar la integración utilizando los comandos **Fetch all** y **Pull**.

Antes de iniciar el proceso estudie las instrucciones del video dispuesto por el equipo del curso:

- a. Video Uniandes de como tutorial de como integrar versiones de código desde VS Code titulado [Tutorial de cómo hacer Git Merge](#).

El **estudiante 2** deberá cumplir con los siguientes pasos para integrar la rama **Est-1** con la **main**:

- p.b.1. Ubíquese en la rama **main** (comando GIT Bash: **git checkout main**).
- p.b.2. Abrir la paleta de comandos en VS Code (Ctrl+Shift+P).
- p.b.3. Busque el comando **git merge Branch** como se ve en Ilustración 19.
- p.b.4. Seleccione la copia local de la rama **Est-1** como lo indica Ilustración 20. Esto iniciara el proceso de integración.

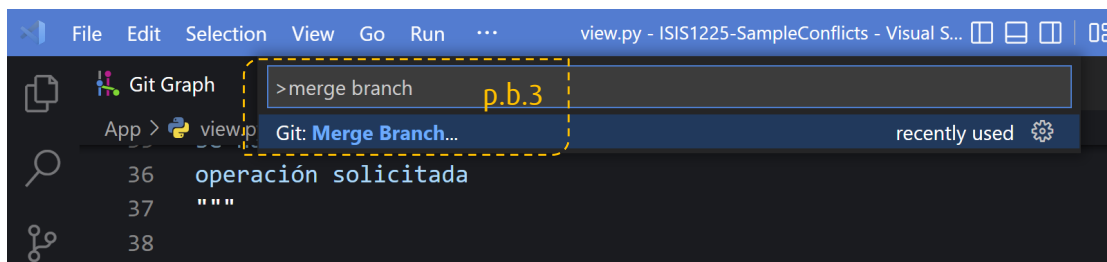


Ilustración 19. Comando GIT Merge dentro del panel de operaciones de VS Code para el estudiante 2.

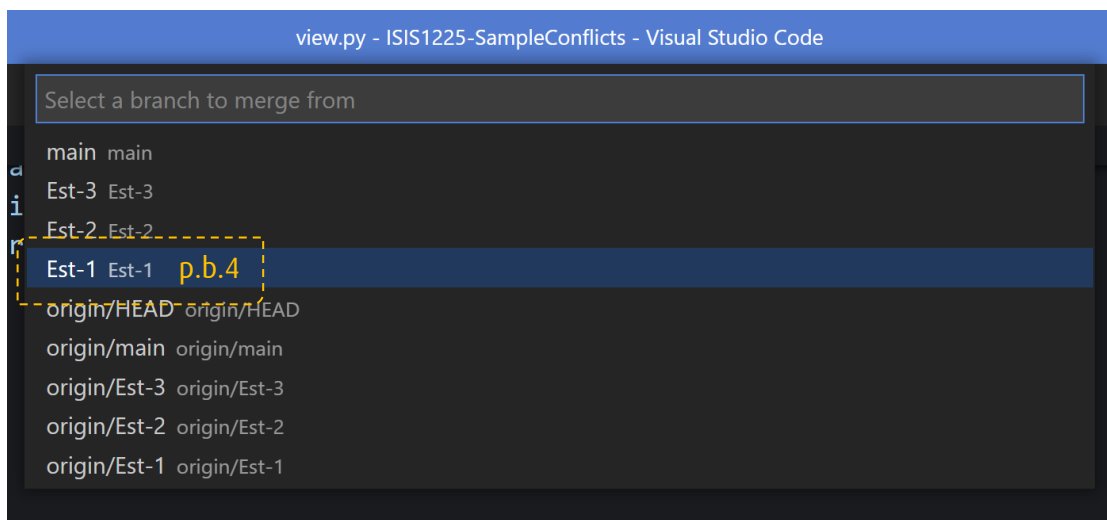


Ilustración 20. Selección de Est-1 dentro del conjunto de ramas locales y remotas activas desde VS Code.

Los bloques de códigos modificados entrarán en conflicto durante la integración. **VS Code** señalará estas inconsistencias en el editor de código. Pero para completar el **Merge** el **Estudiante 2** deberá elegir **MANUALMENTE** la versión que desea mantener en cada uno de los conflictos descritos por el IDE.

Como se ve en la Ilustración 21, el estudiante deberá seleccionar con cuál de las modificaciones desea mantenén. Después de seleccionar todos los cambios preferidos el estudiante deberá completar las siguientes instrucciones:

- p.b.5. Adjunte los cambios del **model.py**, **view.py**, y **controller.py** dentro de la máquina local con el comando **Stage**.
- p.b.6. Confirme los cambios con el comando **Commit** (puede dejar el mensaje generado GIT).
- p.b.7. Envíe los cambios al repositorio remoto en GitHub con **Push**.

Al finalizar el **Merge**, la rama **Est-1** deberá estar completamente integrada con **main** y el historial renderizado del GitHub deberá mostrar algo similar a lo expuesto en Ilustración 22.

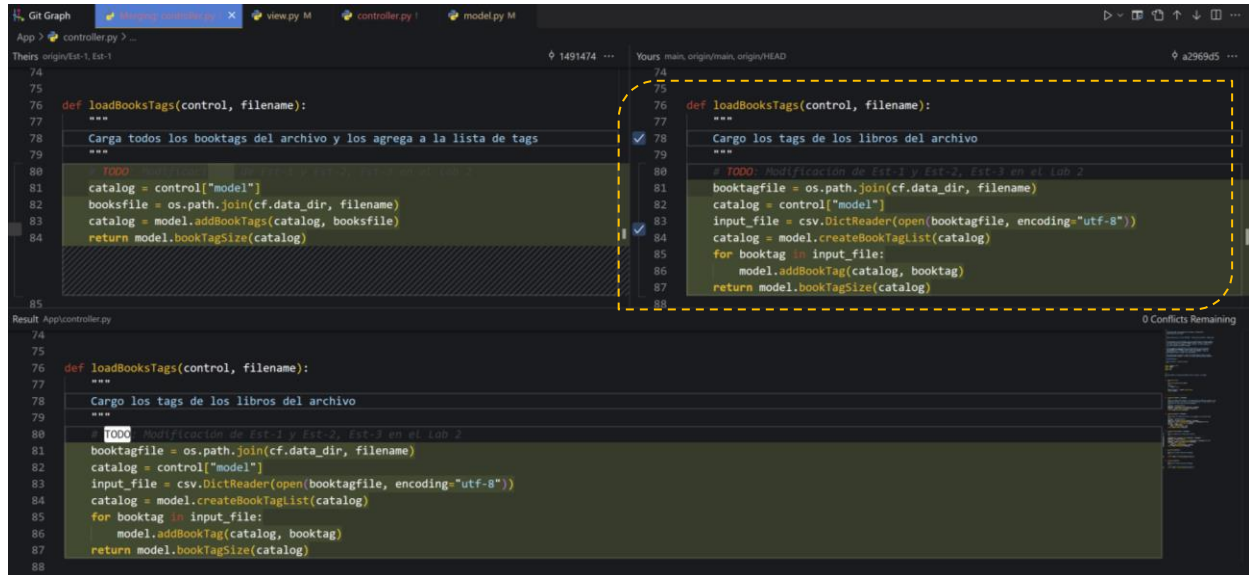


Ilustración 21. Interfaz para la solución de conflictos en entre las ramas Est-1 y main con VS Code.

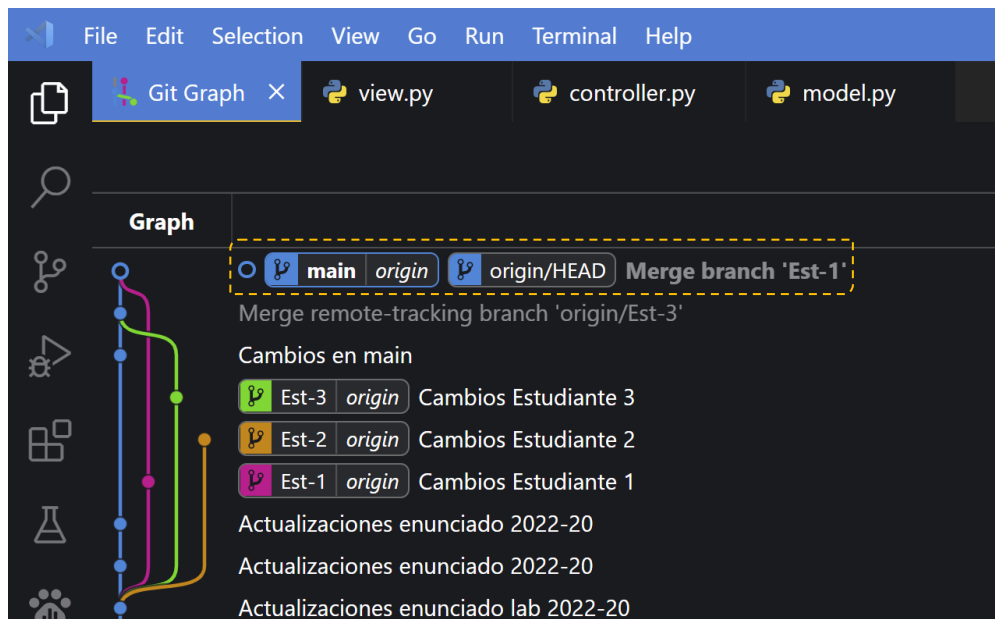


Ilustración 22. Renderizado del historial y el estado de las ramas main & Est-1 integradas dentro de VS Code.

## 4.9 C (Estudiante-3) integrar rama Est-2 con main

Recuerde sincronizar los cambios de la copia local del código antes de iniciar la integración utilizando los comandos **Fetch all** y **Pull**.

Antes de iniciar el proceso estudie las instrucciones del video dispuesto por el equipo del curso:

- Video Uniandes de como tutorial de como integrar versiones de código desde VS Code titulado [Tutorial de cómo hacer Git Merge](#).

El **estudiante 3** deberá cumplir con los siguientes pasos para integrar la rama **Est-2** con la **main**:

- p.c.1.** Ubíquese en la rama **main** (comando GIT Bash: **git checkout main**).
- p.c.2.** Abrir la paleta de comandos en VS Code (Ctrl+Shift+P).
- p.c.3.** Busque el comando **git merge Branch** como se ve en Ilustración 23.
- p.c.4.** Seleccione la copia local de la rama **Est-2** como lo indica Ilustración 24. Esto iniciara el proceso de integración.

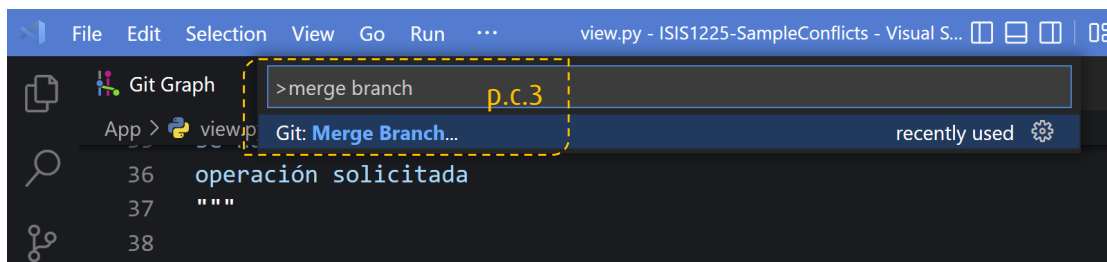


Ilustración 23. Comando GIT Merge dentro del panel de operaciones de VS Code para el estudiante 3.

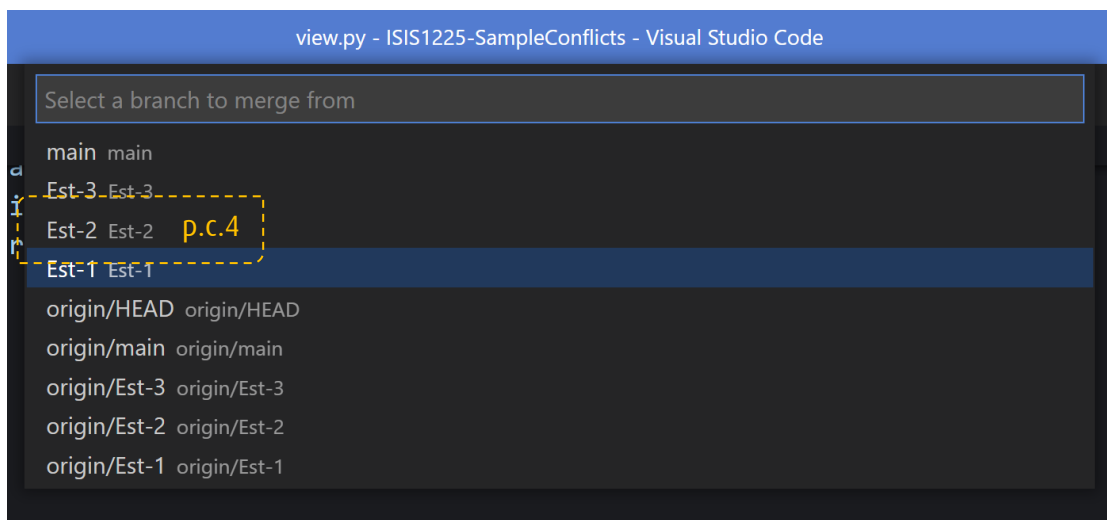


Ilustración 24. Selección de Est-2 dentro del conjunto de ramas locales y remotas activas desde VS Code.

Los bloques de códigos modificados entrarán en conflicto durante la integración. **VS Code** señalará estas inconsistencias en el editor de código. Pero para completar el **Merge** el **Estudiante 3** deberá elegir **MANUALMENTE** la versión que desea mantener en cada uno de los conflictos descritos por el IDE.

Como se ve en la Ilustración 25, el estudiante deberá seleccionar con cuál de las modificaciones desea mantenén. Después de seleccionar todos los cambios preferidos el estudiante deberá completar las siguientes instrucciones:

- p.c.5. Adjunte los cambios del **model.py**, **view.py**, y **controller.py** dentro de la máquina local con el comando **Stage**.
- p.c.6. Confirme los cambios con el comando **Commit** (puede dejar el mensaje generado GIT).
- p.c.7. Envíe los cambios al repositorio remoto en GitHub con **Push**.

Al finalizar el **Merge**, la rama **Est-2** deberá estar completamente integrada con **main** y el historial renderizado del GitHub deberá mostrar algo similar a lo expuesto en Ilustración 26.

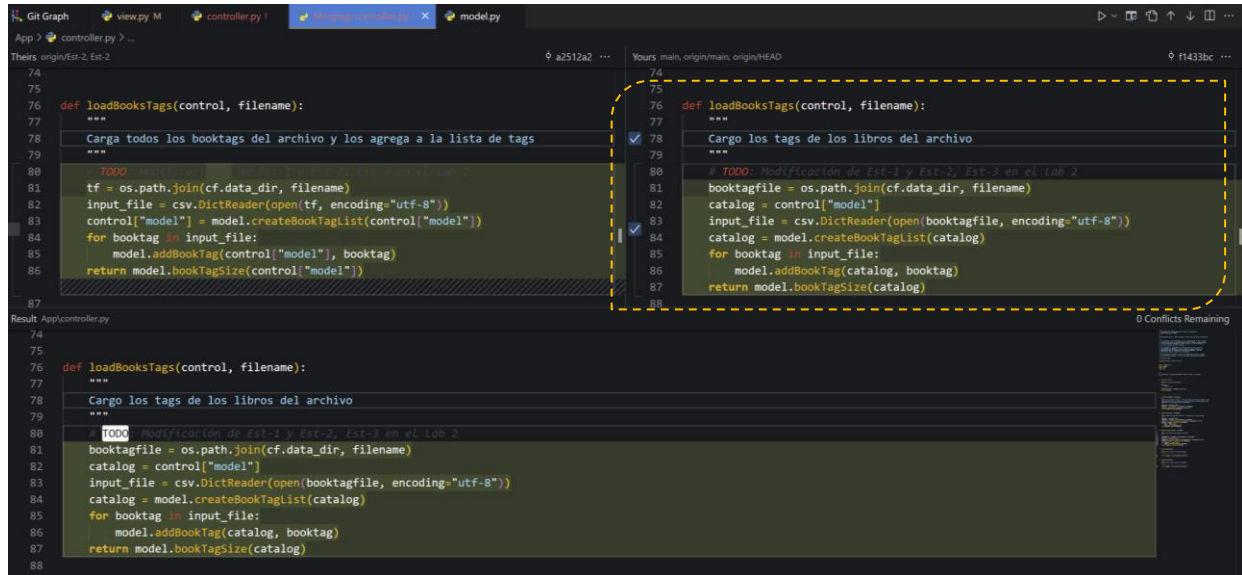


Ilustración 25. Interfaz para la solución de conflictos entre las ramas Est-2 y main con VS Code.

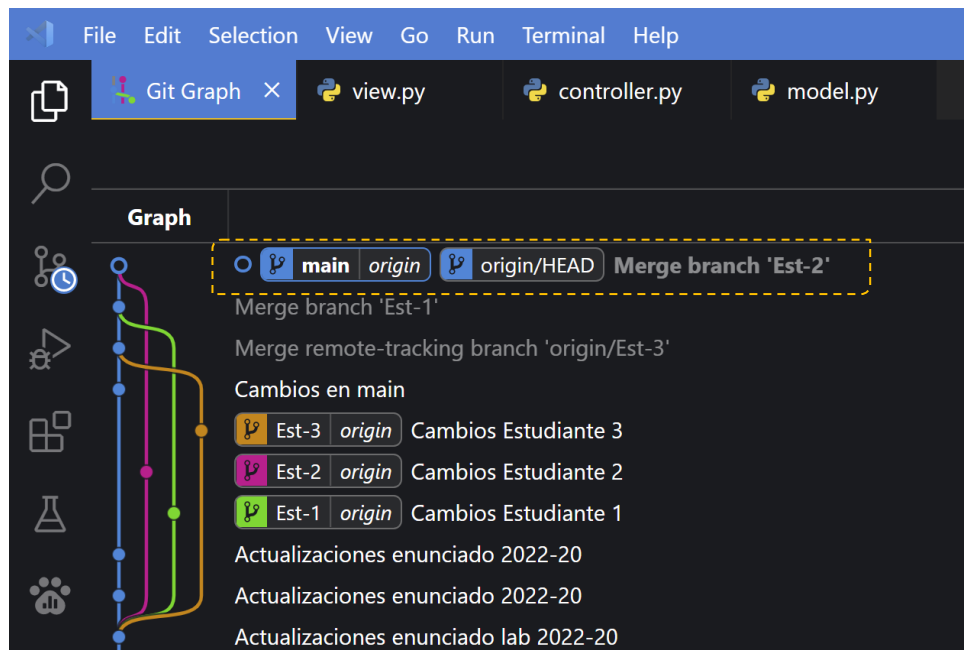


Ilustración 26. Renderizado del historial y el estado de las ramas main & Est-2 integradas dentro de VS Code.



## 4.10 Finalizar actualizaciones en la rama principal

Después de integrar la rama **main** con **Est-2** el estudiante que complete este procedimiento deberá tener la última versión del código utilizando el comando **Pull**. También recuerde siempre sincronizar los cambios del código utilizando los comandos **Fetch all** y **Pull** antes de cualquier modificación.

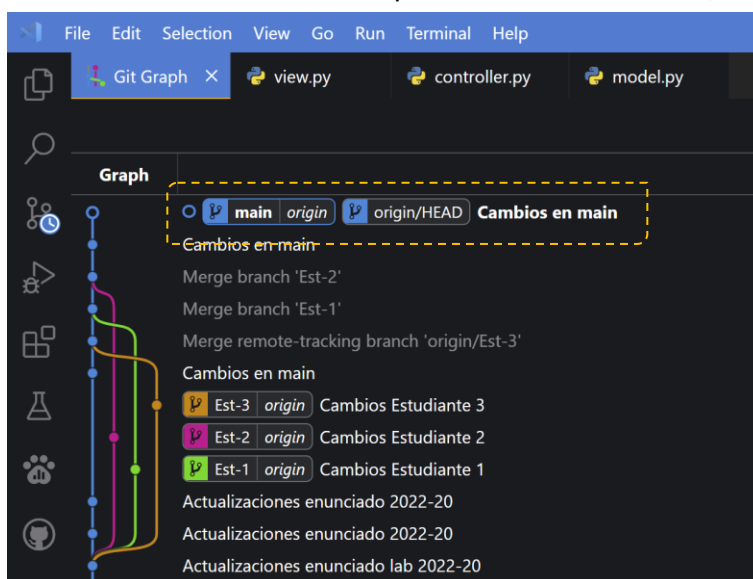
Luego de obtener la última versión del proyecto **uno de los estudiantes** (no importa quien sea) deberá completar las siguientes instrucciones en el **view.py**:

- p.26. modifique las definiciones de las funciones **LoadBook()** y **LoadBooksTags()** removiendo el sufijo “-small” de los archivos que carga la aplicación como lo indica Segmento 21.
- p.27. Adjunte los cambios en su máquina local con el comando **Stage**.
- p.28. Confirme los cambios con el comando **Commit** y el mensaje “**cambios en main**”.
- p.29. Envíe los cambios al repositorio remoto en GitHub con **Push**.

```
def loadBooks(control):  
    """  
    Carga los libros  
    """  
    books = controller.loadBooks(control,  
                                  "GoodReads/books.csv")  
    return books  
...  
def loadBooksTags(control):  
    """  
    Cargar los Tags de libros  
    """  
    # TODO: Modificaciones de Est-1 en el Lab 2  
    booktags = controller.loadBooksTags(control,  
                                         "GoodReads/book_tags.csv")  
    return booktags
```

*Segmento 21. modificaciones main, view.py, definición de las funciones loadBooks() y loadBooksTags().*

Al final de la Ilustración 27 muestra el historial completo de las ramas **Est-1**, **Est-2**, **Est-3** y **main**.



*Ilustración 27. Historial resultante de la práctica de laboratorio para solucionar conflictos en versiones de código.*



## 5 Entrega

### 5.1 Confirmar cambios finales

Confirme los cambios finales siguiendo los pasos aprendidos en prácticas anteriores y adicione el comentario “Entrega Final – laboratorio 2” (**`git commit -m “Entrega Final – laboratorio 2”`**)<sup>7</sup>

No olvide confirmar que sus cambios fueron registrados correctamente al repositorio remoto en GitHub.

### 5.2 Compartir los resultados con los evaluadores

Finalmente, para realizar la entrega de los resultados de la práctica de laboratorio se debe enviar el enlace (URL) del repositorio GitHub a los monitores y profesores de su sección. Para ello, complete las siguientes indicaciones:

- 1) Invitar al profesor de laboratorio y los monitores de su sección a la organización del grupo.
- 2) Incluir en el **README** del repositorio los datos completos de los integrantes del grupo (nombre completo, correo Uniandes y código de estudiante).
- 3) Generar una versión en GitHub con el comentario “Entrega Final – laboratorio 2” (**`git commit -m “Entrega Final – Laboratorio 2”`**) antes de la fecha límite de entrega.
- 4) Enviar el enlace (URL) del repositorio por BrightSpace antes de la fecha límite de entrega.

Recuerden que cualquier documento solicitado durante en la práctica debe incluirse dentro del repositorio GIT y que solo se calificara hasta el último **COMMIT** realizado antes de la media noche (11:59 PM) del **22 de agosto de 2023**.

---

<sup>7</sup> GIT y VS Code no permiten operaciones Commit/Push sin cambios en el código, puede forzar este comportamiento modificando inconsecuentes para el software como lo son los comentarios y la documentación.