

# Ejercicios POLIMENTES

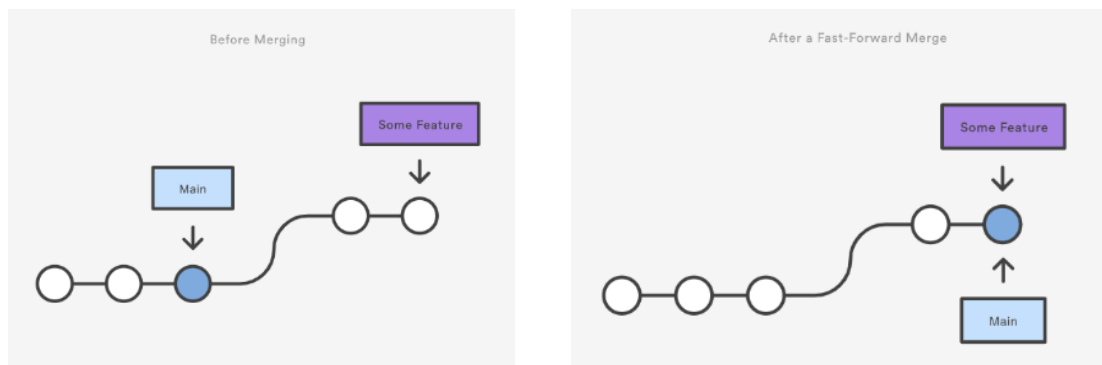
## Capacitación

### Práctica 06082021

#### 1. Conflictos entre ramas

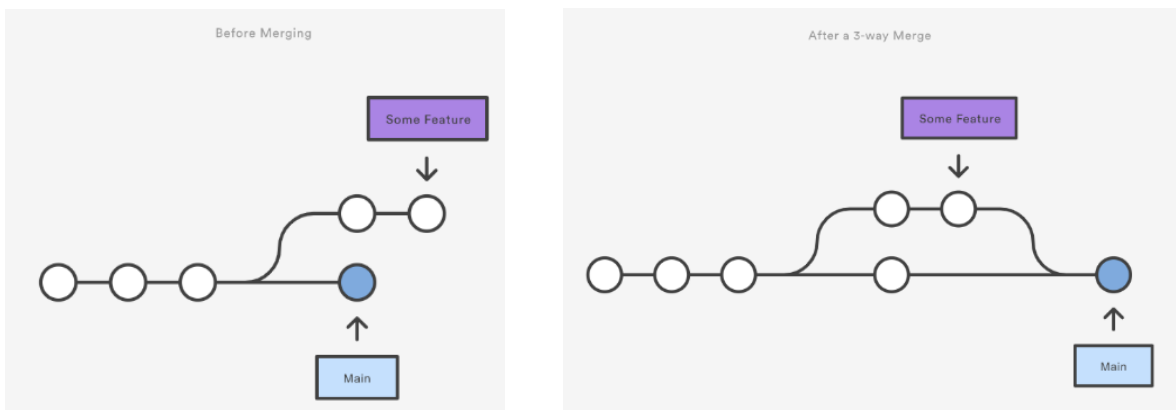
Antes que nada, es importante saber que Git tiene dos maneras de realizar un ‘merge’ o la fusión entre ramas, éstas son:

- **Fast-Forward**: ocurre cuando existe un camino “linear” desde la punta de la rama (último commit realizado en la rama) donde nos encontramos (rama destino) hasta la rama remitente, entonces en este caso Git en realidad en vez de hacer como tal una fusión es moverse a la punta del commit más reciente de la rama remitente. Esto combina ya el historial de ambas ramas y ya todos los commits quedan disponibles desde la rama destino.



- **Recursive o ‘3-way’**: ocurre cuando NO hay un camino linear a la rama destino (normalmente la rama main o master desde donde ejecutaremos el merge) debido a que las rama principal diverge, entonces la opción ejecutar un commit dedicado a unir esa divergencia, ese commit representa su unión.

El nombre “3-way” viene justamente del hecho de que Git utiliza tres commits para generar el merge commit final, dos correspondientes a las puntas de las ramas y su ancestro común.



## ¿Cómo se originan?

Normalmente los conflictos surgen cuando dos desarrolladores del proyecto han cambiado las mismas líneas de un archivo o si algún desarrollador ha eliminado un archivo mientras otro lo estaba modificando. Es importante saber que los conflictos solo afectan al desarrollador que realiza la fusión, el resto del equipo no se entera del conflicto.

```
MINGW64~/Users/fmarc/Documents/Actividades_Polimenes

fmarc@DESKTOP-K1RAAIA MINGW64 ~/Documents/Actividades_Polimenes (main)
$ git branch PruebaConflictos

fmarc@DESKTOP-K1RAAIA MINGW64 ~/Documents/Actividades_Polimenes (main)
$ git branch
  FM-Actividades
  PruebaConflictos
* main

fmarc@DESKTOP-K1RAAIA MINGW64 ~/Documents/Actividades_Polimenes (main)
$ git co FM-Actividades
Switched to branch 'FM-Actividades'

fmarc@DESKTOP-K1RAAIA MINGW64 ~/Documents/Actividades_Polimenes (FM-Actividades)
$ git status
On branch FM-Actividades
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   EjercicioPractica030821.py

no changes added to commit (use "git add" and/or "git commit -a")

fmarc@DESKTOP-K1RAAIA MINGW64 ~/Documents/Actividades_Polimenes (FM-Actividades)
$ git add EjercicioPractica030821.py

fmarc@DESKTOP-K1RAAIA MINGW64 ~/Documents/Actividades_Polimenes (FM-Actividades)
$ git commit -m "Commit desde FM para empezar prueba de conflictos"
[FM-Actividades d2053d6] Commit desde FM para empezar prueba de conflictos
 1 file changed, 1 insertion(+), 1 deletion(-)

fmarc@DESKTOP-K1RAAIA MINGW64 ~/Documents/Actividades_Polimenes (FM-Actividades)
$
```

Figura 1. Creo una nueva rama llamada Pruebas. Me direcciono hacia la rama FM y sobre ella modifico el archivo Ejercicios03082121 y hago el commit correspondiente

```
MINGW64~/Users/fmarc/Documents/Actividades_Polimenes

fmarc@DESKTOP-K1RAAIA MINGW64 ~/Documents/Actividades_Polimenes (FM-Actividades)
$ git co main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.

fmarc@DESKTOP-K1RAAIA MINGW64 ~/Documents/Actividades_Polimenes (main)
$ git co PruebaConflictos
Switched to branch 'PruebaConflictos'

fmarc@DESKTOP-K1RAAIA MINGW64 ~/Documents/Actividades_Polimenes (PruebaConflictos)
$ git status
On branch PruebaConflictos
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   EjercicioPractica030821.py

no changes added to commit (use "git add" and/or "git commit -a")

fmarc@DESKTOP-K1RAAIA MINGW64 ~/Documents/Actividades_Polimenes (PruebaConflictos)
$ git add EjercicioPractica030821.py

fmarc@DESKTOP-K1RAAIA MINGW64 ~/Documents/Actividades_Polimenes (PruebaConflictos)
$ git commit -m "Commit sobre rama Pruebas para conflictos"
[PruebaConflictos 840f5f7] Commit sobre rama Pruebas para conflictos
 1 file changed, 1 insertion(+), 1 deletion(-)

fmarc@DESKTOP-K1RAAIA MINGW64 ~/Documents/Actividades_Polimenes (PruebaConflictos)
$ |
```

Figura 2. Me redirijo a la nueva rama Pruebas y en ella ejecuto un cambio sobre la misma línea del archivo Ejercicios03082021 y realizo su respectivo commit

```
MINGW64~/c/Users/fmarc/Documents/Actividades_Polimenes

fmarc@DESKTOP-K1RAAIA MINGW64 ~/Documents/Actividades_Polimenes (PruebaConflictos)
$ git co main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.

fmarc@DESKTOP-K1RAAIA MINGW64 ~/Documents/Actividades_Polimenes (main)
$ git merge FM-Actividades
Merge made by the 'recursive' strategy.
 EjercicioPractica030821.py | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
```

Figura 3. Regreso a la rama main y hago merge con la rama FM lo cual no representa ningún problema ya que no hay otra rama que interfiera, es decir aquí git usa fast-forward.

```
fmarc@DESKTOP-K1RAAIA MINGW64 ~/Documents/Actividades_Polimenes (main)
$ git merge PruebaConflictos
Auto-merging EjercicioPractica030821.py
CONFLICT (content): Merge conflict in EjercicioPractica030821.py
Automatic merge failed; fix conflicts and then commit the result.

fmarc@DESKTOP-K1RAAIA MINGW64 ~/Documents/Actividades_Polimenes (main|MERGING)
$ git status
On branch main
Your branch is ahead of 'origin/main' by 2 commits.
  (use "git push" to publish your local commits)

You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)
    both modified:   EjercicioPractica030821.py

no changes added to commit (use "git add" and/or "git commit -a")

fmarc@DESKTOP-K1RAAIA MINGW64 ~/Documents/Actividades_Polimenes (main|MERGING)
$
```

Figura 4. Se crea el conflicto a la hora de querer integrar los cambios realizados en la rama Pruebas, ya que se hizo sobre la misma región del documento.

## ¿Cómo solucionarlos?

Cuando dicha situación ocurre, el proceso se detiene justo antes del commit de ese segundo merge que se desea ejecutar para que se resuelva el conflicto manualmente. Se recomienda usar la instrucción 'git status' para conocer cuál es el documento o documentos que necesitan ser tratados.

Al generarse el conflicto git coloca las dos versiones separadas por marcadores sobre el archivo para que el desarrollador elija manualmente la versión definitiva.

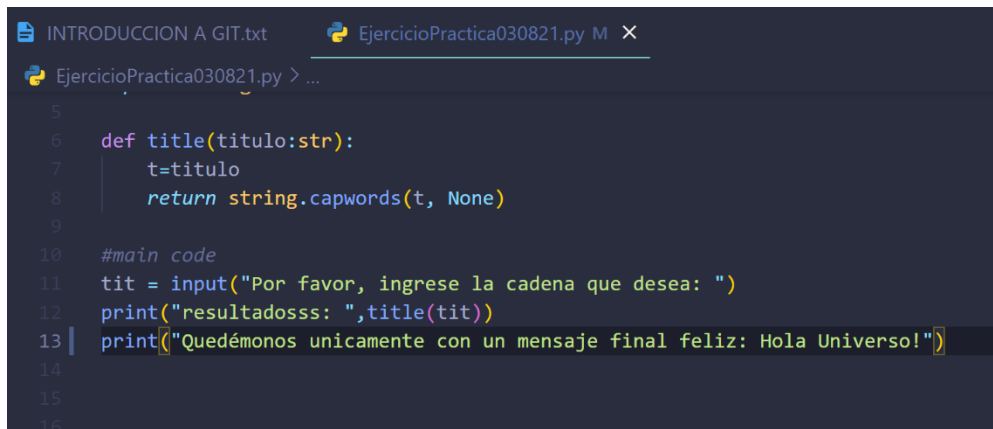
Los marcadores como se puede observar son : <<<<<<<<, ===== y >>>>>>>>. Todo lo previo a ===== es el primer merge que se indicó al sistema que tal cual la rama destino (main) ya tiene y después hasta el marcador >>>>>>>> es la última versión que se busca integrar, el último merge.

```
ODUCCION A GIT.txt EjercicioPractica030821.py 3, 1
cicioPractica030821.py > ...

def title(titulo:str):
    t=titulo
    return string.capwords(t, None)

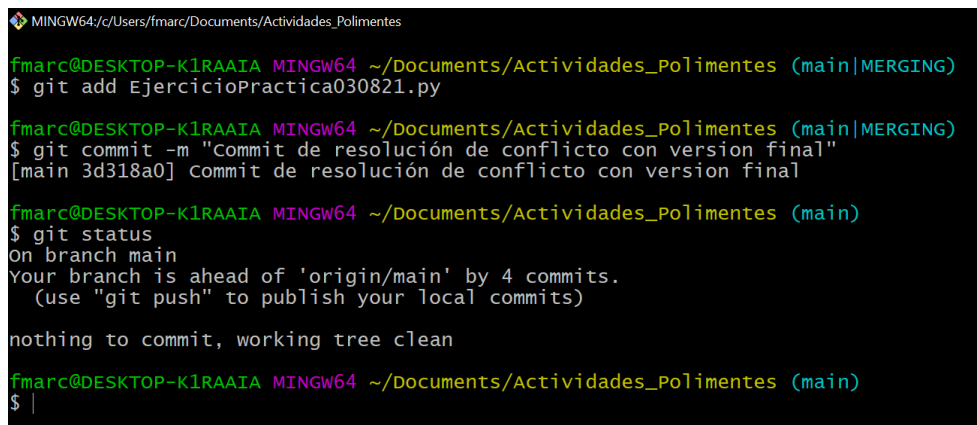
#main code
tit = input("Por favor, ingrese la cadena que desea: ")
print("resultados: ",title(tit))
Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
<<<<<<< HEAD (Current Change)
print("Ahora vamos a probar conflictos en ramas sobre este documento")
=====
print("Este es el texto que modifiko desde la rama de PRUEBAS,aun no es el definitivo")
>>>>>>> PruebaConflictos (Incoming Change)
```

Figura 5. Git modifica el archivo sobre el cual se realizaron los cambios marcando la sección del conflicto.



```
INTRODUCCION A GIT.txt EjercicioPractica030821.py M X
EjercicioPractica030821.py > ...
5
6 def title(titulo:str):
7     t=titulo
8     return string.capwords(t, None)
9
10 #main code
11 tit = input("Por favor, ingrese la cadena que desea: ")
12 print("resultadosss: ",title(tit))
13 print("Quedémonos unicamente con un mensaje final feliz: Hola Universo!")
14
15
16
```

Figura 6. Elegimos la versión final que se quedará en el archivo.



```
MINGW64/c/Users/fmarc/Documents/Actividades_Polimenes
fmarc@DESKTOP-K1RAAIA MINGW64 ~/Documents/Actividades_Polimenes (main|MERGING)
$ git add EjercicioPractica030821.py

fmarc@DESKTOP-K1RAAIA MINGW64 ~/Documents/Actividades_Polimenes (main|MERGING)
$ git commit -m "Commit de resolución de conflicto con version final"
[main 3d318a0] Commit de resolución de conflicto con version final

fmarc@DESKTOP-K1RAAIA MINGW64 ~/Documents/Actividades_Polimenes (main)
$ git status
On branch main
Your branch is ahead of 'origin/main' by 4 commits.
(use "git push" to publish your local commits)

nothing to commit, working tree clean

fmarc@DESKTOP-K1RAAIA MINGW64 ~/Documents/Actividades_Polimenes (main)
$ |
```

Figura 7. Finalmente hacemos add y commit sobre main para fijar la versión final.

[Para más información consultar: <https://www.atlassian.com/es/git/tutorials/using-branches/merge-conflicts> , <https://www.atlassian.com/git/tutorials/using-branches/git-merge> ]

## 2. Diferencias entre ‘git pull’ y ‘git merge’

El comando ‘git pull’ es usado para recuperar y descargar el contenido de un repositorio remoto y de manera inmediata actualizar el repositorio local sobre el cual se corre para igualar el contenido.

De hecho, este comando es una combinación de otros dos comandos, ‘git fetch’ seguido de ‘git merge’. En la primera etapa de operación, el comando pull ejecuta un git fetch que localizará y descargará hasta donde HEAD del repositorio local está apuntando para igualar hasta donde indica el remoto. Una vez descargado el contenido, se ejecutará un merge creando un commit propio actualizando todo por completo, es decir ya se integran TUS cambios locales con los descargados previamente con el fetch.

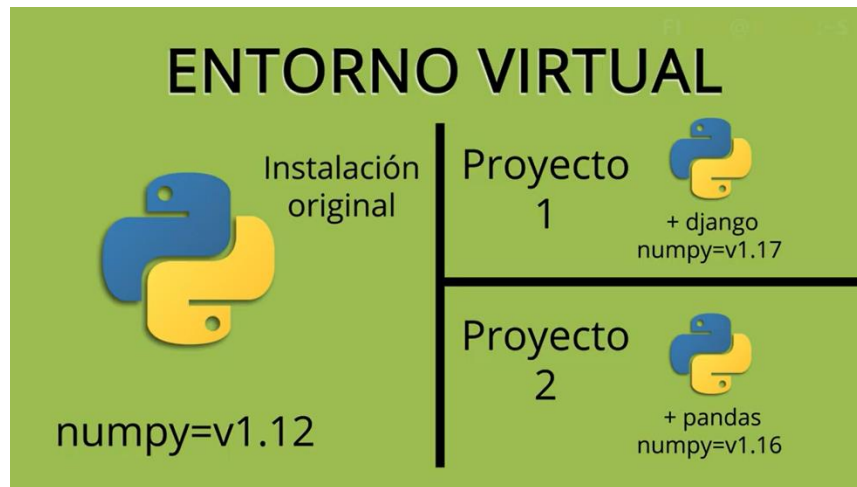
**NOTA:** En los comandos git merge y git pull se puede utilizar una opción (estrategia) ‘-s’ adjunta al nombre de estrategia de fusión que se desees utilizar para realizar la instrucción: recursive, resolve, octopus, ours, subtree, [Consultar: <https://www.atlassian.com/es/git/tutorials/using-branches/merge-strategy>]

## 3. Entornos virtuales

### ¿Qué son y para qué usarlos?

Espacio de trabajo que se crea a partir de una instalación de Python que ya se tiene instalado en el sistema, es decir un entorno de Python parcialmente aislado. En este espacio creado se pueden agregar módulos y

paquetes con el fin de que los use una aplicación en particular sin el riesgo de dañar o “ensuciar” la instalación principal que ya se tiene de Python, esto nos sirve para poder administrar todos los proyectos en distintos ambientes de trabajo sin que nuestros módulos y paquetes choquen entre sí. Por ejemplo si queremos probar un código con distintas versiones de las mismas librerías, al tener dos entornos virtuales diferentes, cada uno con una versión asignada podemos hacerlo.



- Para crear un entorno virtual se utiliza la siguiente instrucción básica: `Python -m venv 'nombre_entorno'`
- Para activar el entorno virtual recién creado es necesario navegar hasta la carpeta 'Scripts' que se crea automáticamente en el entorno recién creado y ejecutar el archivo "activate". Para confirmar que se encuentra activado nuestro entorno podemos fijarnos en la terminal hasta el principio de la línea de comando entre paréntesis el nombre que asignamos al entorno (nombre\_entorno).
- Para desactivar el entorno de trabajo únicamente ejecutamos el comando/archivo "deactivate" dentro de la carpeta 'Scripts' y así regresamos a trabajar dentro de la instalación original y básica de Python.

### Sistemas de paquetes pip

La biblioteca 'pip' nos ayuda a instalar dentro nuestro equipo, así como de nuestro entorno virtual cualquier paquete o módulo necesario para el proyecto (numpy, pandas, Django, etc.).

### ¿Para qué se utiliza el archivo 'requirements'?

Se trata de un archivo txt que contiene todos los paquetes y módulos que instalamos y así compartirlo con nuestros compañeros de proyecto y pueda haber homogeneidad en el trabajo o incluso si deseo configurar una nueva maquina para trabajar en ella y entonces me llevo el archivo para saber exactamente cómo configurar el entorno virtual.

**NOTA:** con el comando *pip freeze* obtengo la lista de todos los paquetes que tengo instalados en el entorno para trabajar.

**NOTA 2:** para generar el archivo de manera automática puedo ejecutar *pip freeze* y enviar toda su salida al nombre 'requirements.txt', de la siguiente manera:

*pip freeze > requirements.txt*

#### **4. Frameworks**

##### **¿Qué es un framework?**

Marco de trabajo, se trata de un conjunto de convenciones, estándares , paradigmas, buenas prácticas y funcionalidades costosas ya desarrolladas, con ellos nos ahorramos tiempo debido a que ya están establecidas funciones, clases y estructuras de directorios ya establecidas, por lo que el usuario solo se encarga de crear ciertas cosas.

Nos ayuda a mejorar nuestro código al obligarnos de cierta manera a seguir buenas prácticas seguir estándares establecidos, facilitando también que nuevos desarrolladores se integren al equipo y trabajen sobre el proyecto y códigos sin ningún problema.

El objetivo principal de un framework es servir como base para los proyectos web, evitar tareas repetitivas, aumentar la productividad gracias a todas las funcionalidades ya integradas, favorecer el trabajo en equipo e infundir buenas prácticas.

Podemos pensar que los Frameworks nos ayudan a tener la siguiente frase en mente “CONSTRUYE TU APLICACIÓN, NO TUS HERRAMIENTAS”, es decir con ayuda de los Frameworks no podemos centrar 100% en el desarrollo del proyecto principal y no preocuparnos por detalles básicos.