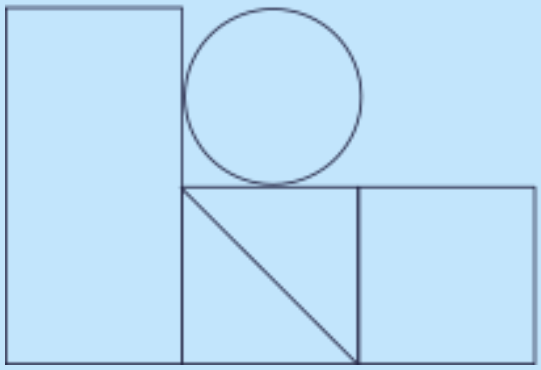
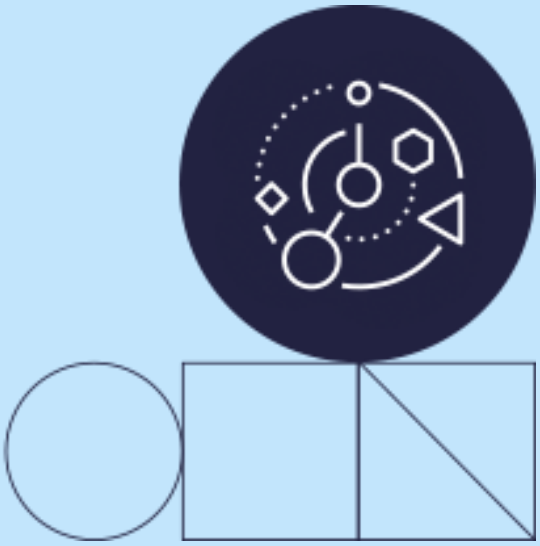
IBM SkillsBuild

IBM

Git y Github

Branch, Merge y conflictos

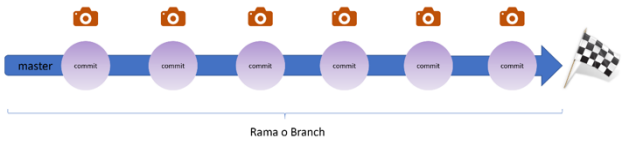




Índice

Branch

Las ramas o *Branchs* son líneas de tiempo que estarán conformadas por todos los "*commits*" que hemos ido haciendo a lo largo de la evolución de nuestro proyecto.



La creación de ramas es una función disponible en la mayoría de los sistemas de control de versiones modernos.

La creación de ramas en otros sistemas de control de versiones puede tanto llevar mucho tiempo como ocupar mucho espacio de almacenamiento.

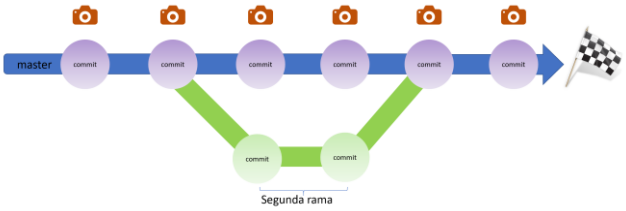
En Git, las ramas son parte del proceso de desarrollo diario. Las ramas de Git son un puntero eficaz para las instantáneas de nuestros cambios. Cuando queremos añadir una nueva función o solucionar un error, independientemente de su tamaño, generamos una nueva rama para alojar estos cambios. Esto hace que resulte más complicado que el código inestable se fusione con el código base principal, y nos da la oportunidad de limpiar nuestro historial futuro antes de fusionarlo con la rama principal.

La rama principal de un proyecto es la rama *master*. Por defecto es la rama en la que se empieza a trabajar.

Nosotros hemos estado trabajando hasta ahora en esta rama master.

Pero *GitHub* nos da la posibilidad de que podamos trabajar en varias ramas simultáneamente. Esto es muy útil para que varios programadores trabajen de forma simultánea sin pisarse unos a otros.

Cada programador creará una rama desde la rama principal, trabajará en ella haciendo los cambios o implementaciones que necesite y cuando haya acabado fusionará su rama con la rama principal. De modo que, al final, en la rama principal quedarán reflejadas todas las implementaciones de todo el equipo de programadores.



Cada programador podrá crear una rama (una "copia") del proyecto. De tal manera que podremos trabajar tanto en el original como en la copia.

Lo ideal es trabajar en la copia y una vez que tenemos la certeza de que nuestro trabajo está perfecto y que no da fallos, lo "inyectamos" en la rama master.

A efectos prácticos es como si nuestro proyecto se dividiera en varios proyectos y cada uno de ellos trabajase de forma totalmente independiente. Una vez que cada "sub-proyecto" está terminado, lo fusionamos con la rama principal.

Tenemos además la ventaja de que, si cometemos algún fallo en el código de la rama secundaria, podemos detectarlo y corregirlo antes de fusionar con la rama master. De esta forma la rama master siempre estará libre de errores.

Cuando se están modificando archivos diferentes en cada rama no suele haber problemas, pero si varios programadores están modificando el mismo archivo en diferentes ramas, puede haber problemas de concurrencia.

Imaginemos que en la rama master modificamos el archivo *index.html* en la línea 70 y que en la rama secundaria modificamos igualmente el mismo archivo en la misma línea. A la hora de fusionar ambos cambios, ¿cuál prevalece? En este caso *Git* nos indicará que hay un conflicto de concurrencia, ya que se está modificando exactamente lo mismo (mismo archivo y misma línea) en diferentes ramas y nos permitirá elegir con cual quedarnos.

Crear un Branch

Para crear una rama deberemos usar el comando *Branch* y especificar el nombre que le queremos dar a la rama:

```
miPC@miPC MINGW64 /g/WORKSPACE/011
GitHub/Proyecto GIT (master)

$ git branch segundaRama
```

Ya tenemos creada nuestra rama. Si hacemos un git log -oneline para ver el estado de nuestro proyecto:

```
miPC@miPC MINGW64 /g/WORKSPACE/011
GitHub/Proyecto GIT (master)

$ git log -oneline

28746f5 (HEAD -> main, tag: v1.0.3,
origin/main, origin/HEAD, segundaRama)
Update index.html
75d4418 Version 1.0.3
fd8ada9 Version 1.0.2
093f1c5 Version 1.0.0
00350f8 Version 1.0.0
```

Podemos ver que ya hay una referencia a nuestra nueva rama (llamada segundaRama).

Con el comando git branch veremos las ramas del proyecto y en qué rama nos encontramos:

```
miPC@miPC MINGW64 /g/WORKSPACE/011
GitHub/Proyecto GIT (master)

$ git branch

* main
  segundaRama
```

Para movernos a la rama secundaria usaremos el comando:

git checkout nombreRama

```
miPC@miPC MINGW64 /g/WORKSPACE/011
GitHub/Proyecto GIT (master)

$ git checkout segundaRama

Switched to branch 'segundaRama'
```

Si volvemos a hacer un git branch:

```
miPC@miPC MINGW64 /g/WORKSPACE/011
GitHub/Proyecto GIT (master)

$ git Branch

  main
* segundaRama
```

Podemos comprobar que ya estamos en la *segundaRama*.

A partir de este momento cualquier cambio que yo haga en mi código se verá reflejado únicamente en la rama secundaria. Los archivos de la rama master no sufrirán ningún cambio.

Si hiciésemos algún cambio en el contenido del archivo index.html, por ejemplo, deberíamos ejecutar un add y un commit. Como vimos anteriormente podemos hacer ambas operaciones a la vez con el comando:

git commit -am “descripción”

```
miPC@miPC MINGW64 /g/WORKSPACE/011
GitHub/Proyecto GIT (master)

$ git commit -am "Versión 1.0.5"


[segundaRama a1bd1cc] Versión 1.0.5
 1 file changed, 2 insertions(+)
```

Ya tenemos un primer cambio hecho en la rama secundaria.

Pero la rama principal no va a reflejar dichos cambios. De hecho, si ahora nos moviésemos a la rama master, los cambios que acabamos de hacer no se verían en el archivo index.html. Y si hacemos un git log –oneline en la rama master, no nos saldrá el *commit* que hemos hecho en la rama secundaria. Es decir, tanto los *commits* como los cambios en los archivos que hagamos en las ramas extra no serán vistos desde otra rama, ni siquiera desde la principal.

Para borrar una rama que tengamos creada:

git branch -d “nombre de mi rama”

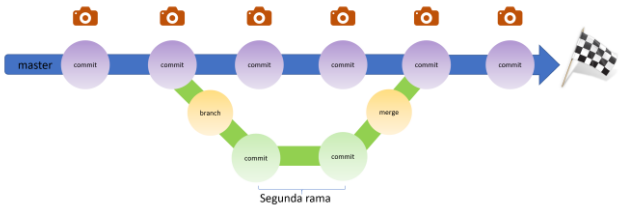
```
miPC@miPC MINGW64 /g/WORKSPACE/011
GitHub/Proyecto GIT (master)

git branch -d segundaRama

Deleted branch segundaRama (was
407013f).
```

Merge

Con el comando merge podremos fusionar una rama extra que hubiésemos creado con anterioridad a la rama master.



En nuestro proyecto teníamos creada una rama extra a la que habíamos llamado *ramaSecundaria*. Para fusionar esta rama con la master deberíamos usar el comando merge junto con al nombre de la rama que queramos fusionar.

Lo primero es movernos a la rama master. Cualquier merge que queramos hacer lo tendremos que hacer siempre desde la rama master. En nuestro caso, a la rama master la habíamos llamado main.

```
miPC@miPC MINGW64 /g/WORKSPACE/011
GitHub/Proyecto GIT (master)

$ git checkout main

Switched to branch 'main'
Your branch is up to date with
'origin/main'.

miPC@miPC MINGW64 /g/WORKSPACE/011
GitHub/Proyecto GIT (master)

$ git merge segundaRama

Updating 28746f5..a1bd1cc
Fast-forward
 index.html | 2 ++
 1 file changed, 2 insertions(+)
```

Los cambios que hicimos en nuestro index.html fueron en una línea en la que nadie más estaba trabajando. Pero supongamos que hacemos un cambio desde la rama secundaria en una línea y que otro programador ha hecho un cambio en la misma línea de la rama máster.

Conflictos

Vamos a insertar código en la misma línea de la rama master y de la secundaria. Al hacer el merge, *Git* detectaría el conflicto y nos sacaría por consola un mensaje como este:

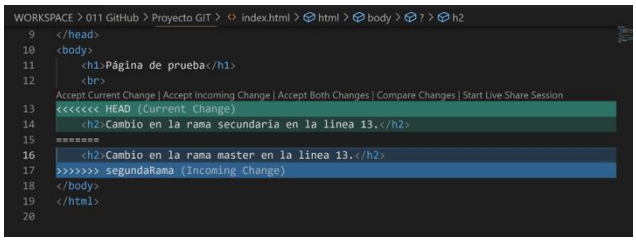
```
miPC@miPC MINGW64 /g/WORKSPACE/011
GitHub/Proyecto GIT (master)

$ git merge segundaRama

Auto-merging index.html
CONFLICT (content): Merge conflict in
index.html
Automatic merge failed; fix conflicts
and then commit the result.
```

Nos está diciendo que hay conflictos, que los arreglemos manualmente y que luego volvamos a hacer el merge.

Y en nuestro *IDE* nos saldrá también un mensaje que nos avisa del problema:



Vemos como *Visual Studio* nos detecta el problema y nos da la opción de aceptar una u otra versión, compararlas, aceptar ambas...etc.

Ahora tendríamos que decidir con qué versión nos quedamos, guardar, hacer el commit y luego el merge.