egg | Proyecto Final Integrador

Trabajando con Git

Instructivo operativo para trabajar con versiones en equipos de trabajo

¡Atención!

Antes de empezar a leer el instructivo es importante que estén familiarizados con algunos conceptos de Git y GitHub presentes en las guías del curso de Full-Stack. Estos no se volverán a abordar aquí, ya que este material tiene como objetivo brindarles un marco de trabajo **operativo** según los roles que adopten en sus respectivos proyectos.

Modos de trabajo

El trabajo con las versiones va a estar dividido en dos roles.

1. **Como desarrolladores (D)** vamos a estar interactuando con el repositorio local y remoto principalmente en dos instancias: al traer actualizaciones de **ramas principales** desde el repositorio remoto hacia el repositorio local y al subir los cambios desde la **rama individual** (la propia) del repositorio local a mi rama en el repositorio remoto.
2. Los alumnos que además de desarrolladores sean **Líderes Técnicos (LT)** van a tener también la responsabilidad de mantener la integridad del código y por ende de las versiones que se integren a la rama principal o ramas principales. Los LT son responsables de hacer algunas configuraciones de seguridad iniciales en GitHub, en donde se alojará el proyecto, y serán quienes autorizarán o no las solicitudes de *merge* (*pull requests)* hacia las ramas principales que los desarrolladores iniciarán desde sus ramas individuales.

Más adelante en este instructivo explicaremos con más detalles las tareas habituales del rol desarrollador y consejos de buenas prácticas generales de trabajo conjunto.

Estructura de ramas del proyecto

- main *o* master
    - developer
        - developer-name1
        - developer-name2
        - developer-name3

- ...

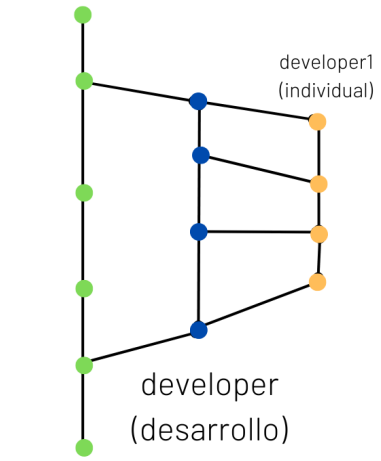


Figura 1

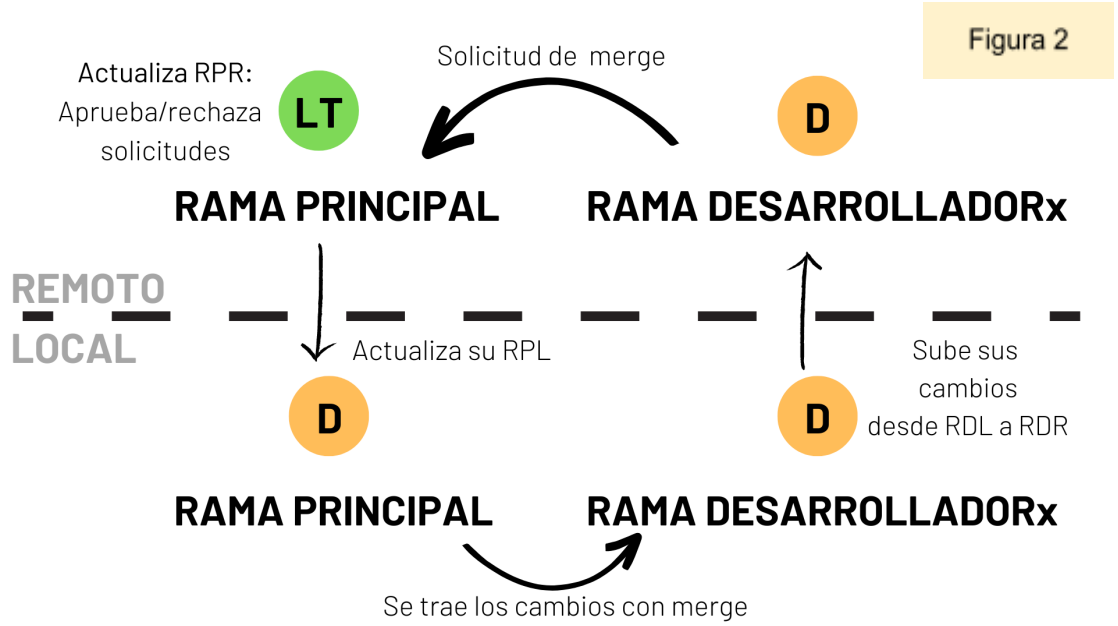
La figura 1 esquematiza la jerarquía de ramas de un proyecto. Al crear el repositorio, nos encontraremos con una rama *main* o *master* a partir de la cual empezaremos a crear las otras ramas. Esta será únicamente accesible por el LT del proyecto que irá sumando los cambios que crea que están listos para unirse a la rama principal (sin errores) cuando por ejemplo hayamos alcanzado uno o más hitos del *sprint*, ya que es esta la versión que mostraremos en cada *demo*. Para ser aún más prolijos con las versiones, crearemos otra rama principal de nombre *developer*, también responsabilidad del LT, que explicaremos en el siguiente apartado. Por último, de esta rama *developer* se desprenderán el resto de las ramas individuales que serán responsabilidad de cada uno de los desarrolladores. Estas ramas suelen llevar el nombre de usuario, el nombre y apellido (nombre-apellido, apellido-nombre) o el id, en caso de haberlo, del desarrollador. Las versiones de desarrollador que se suban también deben, como hábito, cuidar el estilo, no presentar errores de compilación, estar actualizadas con las ramas de origen, etc., pero estos criterios serán aún más importantes cuando se solicite una actualización (*pull request*) con una de las ramas principales.

¿Qué función tendrá la rama intermedia principal *developer*?

En la rama *developer* deberían agregarse todas las modificaciones en estado de prueba **sin errores** y de acuerdo a los requerimientos que hayamos pactado con el resto del equipo (estilos, sin comentarios, testeados, etcétera). Una vez que la tarea de un desarrollador (D) esté finalizada y haya cumplido con todos los requerimientos técnicos y de estilo, el desarrollador podrá solicitar una actualización-merge (*pull request)* con rama developer desde su propia rama. El LT podrá aceptar el merge o rechazarlo y pedirle al desarrollador que haga los cambios necesarios y reenvíe la *pull request*. Una vez que el LT considere que uno o más hitos han sido alcanzados, la versión haya sido *testeada* y no presente *bugs*, hará desde GitHub un merge con la rama *main*.

En la figura 2 podemos ver a grandes rasgos el ciclo básico de transmisión de información de las versiones: primero, desde mi rama en el repositorio local (*DESARROLLADORx*) hacia la rama con mi nombre en el repositorio remoto (que de no existir se crea al ejecutarse el comando *git push origin* 'rama') acción solicitada por los desarrolladores (D). Desde GitHub podremos hacer un *pull request* (solicitud de *merge*) con cualquiera de las ramas de origen (main o developer), aunque lo haremos con la que está más cerca en jerarquía (*developer*). Si el LT aprueba los cambios, aprobará el merge desde GitHub y la rama principal *developer* quedará actualizada. El LT podrá también solicitar un *merge* hacia la rama *main* si considera que los cambios ya están listos para ello. Luego, como desarrollador, desde consola me podré traer los cambios actualizando las ramas principales locales desde las remotas y luego propagando la actualización a mi propia rama desarrollador a través de un *merge*.

Figura 2



Protocolo de trabajo

En esta sección se definirá **una** forma o protocolo de trabajo con Git y se verán los comandos principales que utilizaremos al inicio o bien en la labor diaria, pero dejando afuera otros comandos y procedimientos para casos particulares, de los cuales ya se encargan otras guías y a los que se los invita y alienta a continuar investigando.

Procedimientos de Desarrollador

Creando ramas

Si el LT ya ha hecho las configuraciones iniciales, entonces procederemos a clonar el proyecto y crear nuestras ramas locales. Leer el manual para tener una comprensión más global de las ramas.

Vamos a partir de *main* por lo cual ahora deberíamos replicar la jerarquía del proyecto creando una rama developer a partir de esta.

- Verificamos estar en rama main.
  - ***git status***
- Creamos la rama del mismo nombre que la que tenemos en el repositorio remoto. Podemos corroborar en GitHub.
  - ***git branch developer***
- Nos movemos a rama developer:
  - ***git checkout developer***
- Hasta este punto developer es una copia de main, hay que generar un rastreador, es decir un vínculo con la rama developer remota para luego poder traer los datos. Este procedimiento se puede hacer para vincular cualquier otra rama, reemplazando el primer developer por el nombre de la rama remota y el segundo por el nombre de la rama local.
  - ***git branch --set-upstream-to=origin/developer developer***
- Me traigo la última versión de developer a esta rama.
  - **git pull**
- Creo a partir de rama developer, la rama individual. Así nace mi rama como una copia actualizada que se desprende de la rama principal *developer*.
  - ***git branch mi-nombre***
- Me muevo a mi rama para empezar a trabajar. No hace falta establecer un rastreador ya que es una rama nueva y cuando suba mi primera versión al repositorio remoto se creará la rama *mi-nombre* automáticamente.
  - ***git checkout mi-nombre***

Protocolo diario

A la hora de trabajar en equipo es muy importante que adquiramos algunos hábitos, sobre todo en **dos momentos claves**: antes de sentarnos a trabajar en nuestro código y antes de subir cambios a nuestra rama remota, realizaremos los pasos que están a continuación.

Proyecto Final Integrador

PASO 1: Al comienzo del trabajo diario

1	git status	Verifico en qué rama estoy y si hay cambios por agregar
2	git checkout main	Me cambio a rama main
3	git pull	Actualizo desde repositorio remoto
4	git status	¡Sí de nuevo y cada vez! Es mejor siempre saber dónde estoy.
5	git checkout developer	Me cambio a rama developer
6	git pull	Actualizo desde repositorio remoto
7	git status	'git status' es mi mejor aliado
8	git checkout *mi-rama*	Reemplazo *mi-rama* por el nombre de la rama individual
9	git merge developer	Propago los cambios desde developer local a rama individual.

PASO 2: Al subir cambios:

1	git status	Verificar que esté en mi rama y cuáles son los archivos modificados a agregar al proyecto
2	git add -A	Para agregar todos los archivos de una vez. Verificar la guía en caso de que haya alguno que no quieras subir o uno que siempre deba ser ignorado por Git.
3	git commit -m"..."	Entre comillas una descripción detallada de los cambios efectuados en esta versión. Incluir una oración breve general y luego desplegar observaciones si son necesarias.
4	repetir **PASO 1**	Sí, verifico que no haya nuevas actualizaciones de ramas principales antes de subir mis cambios. Si las hay se deberá hacer nuevamente los pasos 2.1, 2.2 y 2.3.
5	git push origin *mi-rama*	Subo los cambios al repositorio remoto.
6	GitHub: *pull request*	Ver el manual Git del curso para hacer este paso desde GitHub si considerás que está todo listo para solicitar un merge con ramas principales.

Proyecto Final Integrador

Tips de buenas prácticas.

- **Nunca subir una versión a mi rama sin haber hecho una actualización de ramas principales y haberme traído esos cambios con merge**. Esto es particularmente importante sobre todo antes de hacer una solicitud pull desde GitHub.
- **Usar *git status* hasta el cansancio:** luego de levantarme por un café, después de ir al baño, si me habló un familiar. Saber en qué rama estoy va a prevenir que avance mucho en cambios que finalmente voy a tener que deshacer o buscar la forma de trasladar a la rama correcta.
- **Aplicar restricciones a las ramas principales** desde GitHub para prevenir *merge* directos a estas ramas por error. Estas operaciones deben ser aprobadas por el LT que deberá revisar mi código antes de permitir la fusión de ramas.
- **Evitar creación de ramas de desarrollo que no sean desde rama developer** y así evitar que queden ramas desconectadas y sin actualizar.
- **Evitar multiplicidad de ramas de desarrolladores.** Una vez que una rama cumplió su función y se ha hecho el *merge*, eliminar del repositorio remoto (si mi rama sigue existiendo en el local, se volverá a crear en el remoto la próxima vez que haga una solicitud *push*). Mantener la limpieza de ramas en mi repositorio local también va a prevenir posibles errores en versiones.
- **Mantener una coherencia en el nombre de las ramas.** Si se crean más ramas que las principales e individuales de desarrollador, que sean descriptivas de su propósito.
- **Ser descriptivo en los commits**. Es buena práctica que el título tenga el nombre, código o usuario del desarrollador más una descripción breve de cambios.
- **Un solo administrador de versiones** previene posibles errores y diferencias de criterio que entorpezcan la integridad del código. Este rol lo ejercerá el LT del equipo.