***Flujo de trabajo profesional con Pull request***

Antes de introducir el tema de ***Pull request,*** conviene bien dejar clara una noción sobre lo que son los *servidores de desarrollo* (o en ingles, ***staging develop***): básicamente pueden ser definidos como la presentación, de la última versión, de un proyecto en desarrollo (llamese sitio web o aplicación móvil, etc) que está en su última fase de prueba; es decir, es ese espacio previo, para *testear,* lo que queremos *posiblemente* presentar como proyecto final ad portas a ser presentado a un público, usuario, cliente o navegador cualquiera.

Técnicamente los *servidores de desarrollo* tienen su origen en ramas (lo que son) que están justamente por detrás de la rama *master* o *main* (las ramas principales que se suben en el servidor final, la carta de presentación: ***servidor de producción***). Entonces los servidores de desarrollo, que vendrían siendo las ramas que están justamente detrás de la rama *master* o *main,* cuentan también con un servidor que se llama: ***servidor de prueba.***

*Vealo de la siguiente forma:* el ***servidor de producción*** vendría siendo la página web o aplicación movil ya elaborada, que ya superó todas las fases de prueba y que ya está siendo presentada ante usuarios y/o clientes. En cambio, el ***servidor de prueba*** es ese misma página web o aplicación movil en su, *posiblemente,* última versión; la cúal está siendo evaluada por unas últimas fases de prueba para determinar si ya es apta (si se podría considerar o no) como un proyecto finalmente elaborado y exitoso; y poder así ser cargado, posteriormente, en el *servidor de producción*.

*Ahora bien, qué importancia podría tener todo esto en relación a la herramienta de GitHub* ***Pull request?,*** pues que, justo después de que cada desarrollador independientemente haya trabajado en su versión final, según una rama de desarrollo que le haya sido otorgada; todos, sí o sí, deben cargar y fusionar, previa revisión, sus últimos avances *(los que tienen cada uno en sus ramas de desarrollo correspondientes)* primeramente a: la rama del *servidor de desarrollo.* Justo antes de estar todo el código, de todos los desarrolladores con sus respectivas ramas por desarrollador, cargados y fusionados en el servidor de desarrollo: ***se debe revisar*** (como mecanismo de filtro igual, es por eso que es cargado todo el código de todos los desarrolladores, ya fusionado, a esta rama previa a la rama *master: stagind develop*). Ese proceso de revisión que se hace, justo en el momento antes de fusionar en su totalidad (e individualmente) todas las ramas correspondientes a cada desarrollador con la rama *stagind develop,* se le conoce como: ***pull request.***

*Pull request, a modo general, es ese lugar intermedio entre una rama y otra donde se hace un sin número de revisiones (entre lo que exporta una rama e importa otra) antes de llevar a cabo un proceso de fusión entre ambas. Estado intermedio antes de enviar el “merge”.* Esas revisiones se basan o consisten en que otros miembros del equipo puedan ver los cambios que se hicieron, por cada desarrollador, en su rama particular de desarrollo; entonces, hasta que no todos estén de acuerdo en que todos los cambios elaborados era lo esperado, no se hace la fusión entre una rama y la otra (en nuestro ejemplo si una rama de desarrolo en cuestión no gusta, finalmente no podrá hacer un *merge* con *stagind develop*). Ahora bien, en caso que a todos les guste los cambios, éstos son aprobados; y, al aprobarlos, se auto-ejecuta la fusión *merge* en la rama destino; en este caso, siguiendo la línea historica del ejemplo, la rama destino sería *stagind develop.*

Ahora, para hacer la fusión entre *stagind develop* y la rama *master* también se debe pasar por un proceso de ***pull request (herramienta exclusiva de GitHub).***

*un dato: la persona que coordina todas estas revisiones y avala finalmente los cambios y las fuciones entre ramas son los lideres de equipo, muchas veces técnicamente llamados como:* ***DevOps.*** *Un administrador del entorno de desarrollo en su conjunto y que le facilita el trabajo (y lo hace más eficiente) a los desarrolladores.*

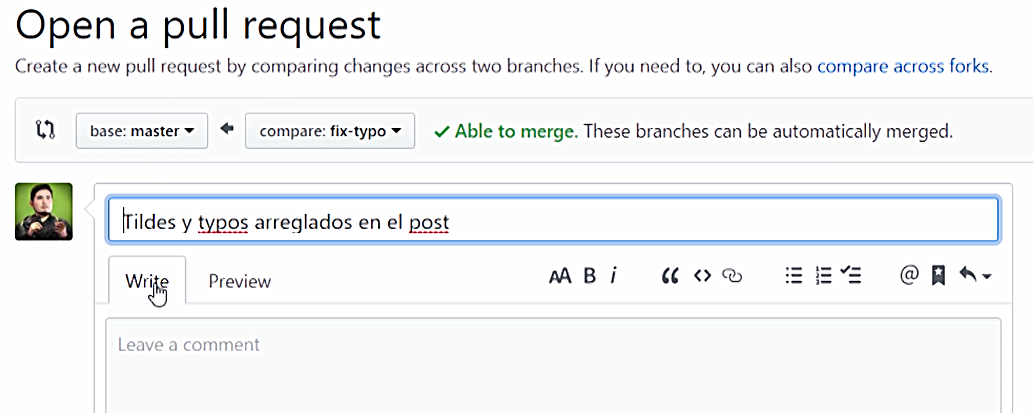
*un dato: es una buena práctica también crear, a partir de la rama base, una rama de desarrollo que consista en corregir todos los errores de todas las demás ramas de desarrollo en cuestión, a esta rama se le suele poner un nombre relacionado con la palabra “fix... ”, esta rama puede ser empujada luego a GitHub.*

***Pull request en acción***

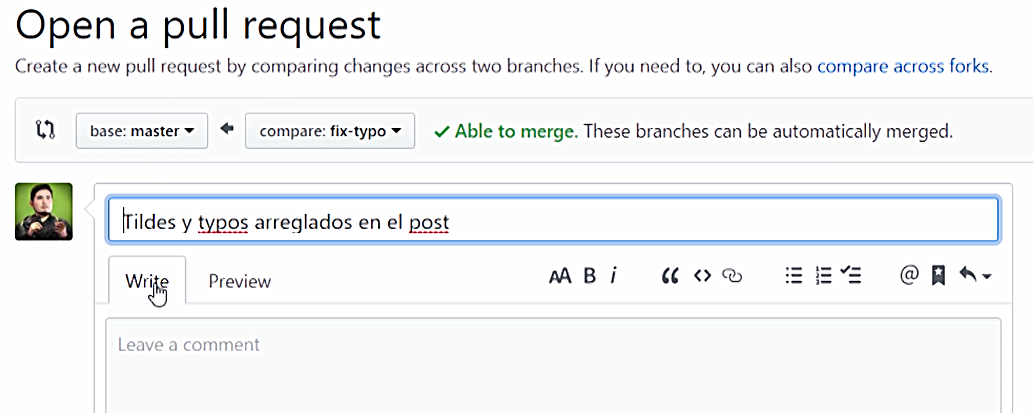
Toda vez que en GitHub registremos un nuevo cambio o se carguen nuevos *commits*, de alguna rama de desarrollo por ejemplo (por parte de un desarrollador tercero), el mismo GitHub *por un tema de buena práctica* nos preguntará si deseamos agregar esos *nuevos cambios,* recién hechos en una rama de desarrollo, a nuestra rama master; pidiendonos primero que, nosotros como propietarios del repositorio remoto en cuestión, revisemos los cambios cargados *(por otro desarrollador comúnmente)* y decidamos si estamos de acuerdo o no con ellos para agregarlos: los cambios cargados en la rama de desarrollo de interés, finalmente a la rama *master.*



Presionado ya el botón de *“Compare & pull request”*, veremos algo más o menos así:



En este ejemplo, se supone que se registraron y cargaron nuevos cambios en la rama de desarrollo *“fix-typo”;* y GitHub, siendo nosotros los propietarios del repositorio remoto, nos pregunta si estamos de acuerdo o no en que esos cambios guardados en *fix-typo* se agreguen a nuestra rama principal *master*. Justo en:



Luego, nos pone un titulo borrador de con qué nombre se guardará el registro del *pull request* que, *posiblemente*, llevaremos a cabo entre la rama *master* y la rama *fix-typo: “Tildes y typos arreglados en el post”;* se dará cuenta que este *titulo borrador* lo asigna intuitivamente GitHub en base al nombre, o mensaje refencial, con el que hemos registrado el *commit*, justo antes de ser empujado a GitHub, en la rama de desarrollo con la que estamos intentando hacer el *merge;* en este caso, esa rama de desarrollo es: *fix-typo.* Es decir, lo que le hace falta a *master* y que tiene *fix-typo* es un último *commit* que fue enviado con el mensaje referencial de: *“Tildes y typos arreglados en el post”;* y, GitHub, sólo quiere saber si usted como propietario del repositorio remoto está de acuerdo o no en agregar esos nuevos cambios a la rama principal *master.*

Posteriormente, debajo sale un recuadro en blanco que sirve para dejar un comentario *(Write).* *Dejar un comentario* es opcional y puede resultar ser una buena práctica; quizás a usted como dueño del repositorio remoto le podría interesar dejar un mensaje particular, en relación a las modificaciones hechas*,* al desarrollador que acaba de cargar cambios (commits) en su rama de desarrollo y que están esperando ser vinculados a rama *master.*

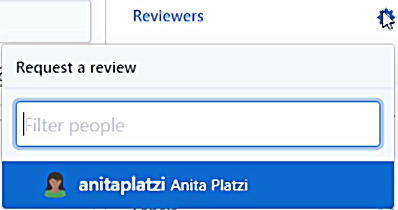
Ahora, si desea ver puntualmente cuáles fueron esos cambios realizados en la rama de desarrollo con la que se está intentando hacer un *merge* a rama *master,* debe irse más abajo. Habrá un editor de código que le señalará, en color verde, el código que fue agregado; y, en color rojo, el código que fue suprimido de una misma línea de código que haya sufrido alteraciones. Veamos el siguiente ejemplo en la línea de código 15:

  
*La palabra “parrafo” pasó a ser “párrafo” y la palabra “increéibles” pasó a ser “increíbles”. Esos fueron los cambios.*

***A usted ahora, como propietario del servidor remoto, le corresponde decidir si esos cambios son válidos o no para hacer une merge con master desde GitHub.***

Sin embargo, si también desea, antes de hacer propiamente el *merge,* puede solicitarle primero a otros desarrolladores que revisen el *pull request* para esperar la validación de todos y decidir si es conveniente o no hacer un *merge* (hasta que todos los solicitados no estén de acuerdo con el *merge,* el *merge* no pasará)*.*

Para invitar a revisión a otros desarrolladores,... eso se hace agregando a un desarrollador de su interés, *o varios*, vinculados al repositorio remoto en cuestión, tal que así: (revise la casilla *Reviewers*):



En este caso estamos suponiendo que sólo tenemos agregado un desarrollador en nuestro repositorio remoto de trabajo actual: *anitaplatzi,* quiere decir que sólo a ella podemos solicitarle una revisión del *pull request.*

Ahora, usted como propietario del repositorio remoto, si solicita una revisión del *pull request* a otro u otros desarrolladores, si bien no se concreta en definitiva el *merge,* está dejando claro que sí está de acuerdo en qué se haga el *merge* en cuestión (como para validar su voto u opinión sobre la decisión o rumbo final que tomen los desarrolladores solicitados para hacer la revisión del *pull request* en discusión). Debe hacer click en:



Los desarrolladores para revisar si les pidieron su opinión (o validación) de un *pull request,* deben situarse sobre el repositorio en cuestión y ver la notificación que le muestre la pestaña *pull request* y hacer click ahí. En caso que estén de acuerdo con el *merge;* es decir, de acuerdo con los cambios que se hicieron en la rama de desarrollo que pretende ser fusionada a *master* (estos cambios los puede visualizar al detalle en la sub-pestaña de la pestaña pull request: *files changed*). Presiona click en:



Pero, si alguno de los desarrolladores no está de acuerdo con los cambios, también puede notificarlo; tal que así:

Luego de que el desarrollador en cuestiónrevise arduamente los cambios realizados en la sesión de *files changed* de la pestaña *pull request* y no le guste lo que está viendo, puede hacer click en:



*Estando ahí, presiona en:*

, *de paso puede escribir un comentario precisando qué fue lo que puntualmente no le gustó de los cambios realizados,... como para no validar el merge; y, adicionalmente, qué sugeriría hacer en cambio.* Finalmente envia su revisión como desarrollador y listo.

Si bien el *merge* no se hace en ese momento, porque hubo al menos un desarrollador de los que se le pidio revisar los cambios; y que, luego de hacer la revisión, no estuviera contento con ellos,... el dueño del repositorio igual puede valerse de su posición de autoridad y validarlos, sobre la opinión de todos los desarrolladores, al ser notificado de que alguno, o varios de los desarrolladores, que no se estuvo de acuerdo con los cambios realizados en la solicitud del *pull request.* Sin embargo, como lider, también podría atender al llamado de atención del (os) desarrollador (es) y trabajar en las mejoras que requieran los cambios según la perspectiva del o de los desarrolladores descontentos con el *pull request* inicial.

Si ese es el caso; entonces, el desarrollador autor de los cambios iniciales que fueron cargados en la rama de desarrollo con la que se buscaba hacer un *merge* a *master*,... debe corregir nuevamente los cambios en ella, ¿cómo? Haciendo un git pull origin + el nombre de rama de desarrollo que requiere ser modificada en su entorno local; si continuamos con el anterior ejemplo, la rama es: *fix-typo,* quedando así: ***git pull origin fix-typo***.

Hacer las respectivas modificaciones solicitadas, cargar nuevo *commit;* y, finalmente, *empujar* de nuevo; tal que así: ***git push origin fix-typo***

Estando en GitHub nuevamente (y ubicado en la rama *fix-typo*), la solicitud del *pull request* inicial nunca se elimina; sin embargo, para comprobar que efectivamente sí se hicieron nuevos cambios *(un nuevo commit cargado y corregido en base a unas sugerencias hechas por otro u otros desarrolladores)* en la rama *fix-typo,* la rama de desarrollo de interés que desea hacer un *merge* con *master;* debe irse de nuevo a la pestaña de *pull request* y entrar a la sub-pestaña: *files changed.*

Comprobado que los cambios en la rama de desarrollo de interés, *fix-typo,* sí se hicieron; todos ya podrían validar el *merge* ahora sí (lo podría validar inmediatamente el propio dueño del repositorio; pero igual, si se quiere, se le pide a los otros desarrolladores, *los que recomendaron hacer un par de cambios*, que ellos mismos autoricen el *merge:* esto se les pide enviandoles un comentario aclarando que *ya los cambios solicitados fueron registrados con exito* -los comentarios se envían desde la sub-pestaña *conversation*).

Ahora, viendo ya como los desarrolladores que solicitaron los cambios, desde el GitHub de ellos... Antes de hacer el *merge,* primero, se valida la revisión, tal que así:

Se ubica nuevamente en:



*Y, estando ahí, presiona en:*



Luego, ese desarrollador colaborador o colaboradores, para que por fin se logre llevar a cabo el *merge,* alguno debe presionar click en:



Luego de fusionada la rama de desarrollo fix-typo a la rama master, por fin, si desea puede eliminar la rama *fix-typo* o dejarla ahí.

Ahora, en su entorno local, debe traerse su última versión del *master,* ramaya fusionada con *fix-typo*, ejecutando el siguiente comando: ***git pull origin master.***

***Fork e introducción a un proyecto Open-Source***

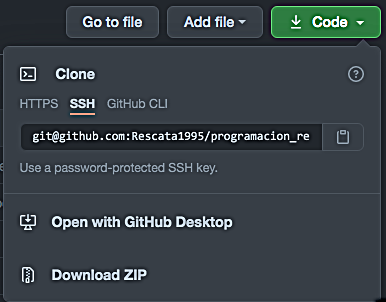
En un proyecto de *Open-Source,* cualquier tercero o persona independiente al proyecto en sí, puede clonarlo (en caso que esté cargado en un servidor remoto como GitHub); sin embargo, no pueden hacer operaciones como: *merge, push, crear ramas o tags, etc.* Lo mucho que podemos hacer, como terceros totalmente desvinculados al proyecto, es copiarlo; esto debido a que es un proyecto *Open-Source,* lo que traduce en ser público y abierto. Adicionalmente, y esto es lo más interesante de la lógica que hay detrás de un proyecto *Open-Source,* uno como desvinculado total del proyecto puede hacer una solicitud de *merge, un pull-request* (sobre algunos cambios que hayamos hecho, en dicho caso; y, posteriormente, propongamos al proyecto real)al dueño del repositorio y a sus colaboradoes vinculados en su conjunto.

*De principio a fin, ¿cómo se llevaría a cabo este proceso? Veamos.*

Primero que todo, si me gusta, debería darle un seguimiento al mismo dentro del repositorio de interés presionando sobre el botón:  (*all activity or watching). ¿*Qué nos permitirá esto?, recibir notificaciones de todas las actividades relevantes que hagan los integrantes directos del proyecto. Toda conversación que ocurra dentro del proyecto, con *watching,* yo la podré ver. Al igual, para dejar claro que el proyecto es de nuestro interés y nos gusta, podemos también ponerle una *estrellita* (nos llegarán de paso avisos para cuando las cosas cambien dentro del proyecto).

Por último, y lo más relevante, es hacer un ***Fork*** del repositorio.***,*** esta herramienta nos permite hacer una copia exacta del estado actual del proyecto y *clonarlo:* tal como si el proyecto pasará ser mío también. De hecho, pasamos a recrear exactamente el mismo repositorio pero, ahora, en nuestra cuenta de GitHub. Claro, sería una copia con exactamente todo lo que tiene el otro (branches, commits, tags, etc); pero, como es copia, no podemos influir reealmente sobre los cambios del repositorio original.

Ahora, lo que nos interesa es tener ese mismo proyecto en nuestro entorno local (para poder manipularlo directamente con Git y hacer los cambios que consideremos pertinentes). Lo que haremos es situarnos en la copia del repositorio, que está ya en nuestra cuenta de GitHub, y hacemos click en el botón *Code,* para *clonar. Dependiendo de qué protocolo de seguridad esté haciendo uso dentro del repositorio, si llaves SSH o HTTPS, usted copiará la URL del proyecto (en este caso estamos suponiendo que se hace uso de llaves SSH).*



Ya sabe qué hacer aquí para clonar un repositorio en entorno local:***git clone URL*** sobre la ruta de interés, dentro de su terminal, donde desee importar el proyecto.

Ya estando el proyecto clonado en su entorno local; usted puede hacer los cambios que considere, en relación idealmente, a las mejoras que pueda sugerir para el repositorio remoto original (a los colaboradores del proyecto real). Suponiendo que encuentra un cambio interesante y lo hace, guardandolo en una copia de seguridad *commit,* usted podrá revisar que el mismo Git tiene la facultad de notificarle qué tanto ha variado su información del proyecto en relación al proyecto clonado desde su repositorio remoto. Si luego del primer *commit* registrado, usted ejecuta el comando ***git status****,* git le dirá lo siguiente:

, quiere decir que su rama, la del entorno local, está más adelante que la rama origin, la que está en su servidor remoto, por un *commit.*

Hecho este procedimiento, ya podríamos hacer un ***git* *push*** (como no existe otra rama más que la rama *master* no se hace necesario especificar a qué rama, de nuestro servidor remoto, queremos exportarle unos cambios: no es necesario poner *git push origin ...* ).

Ahora, ¿cómo podemos solicitar una fusión de nuestros avances, cargados ya en nuestro repositorio remoto de copia, en relación al repositorio remoto real del proyecto?

Pues bien, GitHub es inteligente y sabe que tu repositorio remoto es de origen *o fue originado* a partir de una copia que se le hizo a un repositorio remoto original perteneciente un proyecto real que no tiene nada que ver contigo; es decir, GitHub sabe que hemos hecho lo que, técnicamente, se conoce como: una ***Bifurcación.***

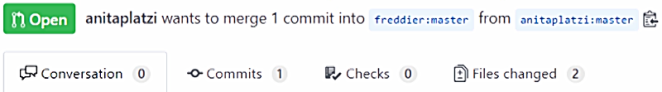
Entonces, GitHub sabiendo eso, nos facilita la tarea de solicitar, al real dueño del proyecto con un repositorio remoto original, un *pull request* en ***New pull request.*** Estando ahí, se deben seleccionar las dos ramas: *la del repositorio del proyecto original,* al que no perteneces ni estás vinculado; *y la rama tuya*, la que proviene de la copia de repositorio que le hiciste al repositorio original y que está situado en tu servidor remoto de GitHub... *especialmente debe seleccionar la rama donde se hizo las modificaciones o le cargaste un commit, con la que le interesa hacer un pull request (fusión); entonces, ambas ramas, la del repositorio original y la tuya, deben ser congruentes de paso.*

Más abajo, mediante una especie de pantalla de edición de texto, usted podrá comparar de manera más visual los cambios que hizo en su copia del repositorio remoto en relación a la edición original con la que obtuvo esa copia del repositorio remoto. Al revisar que todos los cambios que hizo son los que esperaba y los que plantea realmente sugerir a los integrantes del proyecto original (para que sean involucrados en el repositorio remoto original del proyecto en cuestión), usted puede proceder a:  *solicitar un pull request a esos integrantes reales del proyecto; para que su idea, lo que hizo, sea incorporado en él (y se dé un* ***merge*** *en caso que tenga visto bueno).*

dato: en caso que te aburras de esperar una respuesta, también, si deseas, *puedes cerrar el pull request.*

**Ahora veamos esto desde la pespectiva de quién recibe el *pull-request* de un *desconocido*; es decir, desde los dueños reales del proyecto y del repositorio remoto original en cuestión.**

En home y posteriormente en notificaciones, usted como dueño del repositorio remoto real del proyecto, podrá ver que alguien le hizo un *pull-request* a su proyecto. Si presionamos *click* sobre la notificación, veremos que GitHub nos especifica con detalles quién está solicitando hacer un *merge* con nuestro repositorio remoto original a partir del mismo repositorio remoto que fue, si bien recuerda, anteriormente clonado; y que, luego, sufrió de algunas modificaciones,... las cuáles alguien por fuera del proyecto desea incorporarlas al proyecto original por medio de un *pull-request*. Se ve algo más o menos así:



La manera de interpretar esto (lo que está al lado del botón “Open”) es que, yo como dueño del repositorio real del proyecto, veo que alguien desconocido de nombre: “anitaplatzi”, que está completamente por fuera del proyecto original, está solicitando fusionar un avance propio que hizo del mismo proyecto.

Si bien recordamos de nuevo, este proyecto fue clonado en un principio por *anitaplatzi* (recuerde que esto se pudo llevar a cabo mediante la herramienta *fork,* anitaplatzipudo hacer una copia exacta del repositorio remoto del proyecto original en un nuevo repositorio registrado en su cuenta de GitHub; pues, el proyecto original es un proyecto público y *open-source*).

Ahora, para poder visualizar qué fue lo que exactamente modificó la *infiltrada*, en comparación a lo que ya teníamos en el momento en que ella *clonó* el repositorio original del proyecto, hay que irse a la pestaña *files changed.* Ahí podrá visualizar todos los cambios que hizo y, en consecuencia, lo que a ella le interesaría agregar al proyecto original.

Luego procede con un paso que ya conoce: *si decide aceptar (Approve) o no, parcialmente (Request changes), la solicitud de merge.* Esto se decide en el botón *... de paso puede dejarle un comentario explicando la novedad de la decisión que sea toamada.* Sin embargo, si bien recuerda, con *aprobar* los cambios no se hace instantaneamente el *merge.* Debe luego presionar en  y *confirmarlo.*

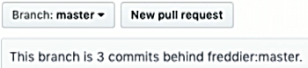
Ahora, los cambios cargados por *anitaplatzi* (junto con todo el historial de *commits*) se han incorporado al repositorio original del proyecto. Puede revisarlo en los *commits* de la rama con la cuál se está trabajando dentro de GitHub, lo mismo pasará si desea exportar esta nueva actualización del proyecto a su entorno local: es lo siguiente que se debe hacer.

El siguiente procedimiento sería, entonces, que actualice su entorno local con el avance actual que ya tiene de su proyecto en el repositorio remoto original de GitHub (al agregarle los cambios solicitados por *anitaplatzi* y que ya fueron fusionados); es decir, haga un ***git pull origin master*** en su PC.

*Luego, puede seguir editando sus archivos, guardar commits y seguir empujando a GitHub.*

***Volviendo a anitaplatzi...***

Ante cualquier cambio o nuevos *commits* empujados por parte del equipo real del proyecto, *anitaplatzi* en la medida se va quedando desactualizada de todas estas modificaciones; de hecho, GitHub propiamente le irá notificando cuántos *commits* han avanzado, *desde cada rama*, los del proyecto original desde el último seguimiento que ella le hizo; se ve más o menos así:

, *plantea que su copia del repositorio original de freddier (dueño real del repositorio remoto y del proyecto en sí), en la rama master, está atrasado por 3 commits.* De hecho, puede visualizar exactamente qué tiene ahora el repositorio original, que no tenga su copia, al comparar a ambos; esto se hace en:  ***compare.*** Y, luego, en 

Entonces, en caso que ella aún quiera hacerle seguimiento al proyecto real en sí y mantenerse de nuevo actualizada con el repositorio remoto original del proyecto, debe...

Al estar situada ahí, donde la dejamos (haciendo las comparaciones, etc), se debe hacer un *pull-request;* es decir, hacer click en: ; luego, posteriormente, debe autorizar el ***merge.*** Y listo.

Sin embargo, esto se puede hacer de una forma más interesante desde la consola, en su entorno local, al crear otra *fuente* para hacer *pull...* para traer los cambios del repositorio remoto original del proyecto. Veamos qué tal.

Comprobamos, de nuevo, con *pwd,* que estamos en la carpeta donde está el repositorio de trabajo actual. Ahora necesito crear un repositorio *remoto* que me traiga los cambios, ya actualizados, de la rama *master (master porque es la rama de trabajo actual, del proyecto real, en la que estamos atrasados... )* del repositorio remoto original del proyecto. Entonces pasamos a *clonar* la rama *master,* del repositorio remoto del proyecto de interés, de *freddier*; copia su URL según el protocolo de seguridad con el que esté trabajando en su repositorio, llaves SSH o HTTPS, y ejecuta lo siguiente en consola: ***git remote add + “nuevo nombre de repositorio” + URL copiada,*** esto nos va a generar una nueva fuente de datos o repositorio *remoto,* diferente a, por ejemplo, *origin.* Y lo puede comprobar ejecutando el siguiente comando:***git remote -v***

*Entonces, como hemos creado ese nuevo repositorio remoto, desde git, necesitamos traerlo a git; traerlo con la rama de nuestro interés, en la que estamos desactualizados con los cambios, la rama master.* Tal que así:

***git pull + “nuevo nombre de repositorio” + master.*** Al hacer esto nos traemos todos los cambios de *master,* en los que estabamos desactualizados, del repositorio original del proyecto que hemos venido siguiendo.

Lo que prosigue entonces es *empujar* estos cambios, ya actualizados, a nuestro repositorio remoto *origin* en la rama *master,...* la que tenemos desactualizada... hasta ahora. Tal que así: ***git push origin master.***

***Ahora, nuestro repositorio anitaplatzi, ya está al día nuevamente con el repositorio remoto original del proyecto, el de freddier.***

***Introducción a gitignore***

Hay archivos que reciben un tratamiento diferente o, al menos, necesitan estar completamente aislados (lejos de la vista pública); para ser más precisos, no pueden hacer parte de tu repositorio remoto (por ej. las contraseñas, etc). En especial, si estás dentro de un proyecto de tipo *open-source* o alguien más tiene la contraseña de tu repositorio remoto, resulta inviable que haya información sencible dentro del servidor, cierto? Pues bien, podemos evitar exponer esta información con la herramienta de trabajo *gitignore.*

*consejo: evite que archivos binarios (como las imagenes .jpg) hagan parte de un repositorio.*

Primero que todo, ubiquese en la raiz o carpeta base donde tiene inicializado *git* en su entorno local*,* tiene que ser la carpeta *git* donde actualmente se encuentra trabajando en un proyecto de interés. Ahí debe crear un nuevo *archivo* que se llame: ***.gitignore (con el punto al principio):*** *aquí, en .gitignore, se hace un listado de todos los archivos que pretendemos ignorar, los que queremos mantener en la confidencialidad.*

*Gitignore* funciona más o menos similar a los comandos de la *Shell* a la hora de querer referirse a una carpeta o a un archivo (digamos, para ejecutar un comando sobre ellos); si quisieramos, por ejemplo, incluir en la lista de *“gitignore”* todos los archivos tipo *.jpg* debemos hacerlo de la siguiente manera: ***\*.jpg*** (le estamos diciendo a Git que se ignoren ***todos*** los archivos *.jpg*); es decir, todos los archivos que agregemos dentro del listado de *.gitignore* no serán subidos a ningún repositorio remoto.Adicionalmente, *git status,* no dejará en evidencia directa ningún cambio que se haga sobre esos archivos que fueron agregados al listado de .*gitignore*; sin embargo, sí será notificado de los cambios que se hagan propiamente sobre la carpeta oculta llamada: *.gitignore.* (Ojo, es sobre la carpeta *gitignore,* no sobre los archivos que estén llamados dentro de ella).

Dicho lo anterior, entonces, cuando le hagamos seguimiento nuevamente a todas nuestras modificaciones, le asignemos un *commit;* y, de paso, empujemos esa copia de seguridad a repositorio remoto; tendremos que, en *GitHub,* no se verán los archivos que, si bien están agregados tangiblemente dentro de su repositorio local, no podrán ser visualizados en el servidor remoto de GitHub al ser cargado un *commit.*

*de interés: en su archivo .gitignore, antes de nombrar a los archivos que desea ocultar, puede poner comentarios con #...*

***Ignorar ficheros o directorios ya trackeados***

*Una cosa a tener en cuenta es que a través de .****gitignore****sólo pueden ignorarse los ficheros antes de que pertenezcan al stage de****GIT****. Es decir, si el fichero ya existe en el control de versiones, no se borrará ni tampoco será ignorado a partir de incluirlo en el .gitignore*

*Tendremos que eliminarlo de la caché de git, para ello utilizaremos el siguiente comando:*

***git******rm*** *[file\_or\_dir] --cached*

*Esto lo eliminará del control de versiones pero no de nuestro proyecto en local. Una vez hecho esto si podemos incluirlo en nuestro .****gitignore.*** Entre otras cosas, es esto lo que le toca hacer siempre con el archivo .DS\_Store propio de Mac. Ahora, si pretende ignorar todos los *.DS\_Store*, no solo los de la carpeta actual; sino, incluso, los de las subcarpetas. En vez de llamar, por ejemplo, **“.DS\_Store”**, sería: **“\*DS\_Store”**. Tal que así:

../../../../Captura%20de%20pantalla%202022-03-18%20a%20las%203.50.46%20p.m..png

***Introducción a README.md***

Los archivos en ***Markdown*** se guardan con la extensión ***.md,*** y es una herramienta que figura ser *buena práctica;* especialmente en proyectos *open-source.* Los archivos de tipo ***markdown*** son una especie de código que nos permiten cambiar *ligeramente* la forma en cómo vemos un archivo de texto (para darle más estilo, por ejemplo); es por lo anterior que, sirven como *carta* *de* *presentación* dentro de un proyecto público; y, debido a eso, son extendidamente usados, con el nombre de: “*readme.md”,* en repositorios remotos como *github.* Ese archivo llamado *readme.md,* si se encuentra trabajado desde un repositorio local vinculado a uno remoto, siempre debe ir en la carpeta raiz del repositorio local y, eventualmente, lo estará *empujando* a un repositorio remoto*.*

Los archivos *readme.md* pueden presentar de manera muy visual, detallada y, digamos, *colorida* todo lo que tiene que ver con, o de lo que se compone, su proyecto; de hecho, es ese su uso extendido cuando se proyectan en un servidor (para que los lean, *read me*). Dicho lo anterior, a este tipo de archivos, se les permite ser gráficos o tener componentes de *estilo* como se ve en el lenguaje *CSS* o tener un poco de *marcado* como en *HTML*; y sí, en los archivos *.md* puedes agregar *HTML.* Entonces, convengamos que, los archivos *.md* son como un punto medio entre un archivo de texto plano y archivo *.html*.

Hay editores online especializados en el tratamiento de archivos tipo *.md*, uno muy popular es: ***editor.md***, nos asiste bastante bien si queremos presentar de manera *lujosa* nuestro proyecto en *github.*