Universidad Central de Venezuela
Facultad de Ciencias
Aplicaciones con la Tecnología Internet
Noviembre, 2017

Laboratorio #6: Trello & Git

1. Objetivos

Conocer el funcionamiento de las herramientas de gestión de proyectos de software con la finalidad de llevar una correcta planificación al momento de desarrollar un determinado proyecto, y comprender el significado de un software de control de versiones, las funcionalidades que provee, así como también su utilidad.

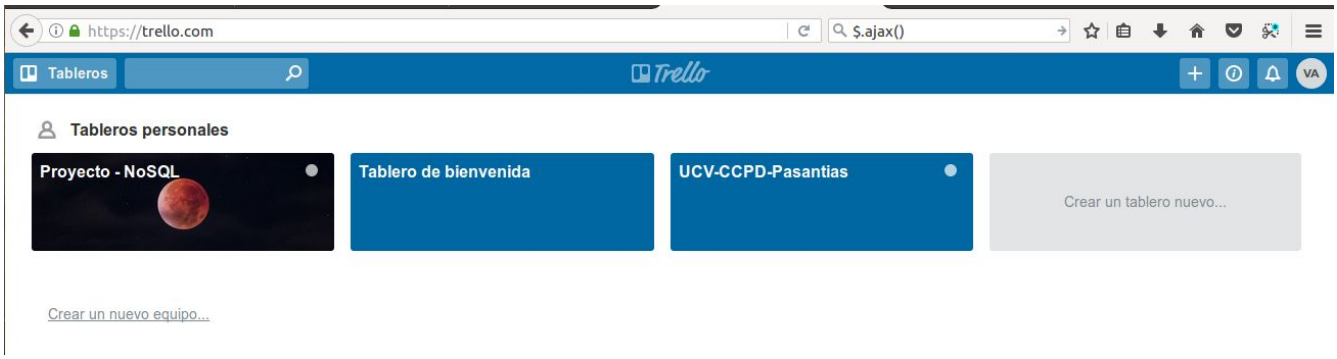
2. Metodología

El laboratorio está dividido en dos partes, la primera consta de un cierto número de actividades que conforman el uso de Trello como herramienta de gestión de proyectos, dichas actividades se realizarán junto con el preparador, y la segunda parte conformada por actividades que hacen referencia al uso de Git, cómo configurarlo, cómo crear una cuenta, y todos los aspectos concernientes al uso de este tipo de sistemas de control de versiones.

3. Trello

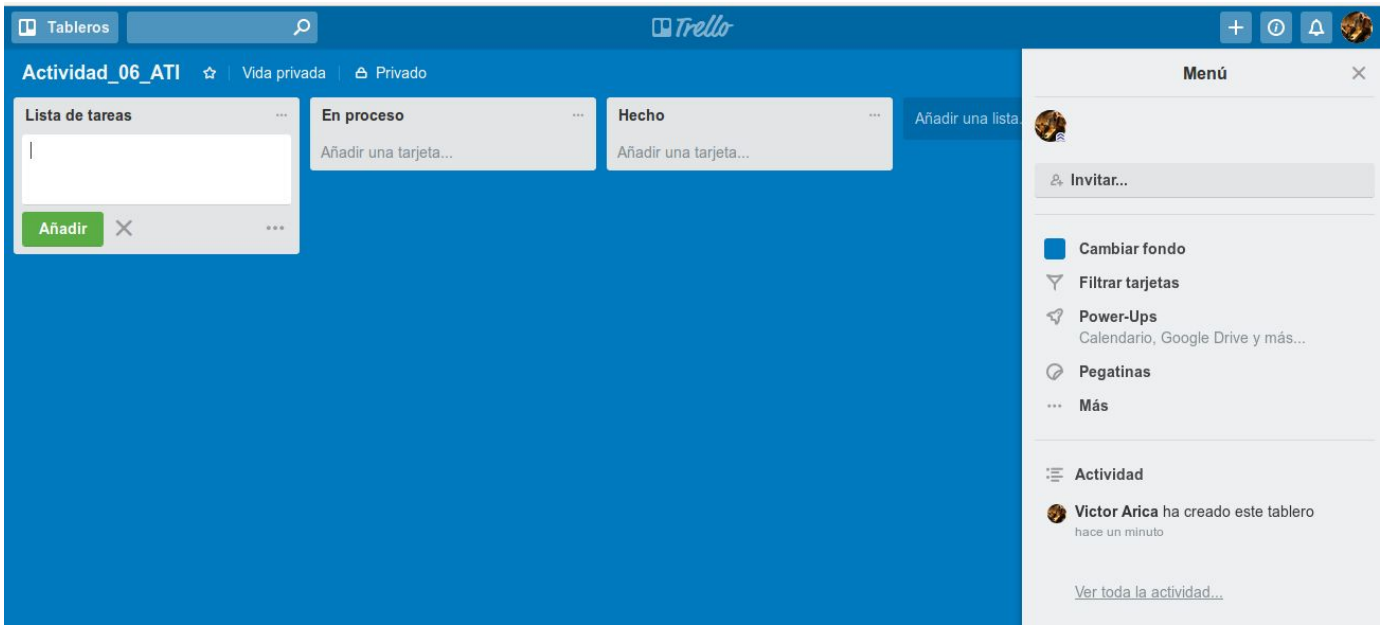
1. Configuración

Trello es una herramienta online, por lo tanto, no es necesario instalarla localmente en el sistema, basta con crear un usuario en https://trello.com/ para acceder a la página principal donde registraremos nuestras actividades:



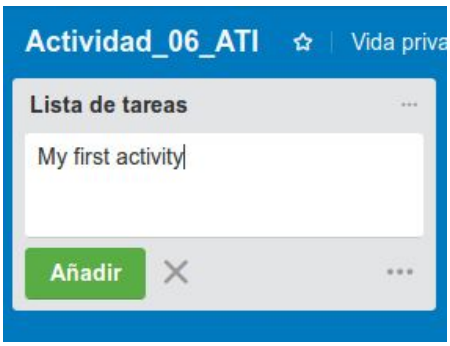
Trello básicamente es una herramienta colaborativa que permite gestionar las actividades realizadas o que se encuentran por realizar y planificarlas mediante el uso de tableros, donde cada tablero hace referencia a un proyecto determinado, si queremos observar su funcionamiento, es necesario crear un nuevo tablero:

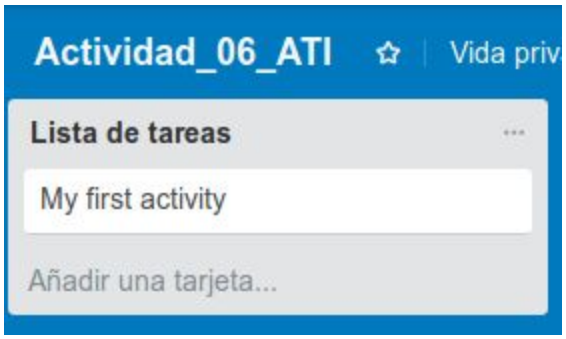
Una vez realizada esta opción se despliega una interfaz donde aparece un cuadro de texto que almacenará el título del tablero, lo completamos colocando "**Actividad_06_ATI**" y le indicamos que el tablero es privado, luego procedemos a crear el tablero.



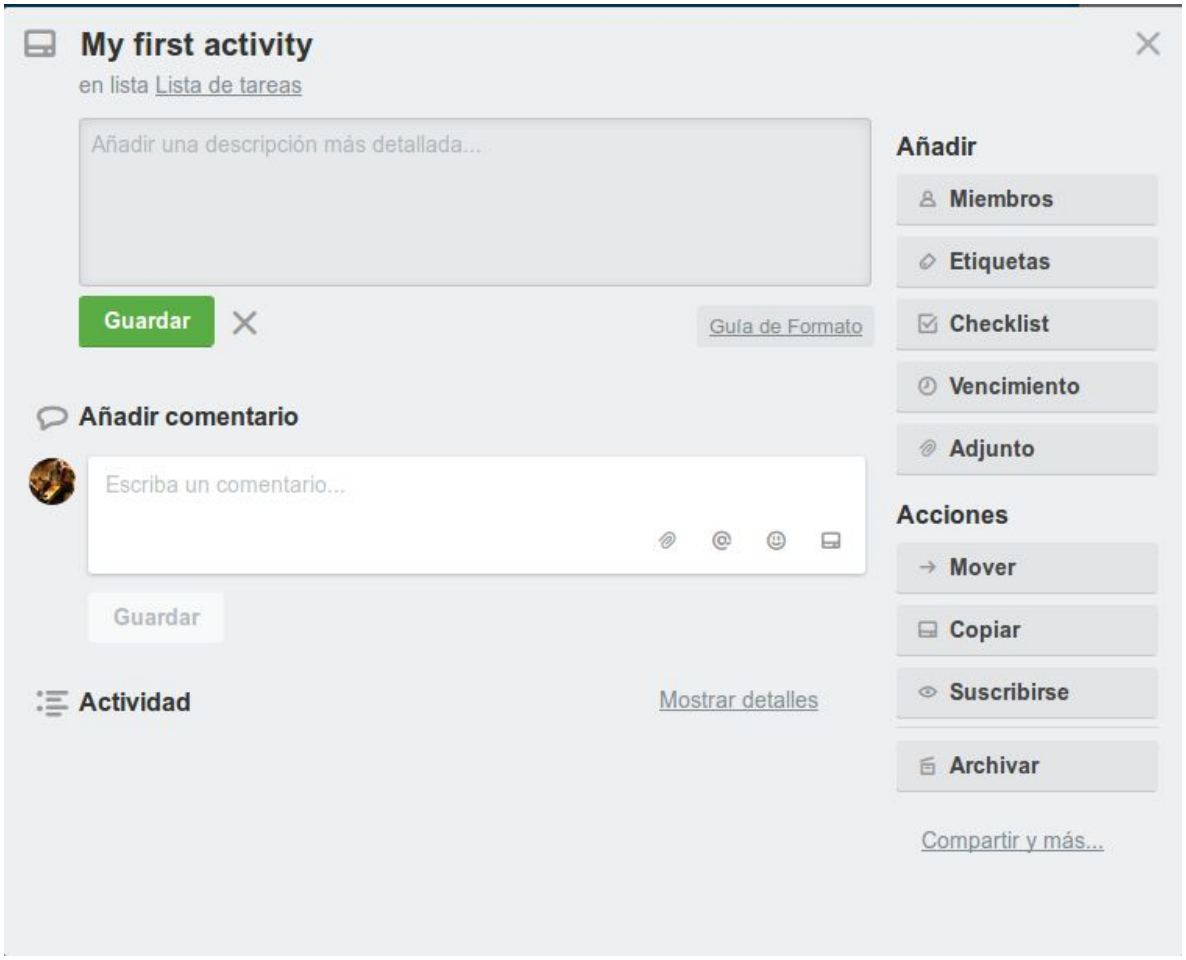
Podemos observar que se presentan tres columnas, una denominada **Lista de tareas**, en ella realizaremos la publicación de las tareas que se van a realizar durante el desarrollo del proyecto, en la columna **En proceso** se registran mediante **tarjetas** las actividades que se encuentran en proceso, las tarjetas son la unidad atómica en Trello, y la columna **Hecho** se registran las actividades que ya fueron culminadas.

Vamos a crear una nueva tarea **My first activity** y procedemos a añadirla:

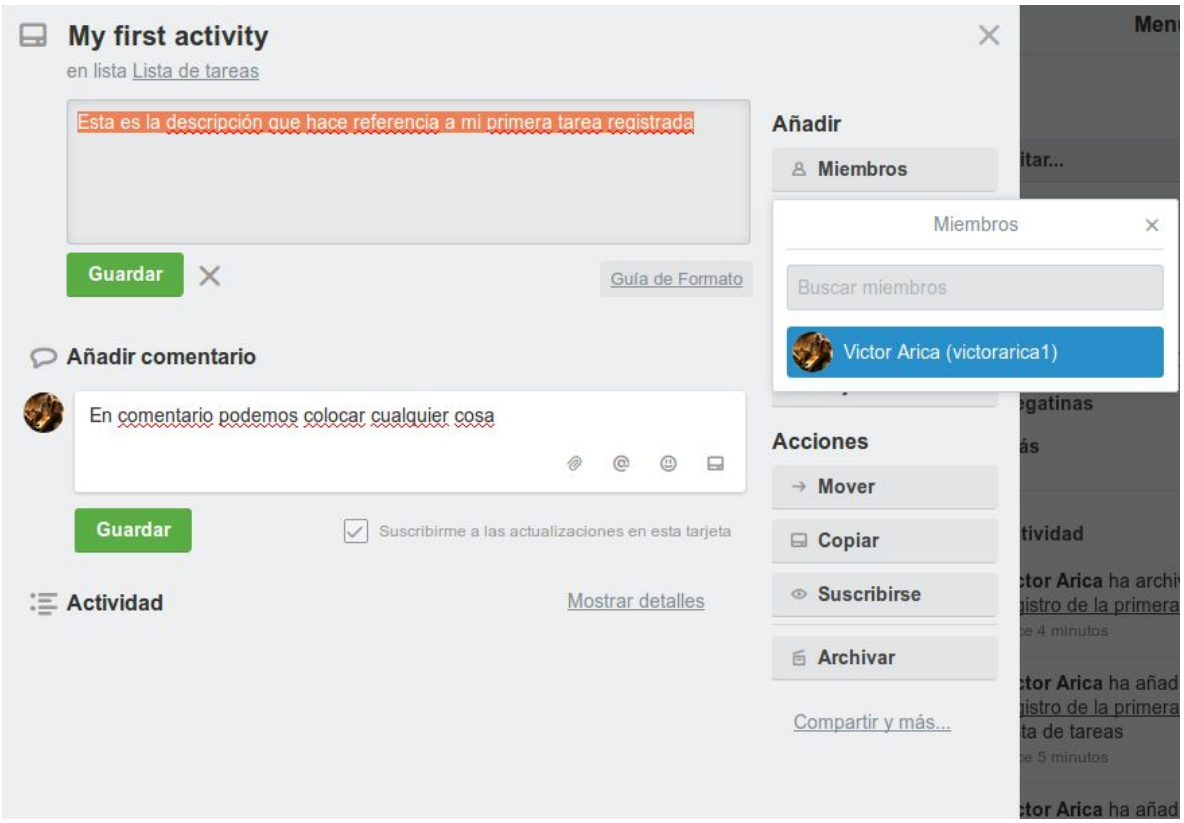




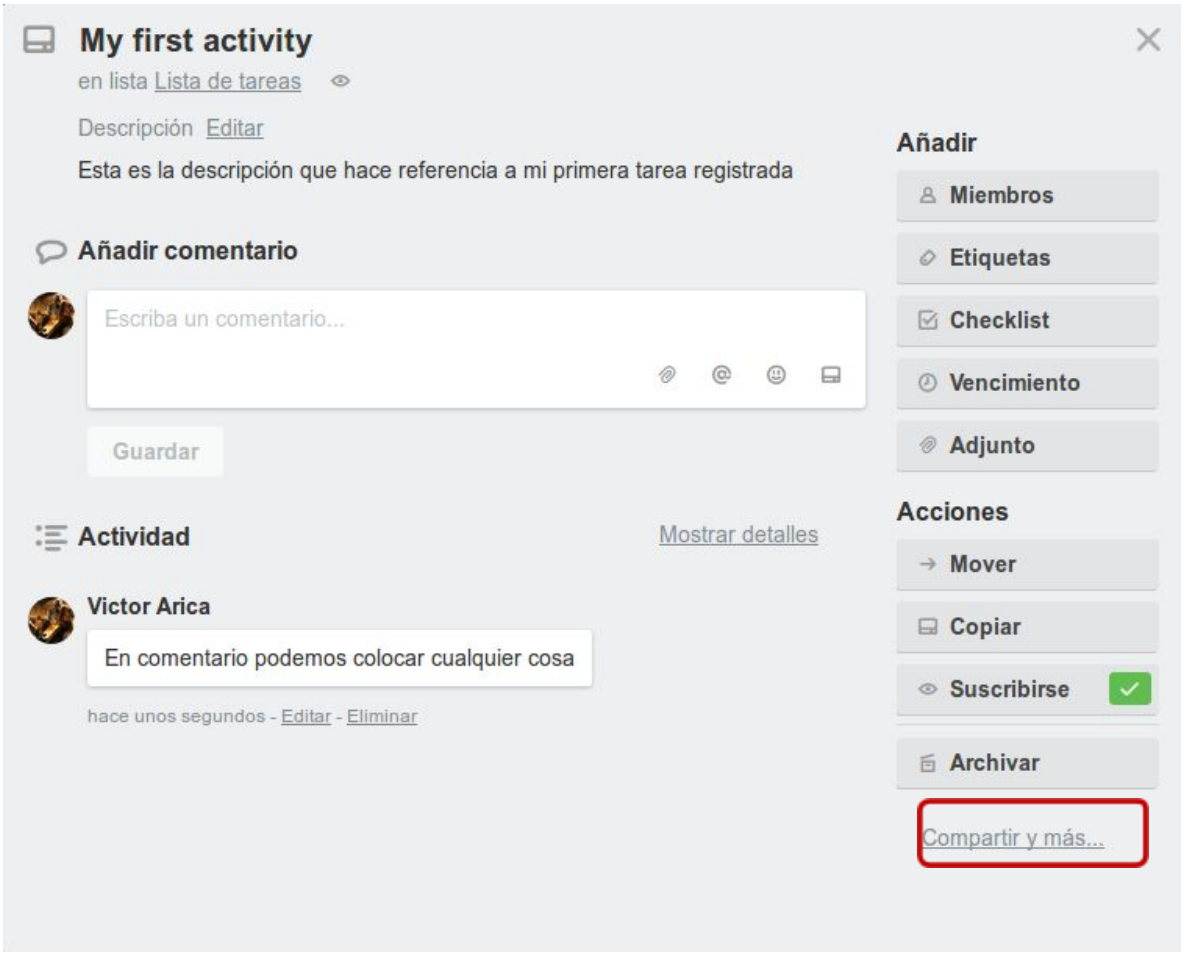
Generamos la primera tarea y la seleccionamos con un click.



En este punto accedemos directamente a la actividad, podemos añadir una descripción, colocar un comentario, añadir miembros que colaboren con el tablero, añadir etiquetas que hagan referencia a la actividad para poder realizar filtros sobre la misma, adjuntar archivos, entre muchas otras funcionalidades.

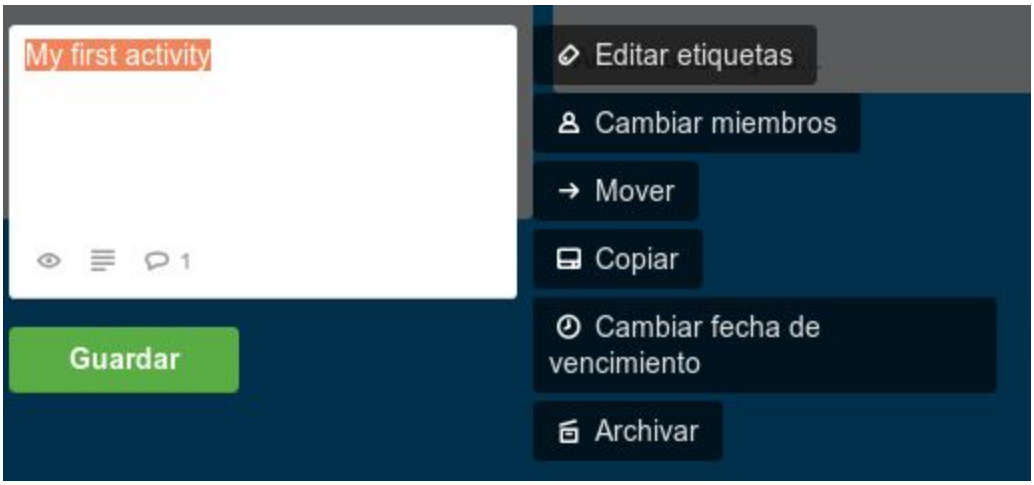


No puedo agregar miembros actualmente porque no he compartido la tarea con ningún usuario, pero básicamente funciona así, luego que guardamos todos los datos registrados veremos la siguiente interfaz:

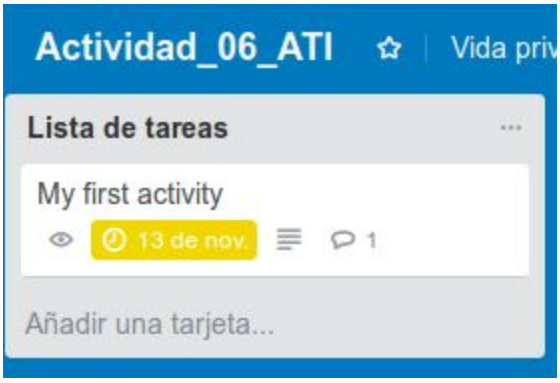


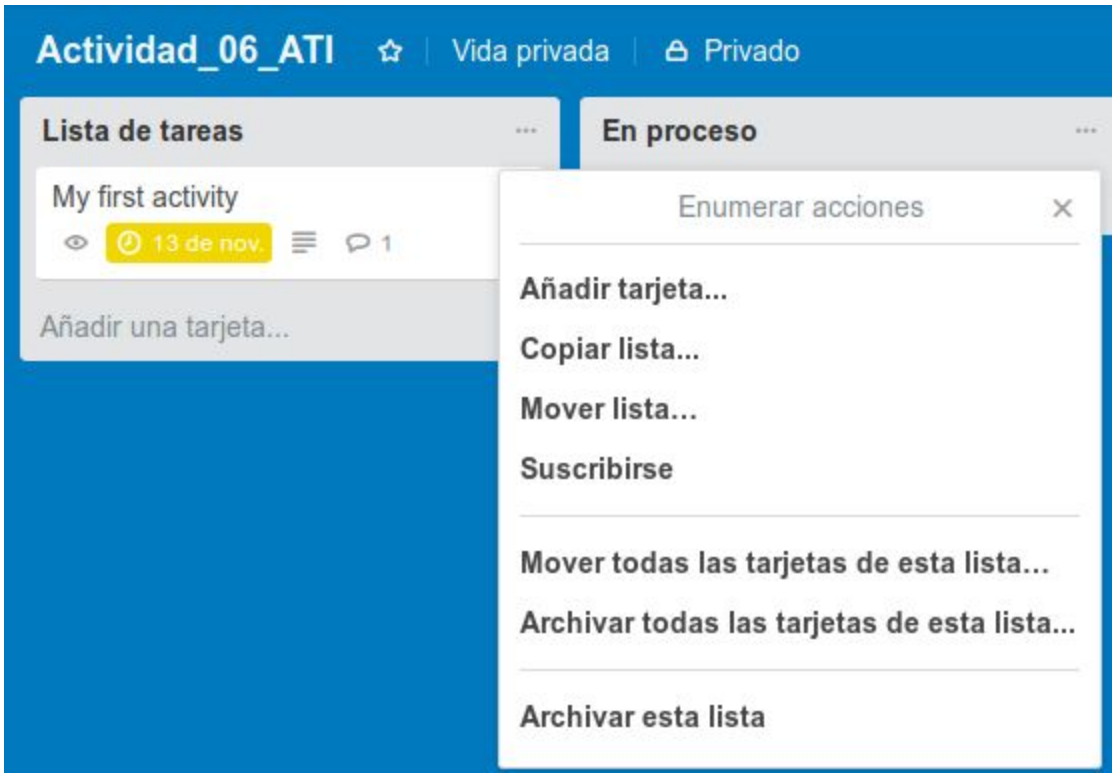
Podemos compartir la actividad mediante el enlace "**Compartir y más**".

Y se genera un enlace mediante el cual podemos compartir nuestras actividades. Además si realizamos edición de nuestra tarjeta, podemos asignar una fecha de vencimiento para la actividad.

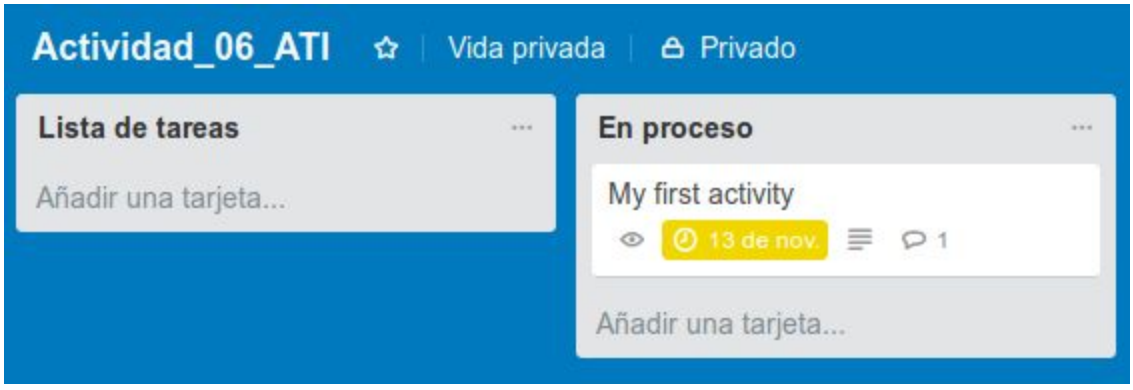



Cambiar fecha de vencimiento
Fecha
Hora
13/11/2017
3:33
Ant.
noviembre 2017
Sig.
Lu Ma Mi Ju Vi Sá Do
1 2 3 4 5
6 7 8 9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30
Guardar
Quitar
Habilite el Power-Up del calendario.
Obtendrá una vista de calendario de sus tarjetas y fuentes de iCal. ¡Guau!


Actividad_06_ATI
Vida priv
Lista de tareas
My first activity
13 de nov.
1
Añadir una tarjeta...

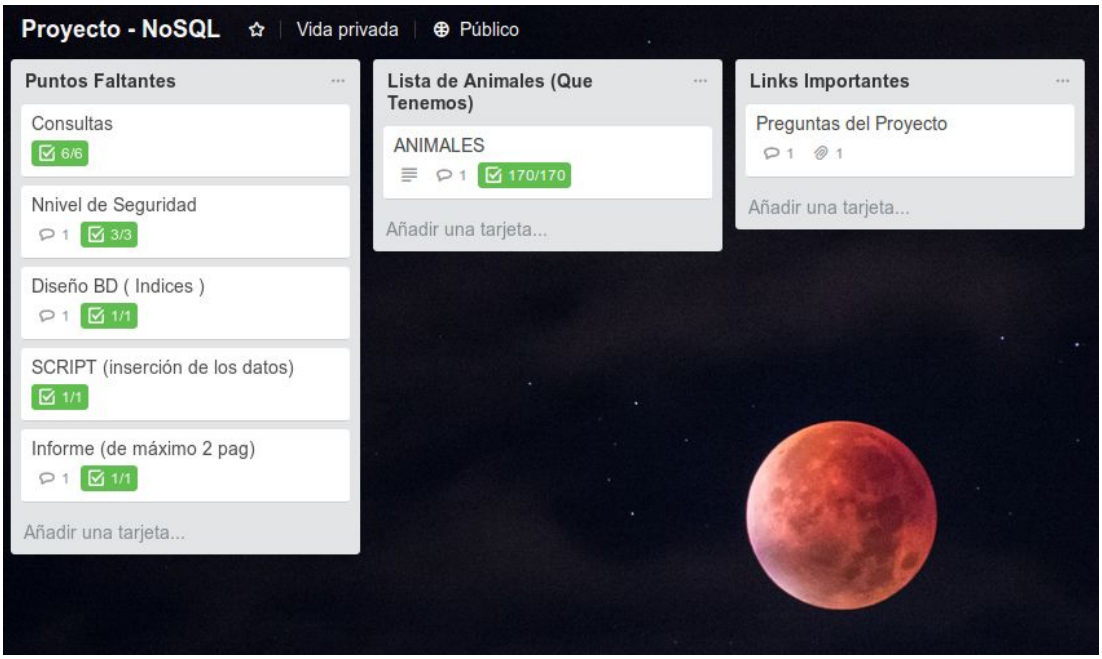


Y en el menú de opciones podemos Mover nuestras tarjetas a las columnas **En proceso** y **Hecho**.



Al moverlas a la lista **En proceso** simplemente se cambia la dirección de la actividad, y podemos realizar lo mismo, desde la lista **En proceso** hasta la lista **Hecho**. De igual modo no es necesario, puedes agregar otras listas definidas por cuenta propia sin necesidad de utilizar las que trae Trello por defecto. Y de esa forma se van planificando las actividades relacionadas con el proyecto.

Un ejemplo del uso del tablero de Trello para realizar la planificación de un proyecto puede ser:



El cual lo realicé durante el desarrollo del Proyecto de NoSQL, se pueden apreciar unos iconos con color verde, son opciones de checklist que también pueden ser incorporados durante la edición de la actividad y permiten llevar un seguimiento del grado de avance o culminación de la actividad.

4. Git

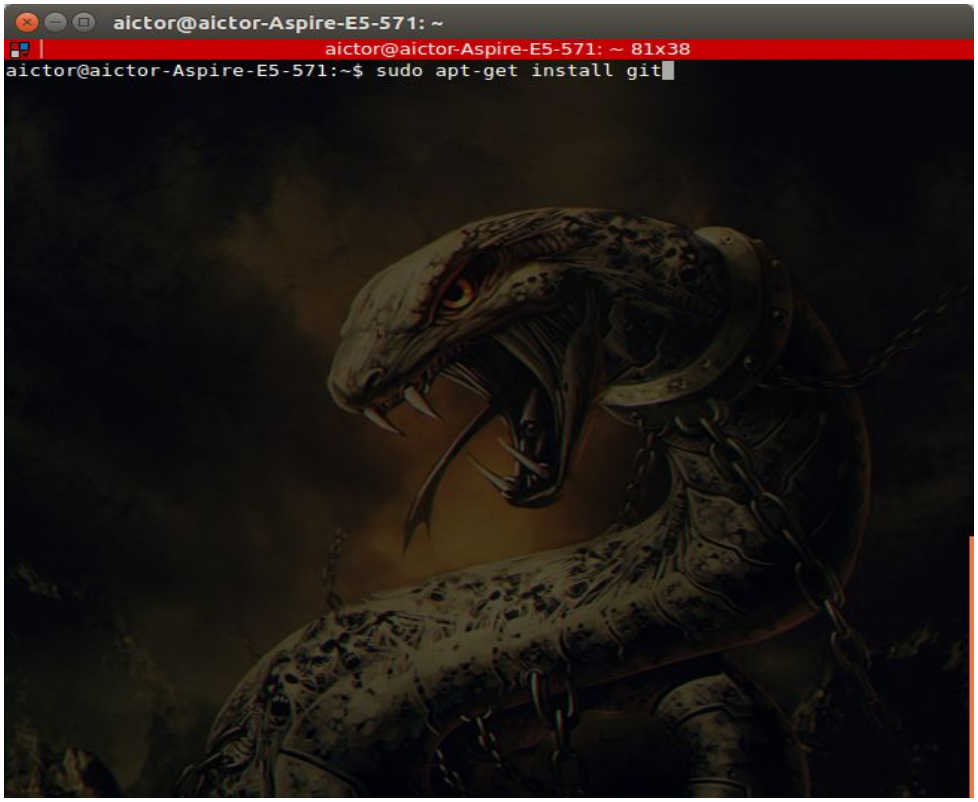
De acuerdo a su página oficial, Git es un sistema distribuido de control de versiones libre y de código abierto, diseñado para gestionar desde pequeños hasta grandes proyectos con rapidez y eficiencia.

Git es fácil para aprender, tiene un rendimiento muy rápido y presenta las siguientes características:

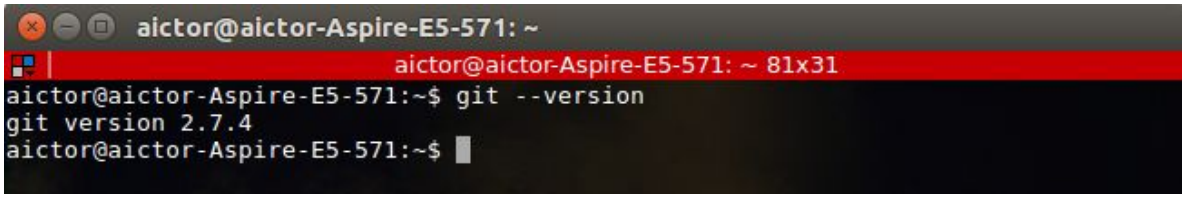
- Branching & Merging
- Pequeño y rápido
- Distribuido
- Aseguramiento de datos
- Staging area
- Libre y código abierto

A continuación el preparador explicará cada una de estas características presentadas anteriormente.

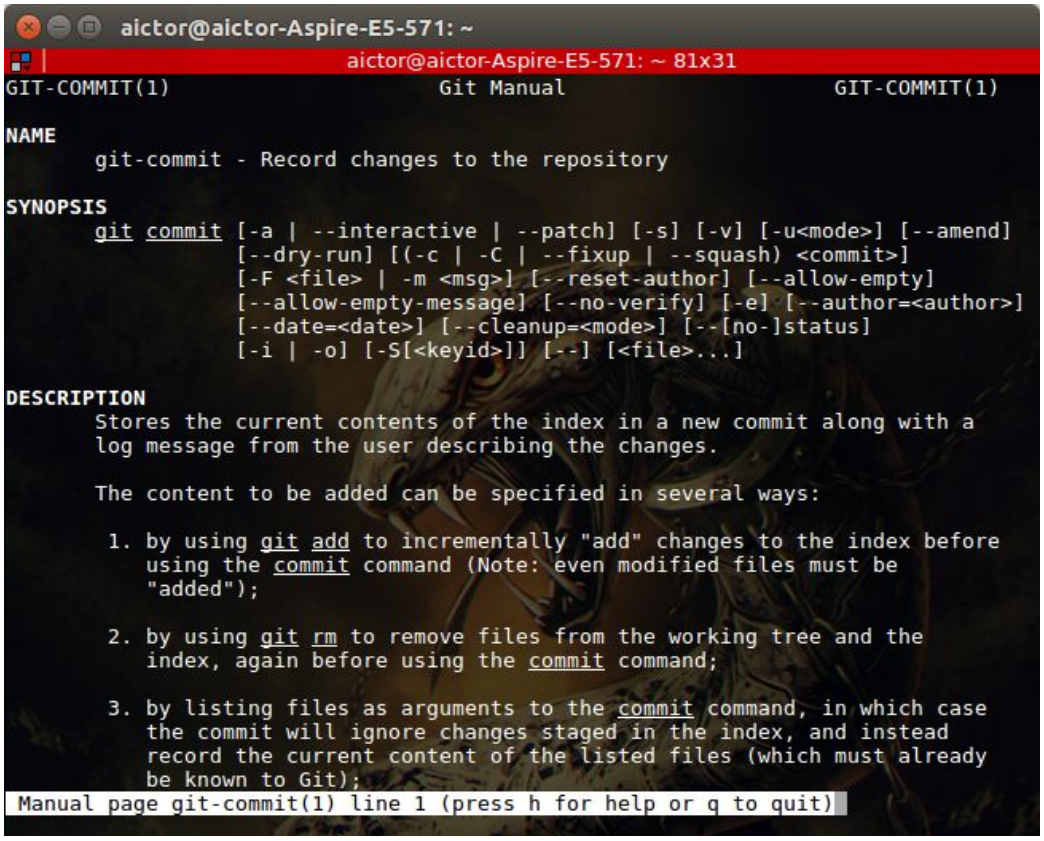
1. **Instalación de Git:**
   a. **Para comenzar con la instalación, debemos acceder a una terminal y colocar el comando "sudo apt-get install git"**



   b.
   c. **Verificamos la instalación obteniendo la versión de git que fue instalada: "git - -version"**

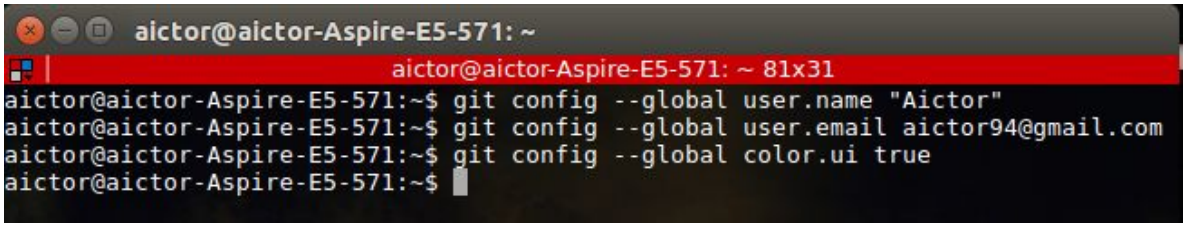


d. Si queremos acceder al manual de Git, podemos utilizar el comando "git help <instrucción>"

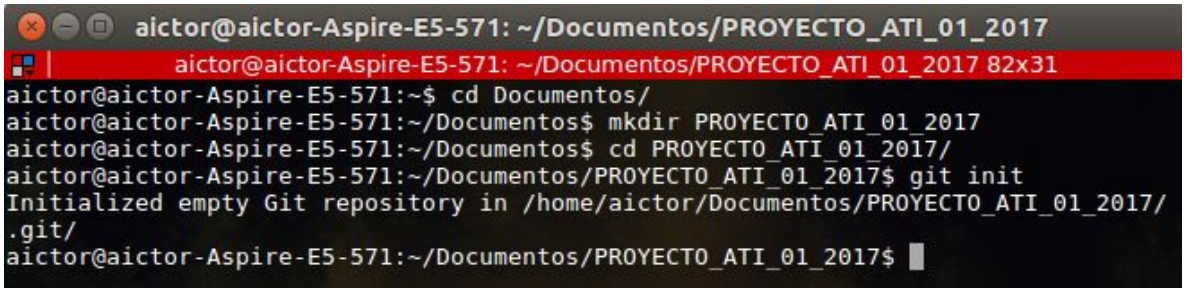


e. Configuración de Git

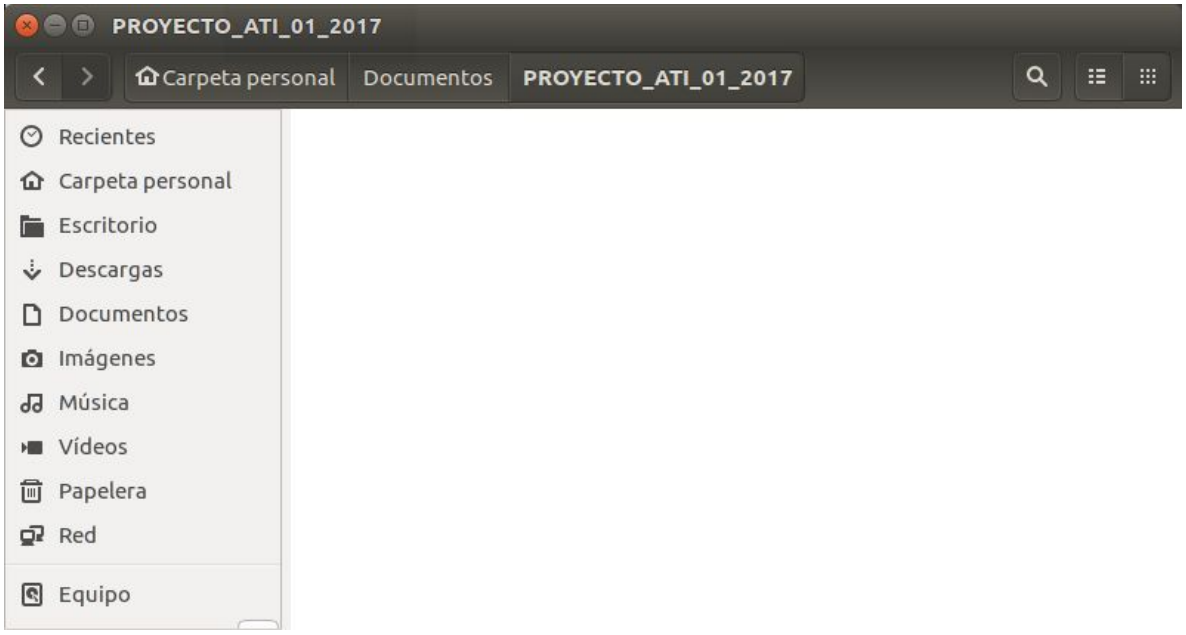
Existe una configuración global que puede ser asignada a través de los siguientes comandos:



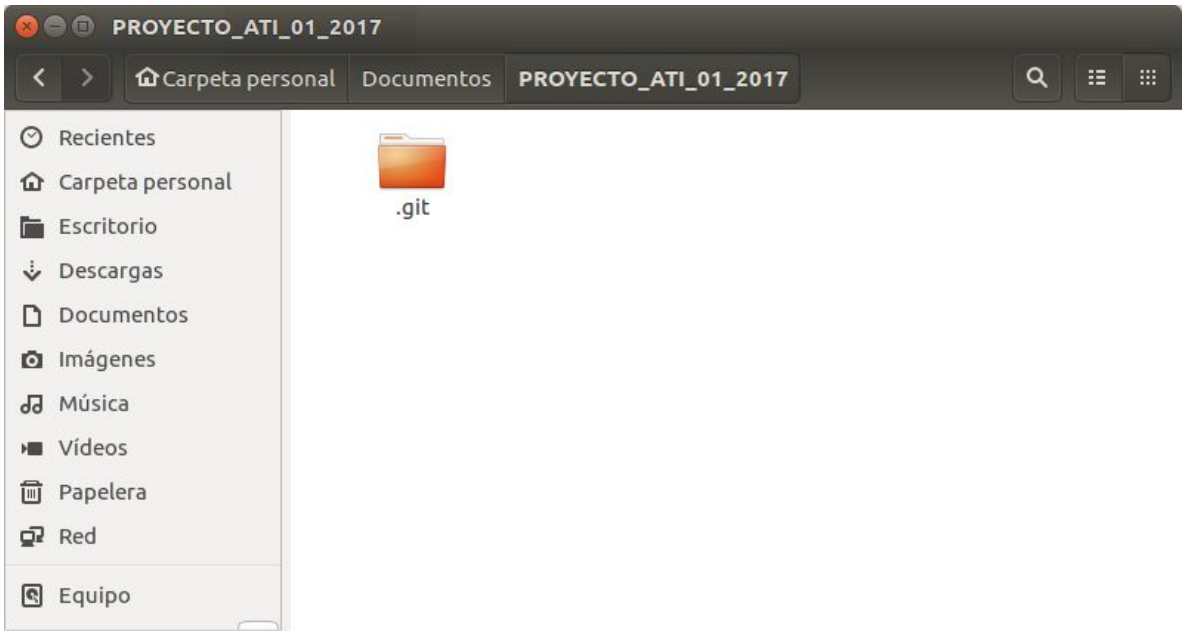
f. Ahora que se ha realizado la configuración de Git, podemos comenzar a crear nuestro primer directorio de trabajo:



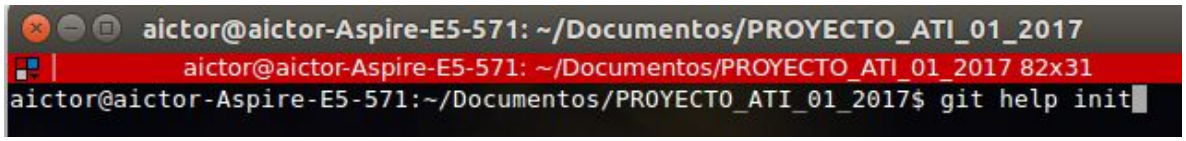
Básicamente accedemos a un directorio de nuestra preferencia, luego con el comando **mkdir** creamos un directorio nuevo llamado "**PROYECTO_ATI_01_2017**", accedemos a ese directorio nuevo creado, y una vez posicionado en dicho directorio procedemos a ejecutar el comando **git init**, se supone que debe crearse un directorio **.git**, abrimos el sistema de archivos para comprobar esto.

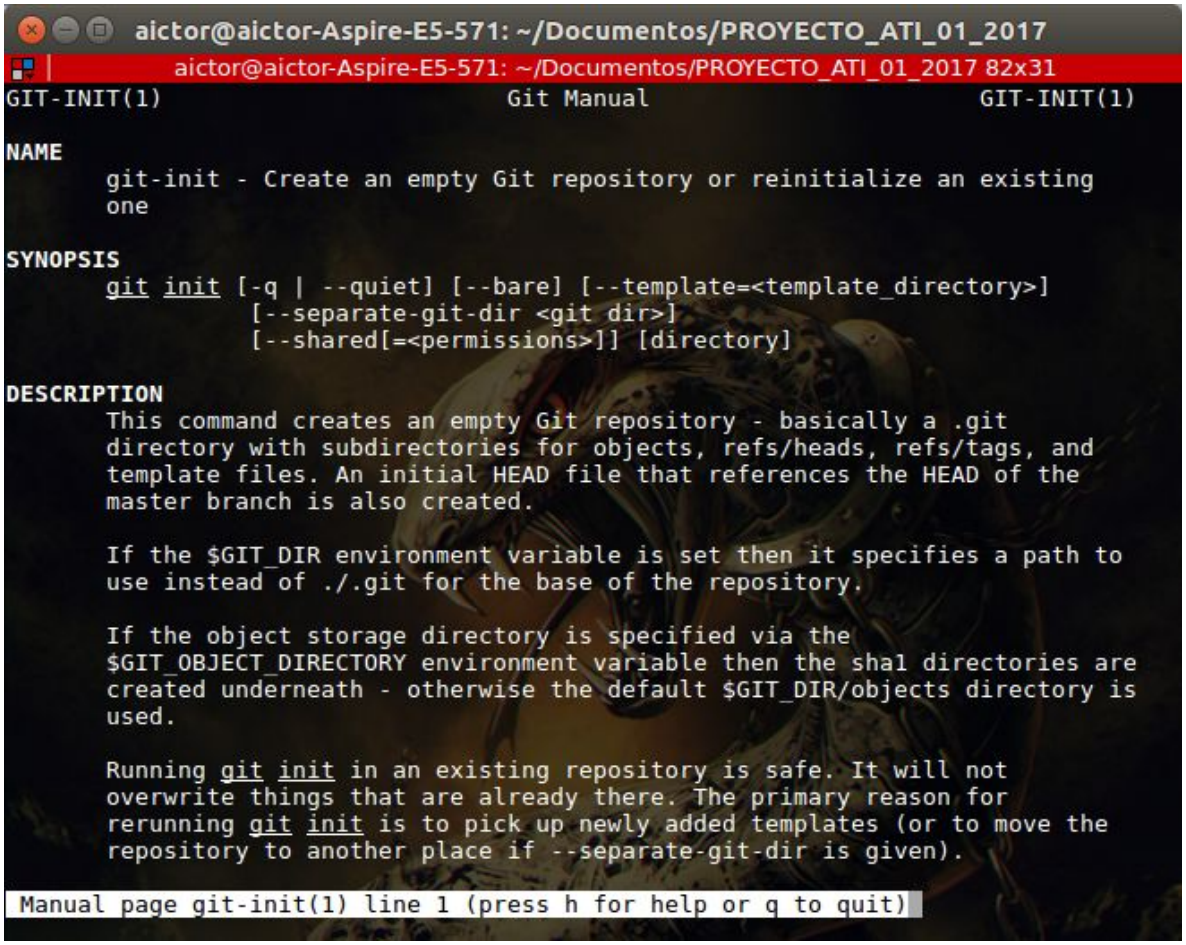


Sin embargo, el directorio aparentemente se encuentra vacío, esto se debe a que dicho directorio se encuentra oculto, para mostrar los archivos ocultos en el sistema presionamos las teclas **Ctrl + h** y revisamos de nuevo.



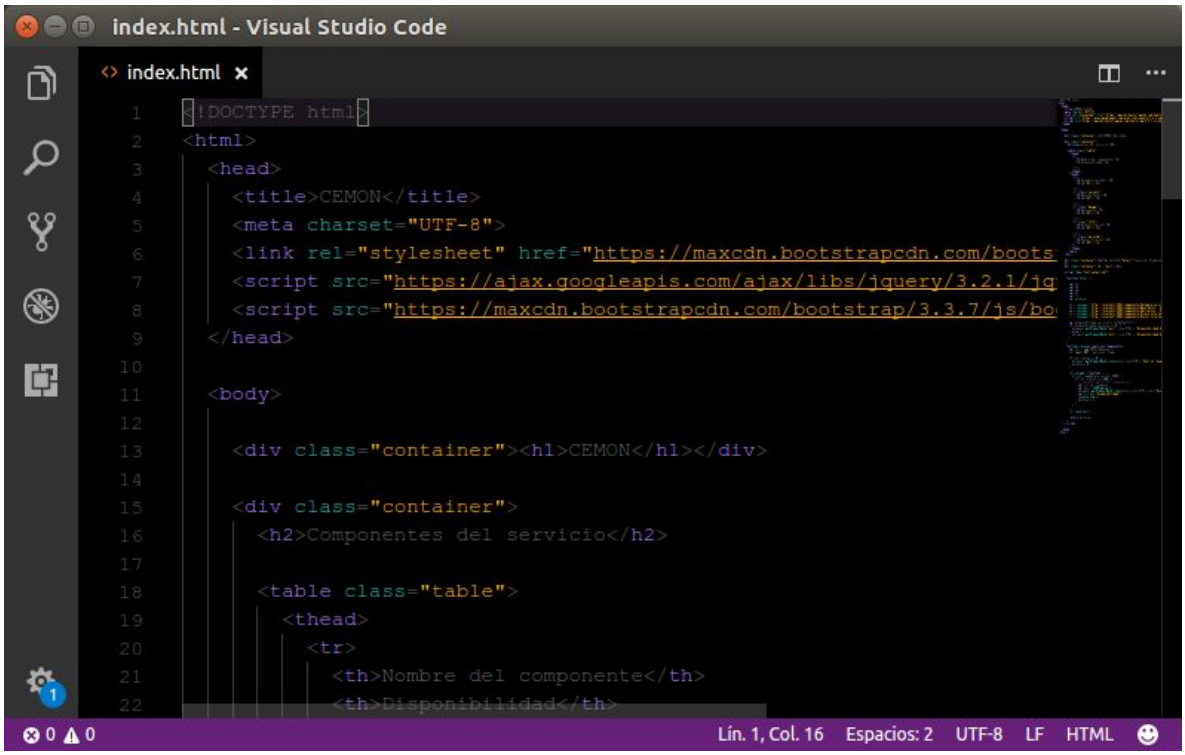
Pero realmente, ¿qué es este directorio? Lo podemos averiguar mediante el uso de la terminal.



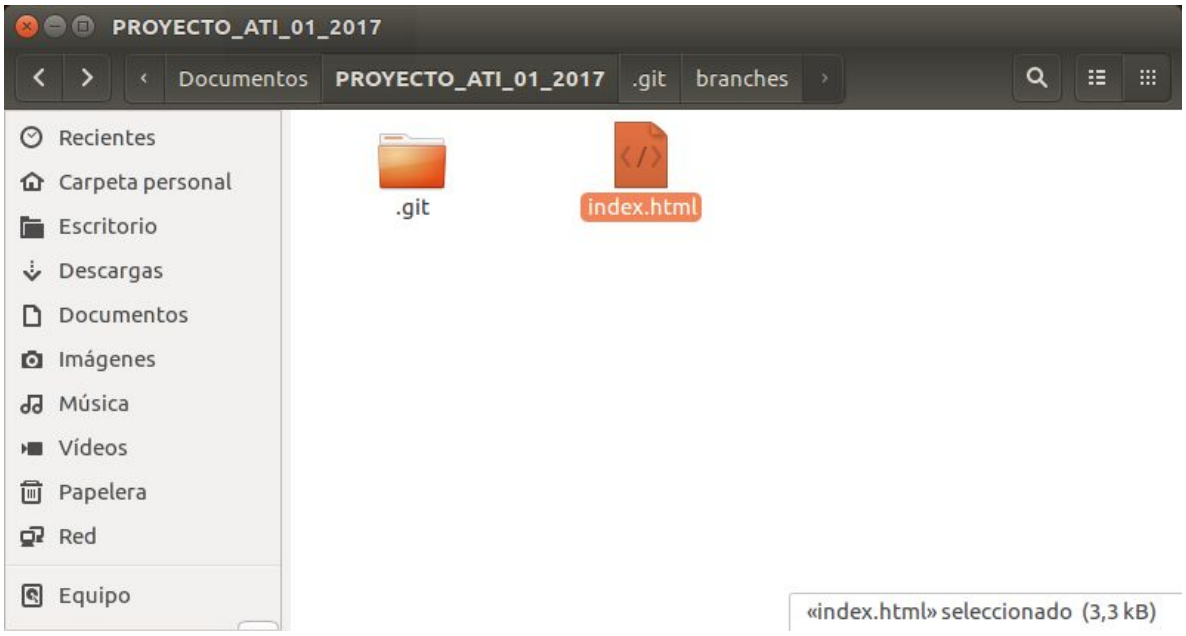


De acuerdo al manual de Git, el comando realizado anteriormente "**git init**" crea un repositorio Git vacío o reinicializa uno que se encuentre existente, y se crean los archivos y subdirectorios adicionales. Algo importante a considerar es que el archivo HEAD hace referencia al HEAD de la rama master, que una vez que es realizado el comando, dicha rama se crea por defecto.

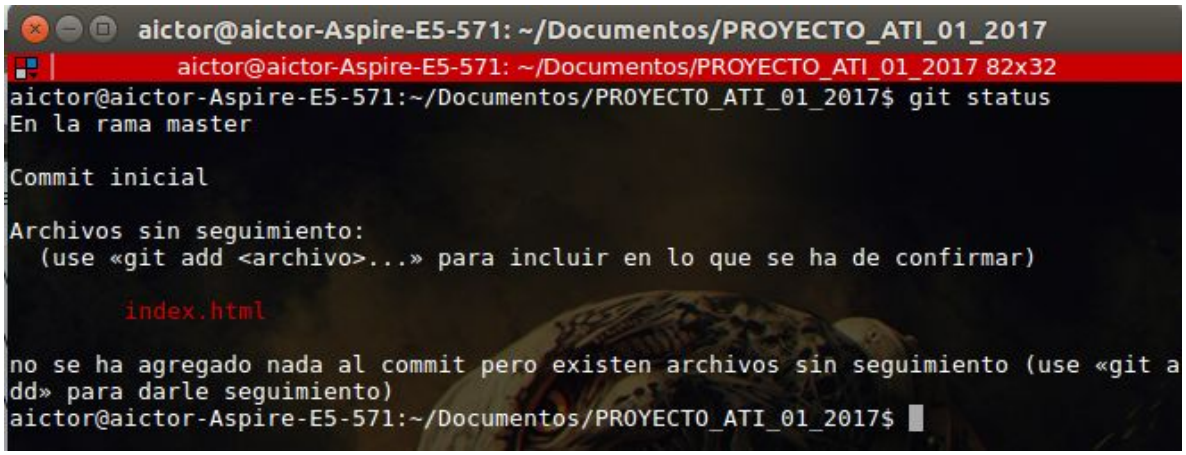
g. Agregando un archivo nuevo a nuestro directorio de trabajo



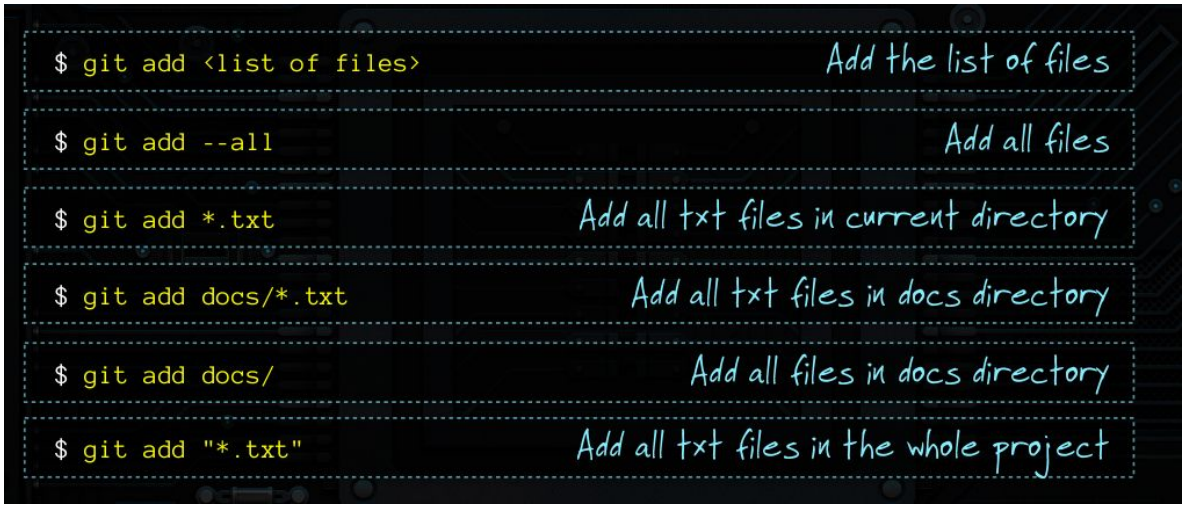
Asumiendo que este es un archivo que se quiere agregar al directorio, simplemente se guarda en el directorio, y nos aseguramos que se guarde correctamente.



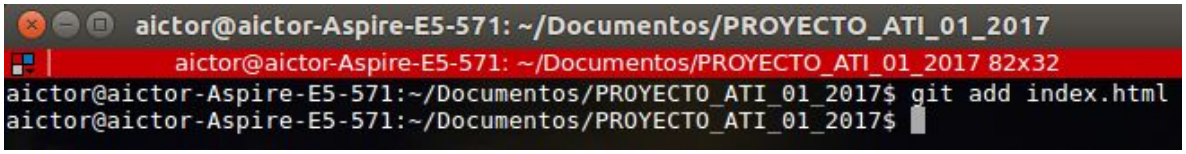
En este punto es importante señalar que existe un comando que nos permite obtener el estado en el cual se encuentra nuestro directorio de trabajo, dicho comando es "**git status**":



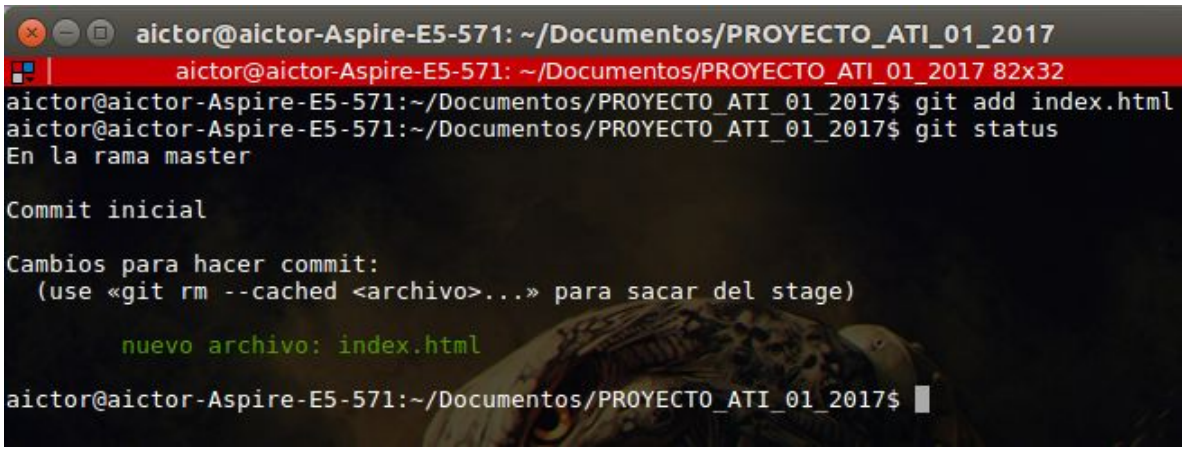
Al utilizarlo nos indica que ha sido agregado el archivo "**index**.html", y el mismo se encuentra en la rama "master" (por defecto), sin embargo, no es suficiente, si queremos subir o agregar este archivo a nuestro repositorio, debemos realizar el comando "**git add**", el cual tiene una gran cantidad de opciones:



En nuestro caso, basta con realizar "**git add index.html**".



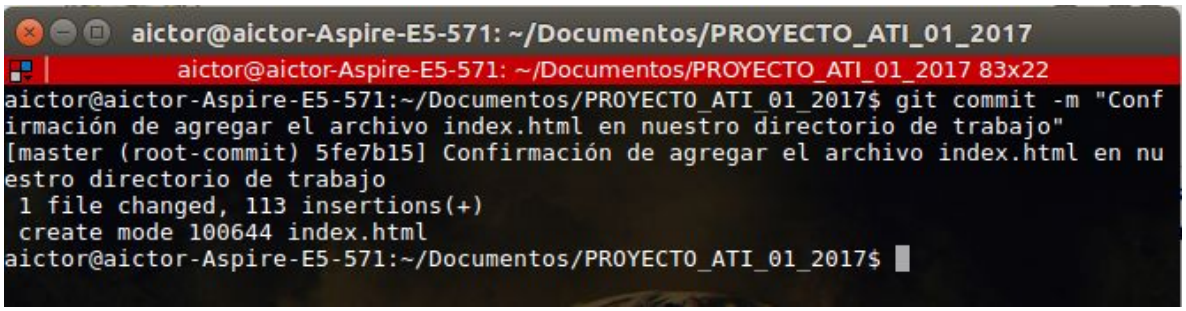
Este comando permite que se agregue el archivo a nuestro directorio local y si obtenemos el status:



Nos indica que aún queda pendiente realizar commit del archivo anteriormente agregado.

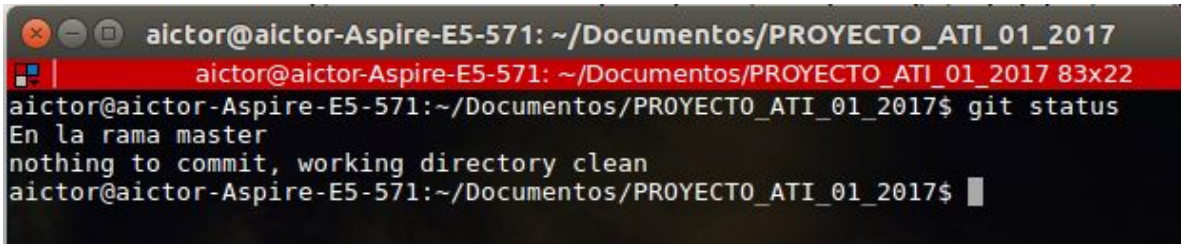
h. Commit en git

Esta instrucción nos permite confirmar de manera absoluta la inserción del archivo o los conjuntos de archivos que se encuentran en seguimiento, para proceder a pasarlos del área de Staging a registrarlos de manera local.



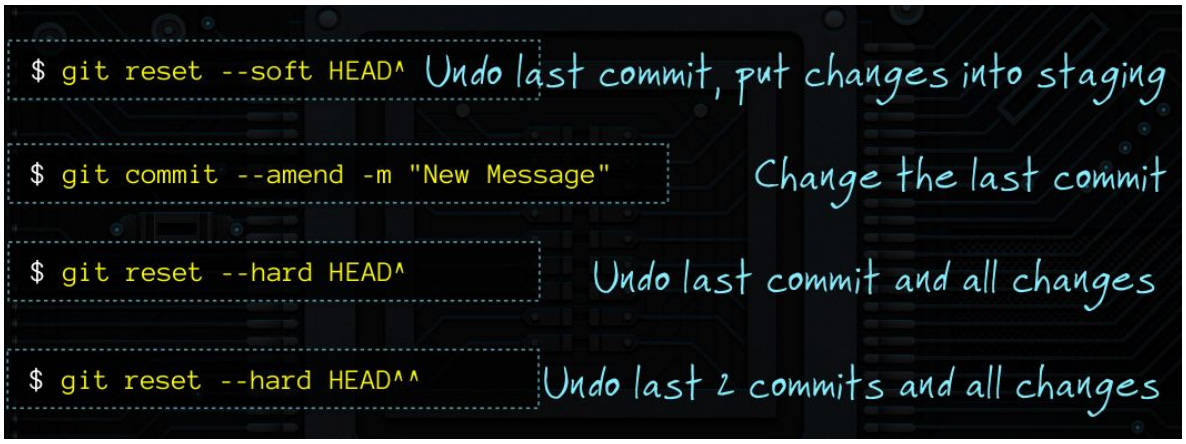
La sintáxis es "**git commit –m <mensaje>**", cabe destacar que una vez que se realice commit, valga la redundancia, git hará commit de todos los archivos que se encuentran en el área de Staging y que fueron agregados mediante la instrucción "**git add …** ", si queremos ignorar un archivo que se encuentra en el área de Staging, podemos eliminarlo de la misma utilizando el comando "**git reset <nombre_de_archivo>**", y "**–m**" indica el mensaje que aparecerá una vez que se registre el commit.

i. Realizando "**git status**"



Nos indica que no hay cambios que requieran realizar commit's, y que el directorio de trabajo se encuentra limpio.

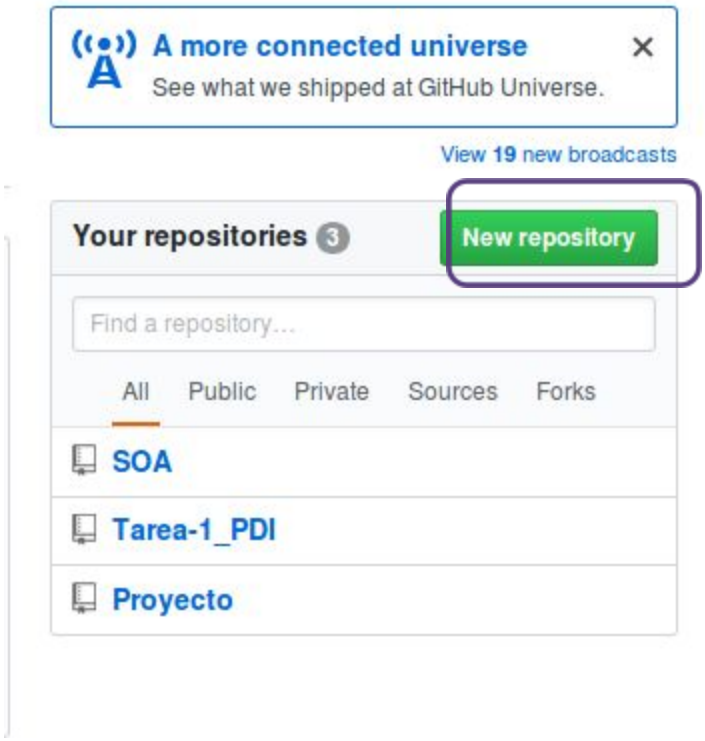
j. Comandos adicionales para deshacer o modificar commit's:



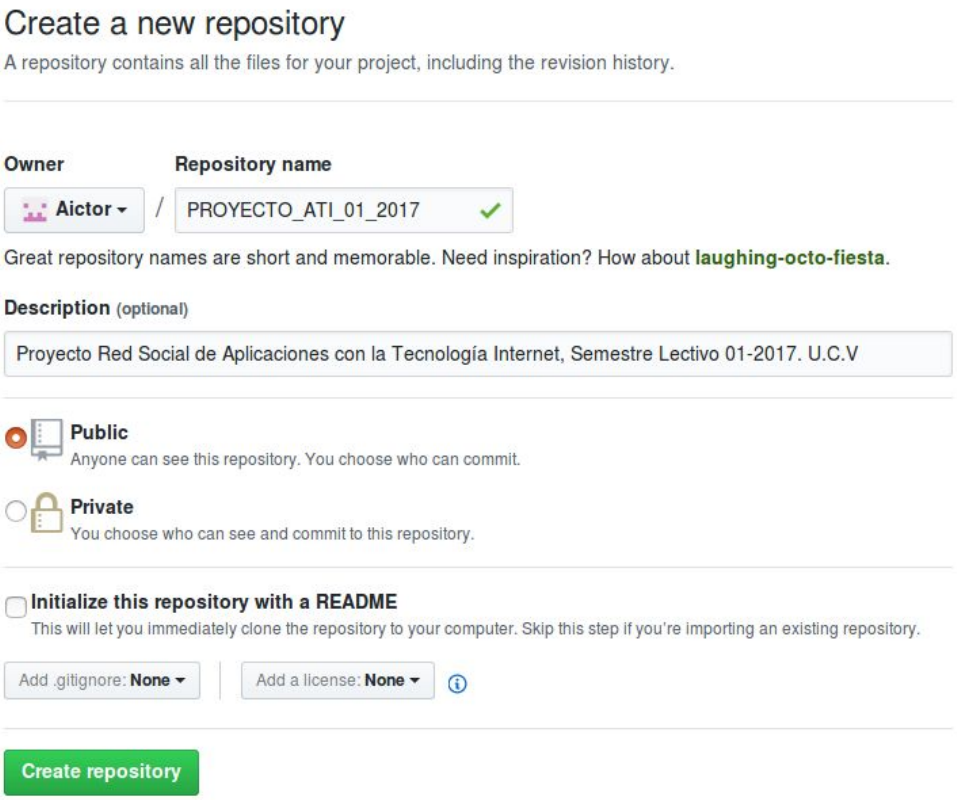
k. Repositorios remotos

Sabemos que todo lo que hemos realizado anteriormente ha sido almacenado de manera local, ahora bien, si queremos guardar nuestro proyecto y utilizar el sistema de gestión de versiones desde un repositorio remoto sería mucho más factible para nosotros y de esta forma aumentamos la seguridad en torno al riesgo de perder nuestro proyecto por causa de una falla local en nuestra máquina.

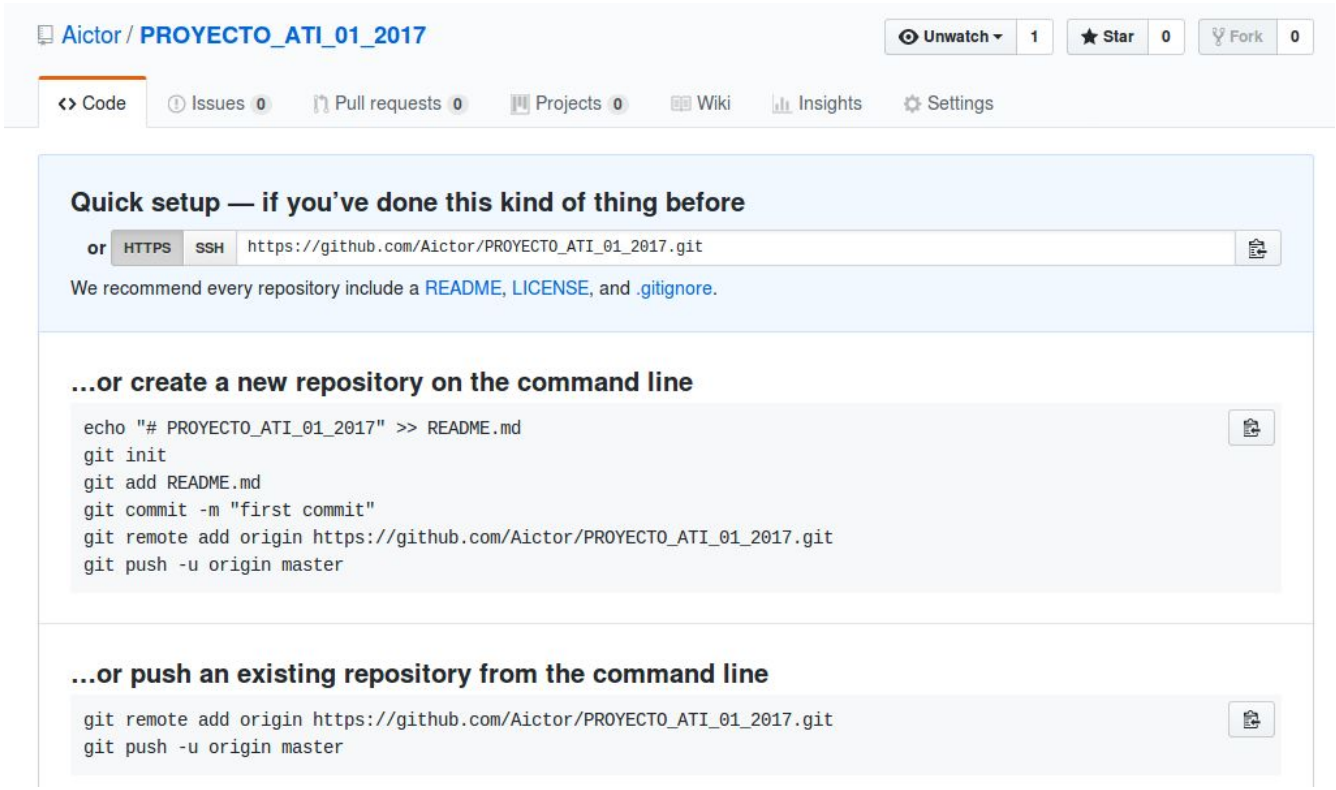
Pero ¿cómo podría utilizar un repositorio remoto? Es sencillo, lo primero que debemos hacer es crearnos una cuenta en https://github.com, luego accedemos a la cuenta creada y sólo nos queda crear un repositorio nuevo.



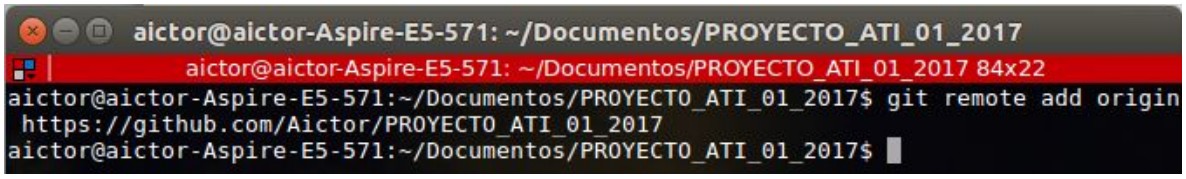
Este será el repositorio remoto donde almacenaremos el proyecto.



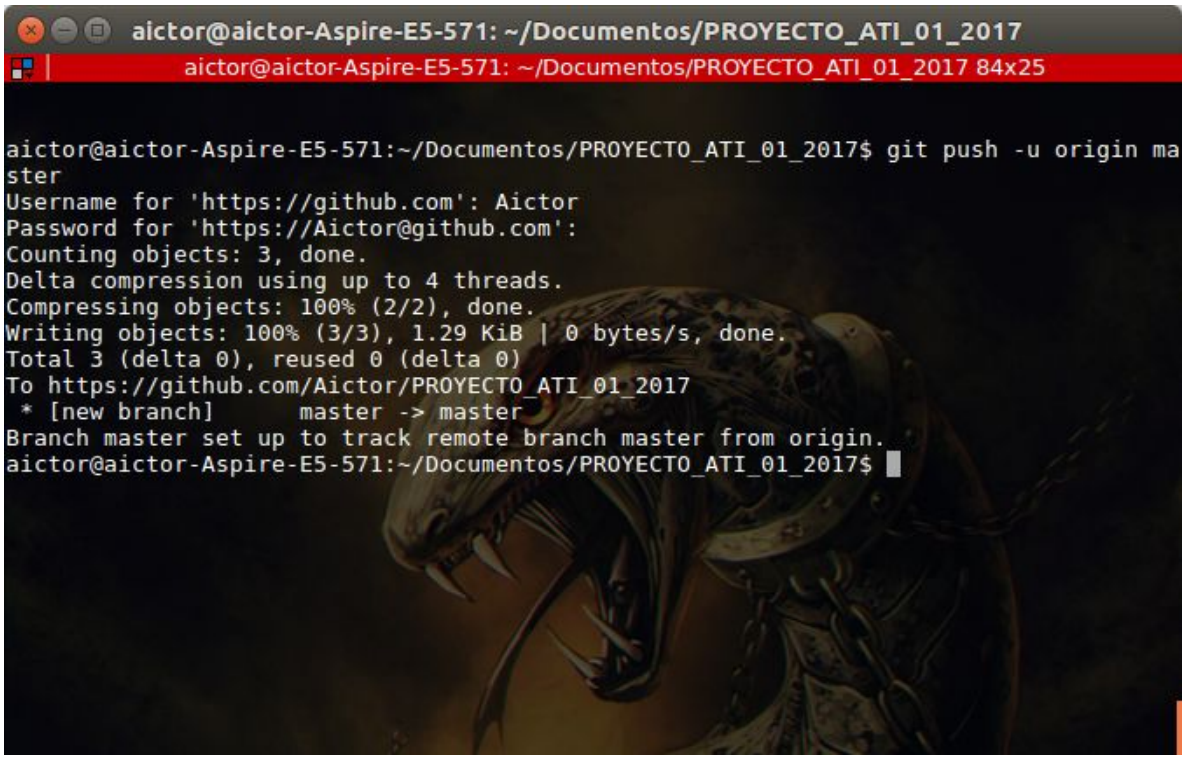
Al seleccionar la opción de "**New repository**" se desplegará esta interfaz, en la cual ingresaremos el nombre de nuestro repositorio, una breve descripción, si es público o privado (en caso de ser privado se debe pagar) y si deseamos que se inicialice con un archivo README, seguidamente procedemos a crear el repositorio.



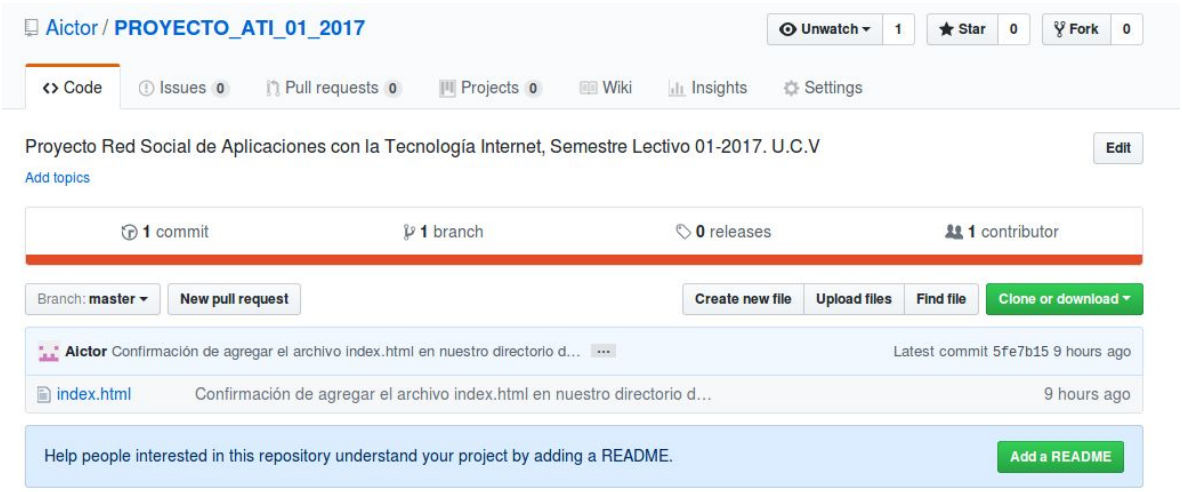
Se despliega una interfaz indicando el nombre del repositorio, la dirección que le corresponde, y un conjunto de instrucciones en caso de querer crear un nuevo repositorio en este repositorio remoto o simplemente agregar un repositorio ya existente en este mismo repositorio remoto. Anteriormente creamos un directorio de trabajo local que contiene un archivo "**index.html**", nuestra próxima tarea será agregar este directorio de trabajo sobre el repositorio remoto creado anteriormente, para ello utilizaremos la segunda opción, no es necesario realizar "**git add …** " ni "**git commit …** " porque asumimos que lo último que realizamos fue un commit, y el directorio de trabajo se encuentra limpio, es decir, sólo nos queda realizar el comando "**git push –u origin master**", **origin** hace referencia al repositorio remoto que en nuestro caso será el agregado anteriormente, mientras que **master** es la rama principal donde estamos realizando los cambios por defecto, sin embargo, como origin no se encuentra definido, debemos hacerlo mediante "**git remote add origin** https://github.com/Aictor/PROYECTO_ATI_01_2017.git".



Listo, ahora que agregamos el repositorio remoto sólo nos queda subir nuestro directorio de trabajo hacia ese repositorio remoto, eso lo lograremos con el comando "**git push –u origin master**".

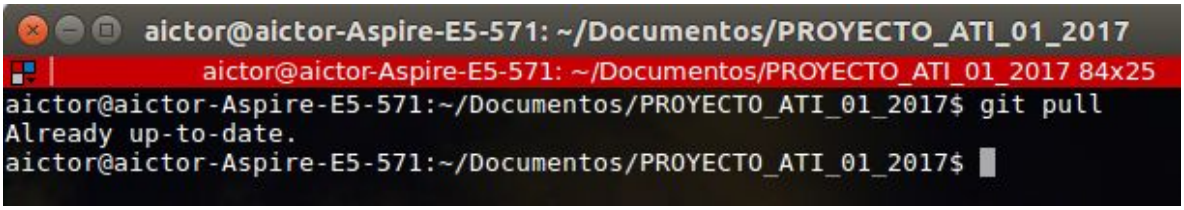


Ahora vamos a la interfaz web y recargamos la página:



Y podemos observar que el directorio es agregado satisfactoriamente, además desde la página podemos acceder al historial de commit's, seleccionar el branch que está siendo utilizado, entre otras funcionalidades.

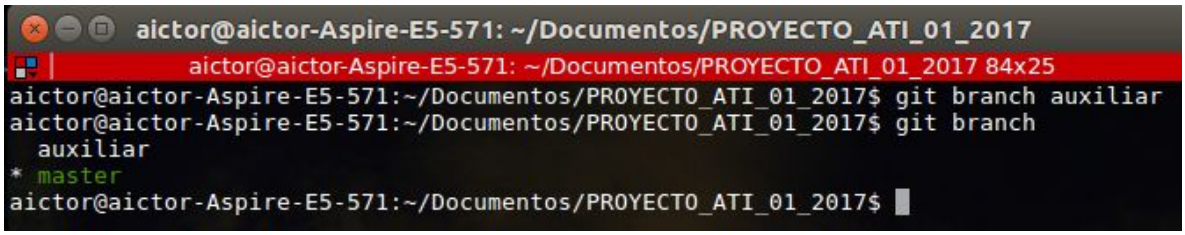
En caso de querer obtener los últimos cambios realizados sobre el repositorio remoto realizamos el comando "**git pull**"



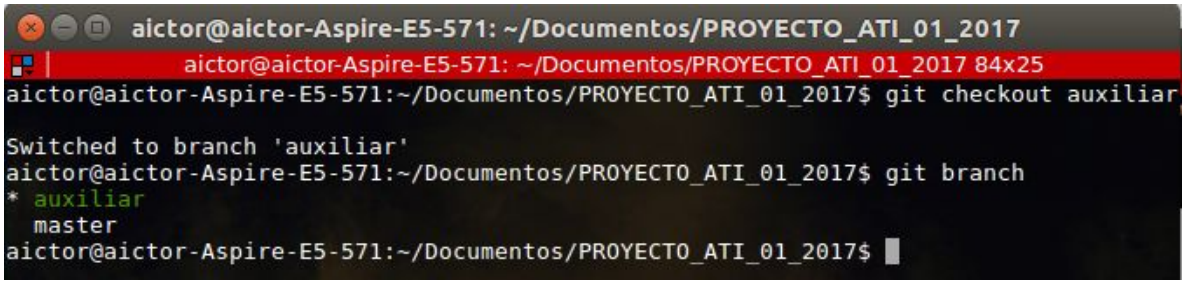
No ocurre ningún cambio porque no hemos realizado modificaciones sobre el repositorio remoto, en caso de hacer modificaciones se traerá las mismas y las generará sobre la copia que se encuentra local.

I. Branch

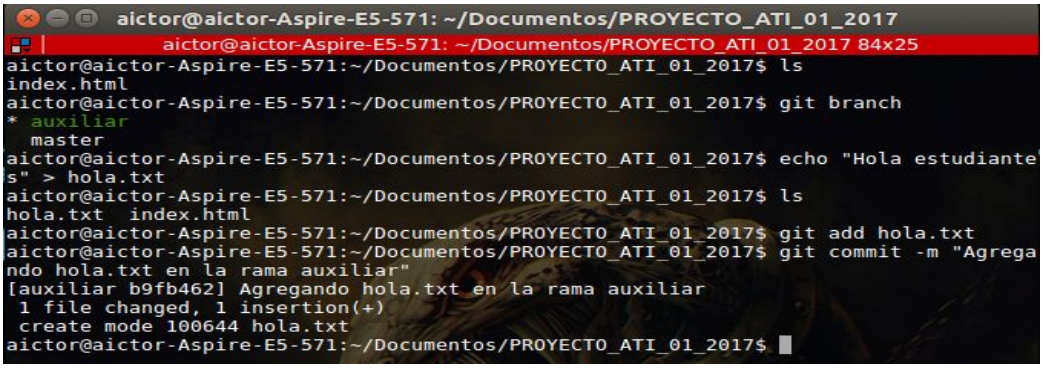
Sabemos que la rama principal por defecto es "**master**", sin embargo, si queremos utilizar una rama auxiliar para no modificar la rama principal del proyecto, podemos crear un nuevo **branch**, para este ejemplo crearemos un nuevo **branch** denominado **auxiliar**.



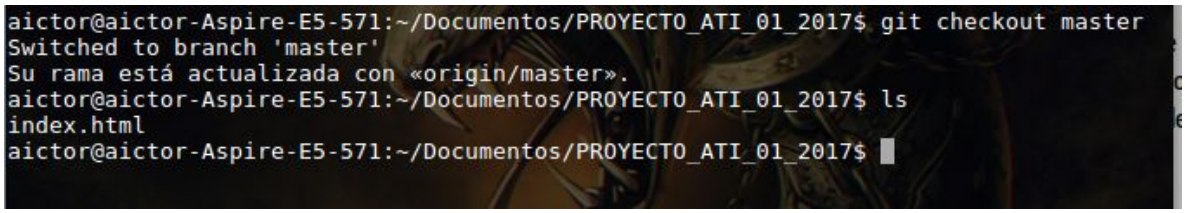
No obstante, podemos observar al utilizar el comando "**git branch**" que aún nos encontramos en la rama **master** y que la rama auxiliar si fue creada mediante el uso del comando anterior "**git branch auxiliar"**, para desplazarnos a la rama creada anteriormente necesitamos hacer uso del comando "**git checkout auxiliar**".



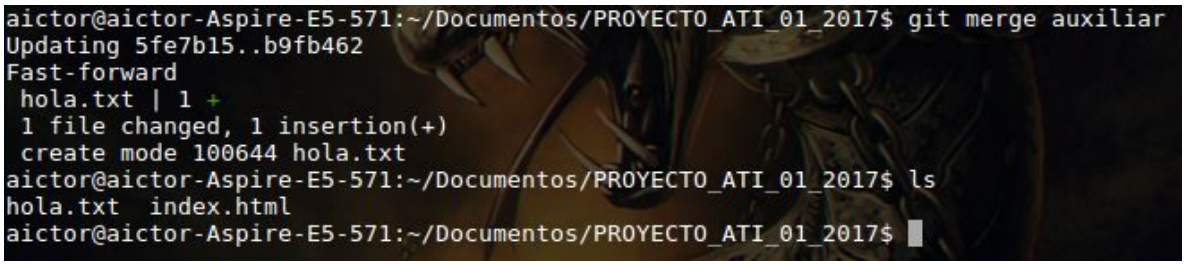
Ahora si nos encontramos en la rama **auxiliar**, cabe resaltar que una vez que nos encontremos en una rama, los cambios realizados, archivos eliminados y agregados sólo se mantendrán en la rama seleccionada, en caso de cambiar de rama, dichos cambios no serán visibles por las otras ramas. Si queremos unir lo que tenemos en la rama **auxiliar** con la rama principal **master** necesitamos realizar un **merge** que consiste en la unión de dos o más componentes presentes en el proyecto. Gráficamente lo podemos ver de la siguiente manera:



Una vez encontrados en la rama **auxiliar,** creamos un nuevo archivo de nombre **hola.txt**, si hacemos **ls** podemos ver que el archivo se encuentra junto con el **index.html**, procedemos a realizar "**git add …**" para agregarlo, luego **commit** y por último nos cambiamos de branch, al **master:**



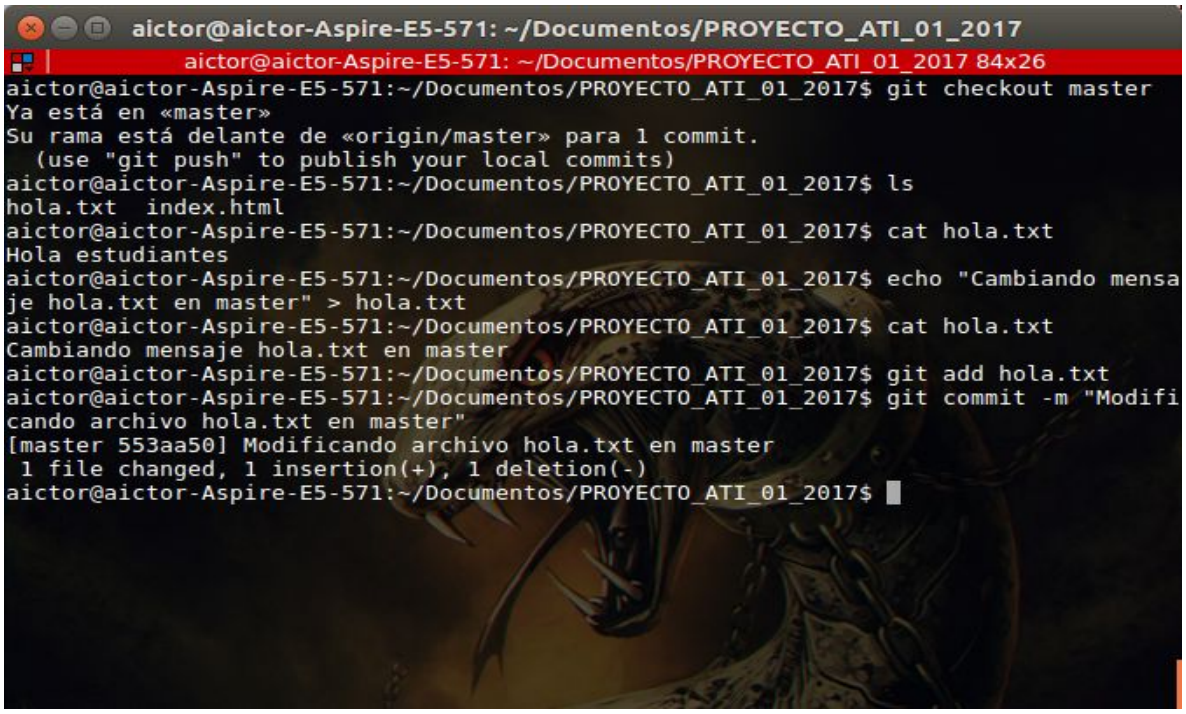
Se observa que en **master** no se aprecian los cambios realizados en la rama **auxiliar**, para poder obtenerlos, debemos aplicar un **merge** desde la rama de **master**:



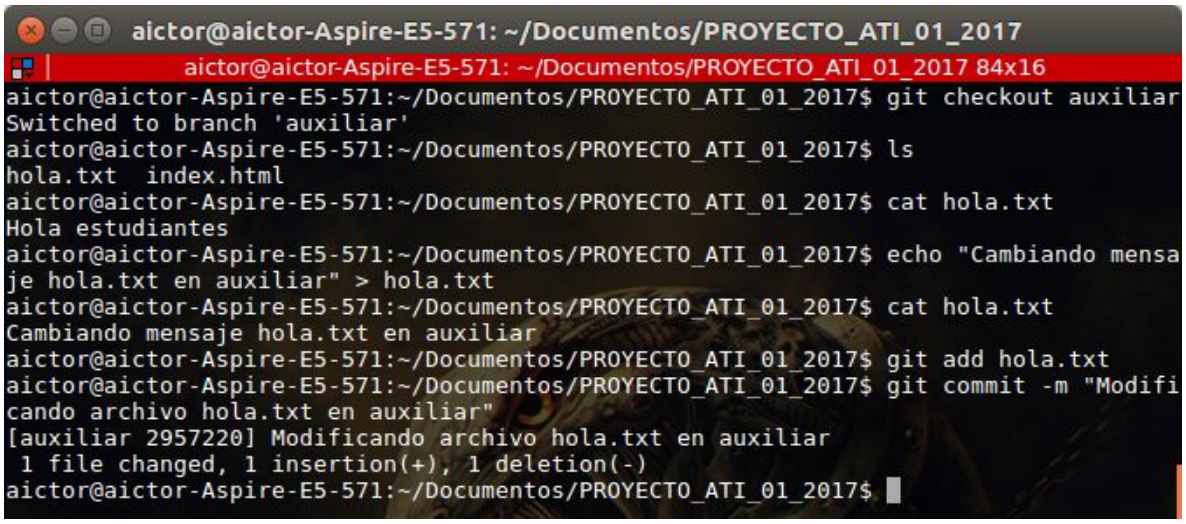
m. Conflictos

Volviendo al ejemplo anterior, imaginemos que en la rama **master** también existía un archivo **hola.txt** con una cadena de texto, eso automáticamente generará un conflicto al momento de realizar el **merge** de la rama **auxiliar**. Otros conflictos que pueden presentarse en los ambientes de colaboración es que dos o más integrantes del equipo realicen modificaciones sobre un mismo código o archivo a la vez, así como también se pueden generar conflictos en los casos que un integrante elimine un archivo y otro realice pull hacia el repositorio remoto, se generará un conflicto ya que posee un archivo que fue anteriormente eliminado por otro integrante.

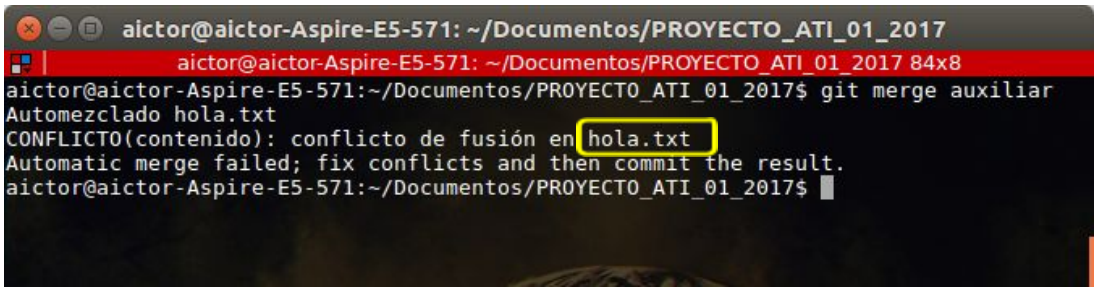
Tomando en cuenta el primer escenario crearemos un conflicto a propósito:



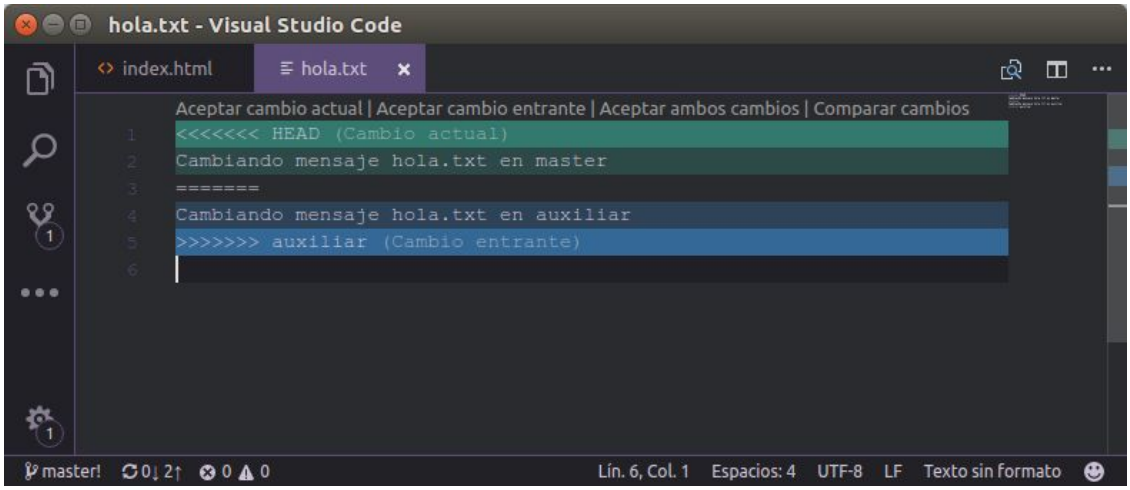
Modificamos el archivo **hola.txt** desde el **master** y procedemos a seleccionar la rama **auxiliar** para realizar la misma acción:



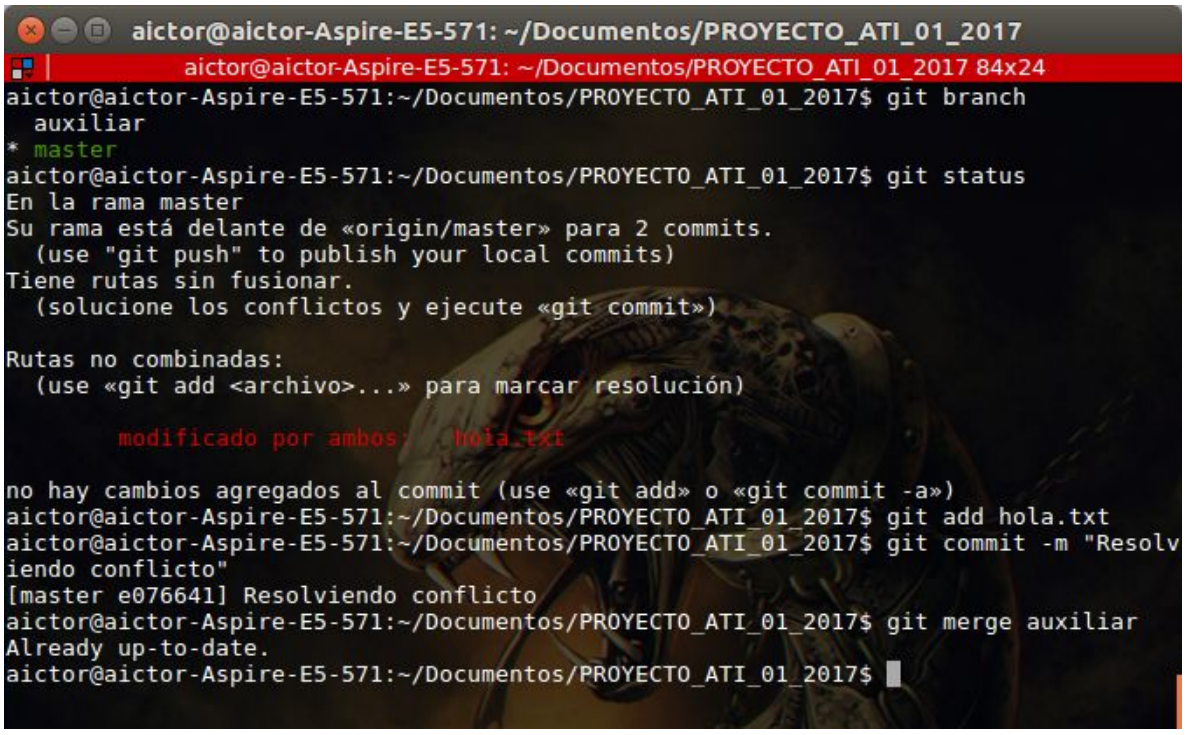
Si nos posicionamos en **master** aún se encuentra el archivo con el contenido que fue modificado en la rama **master**, como queremos traer los cambios realizados en el archivo **hola.txt** desde el branch **auxiliar**, realizamos nuevamente un **merge:**



Y ocurre un conflicto porque se supone que dos ramas están modificando un mismo archivo, y al hacer **merge** entre ellas, Git no sabe que versión escoger, si la instantánea que tiene de la rama **master** o la instantánea de la rama **auxiliar**. La forma de arreglar el conflicto es accediendo al archivo que lo ocasionó, incluso en la terminal te indica el archivo que presenta el conflicto, en este caso es **hola.txt**.



Los conflictos usualmente aparecen con esta estructura, en la cual **<<<<<<< HEAD** hace referencia al cambio que existe en el branch actual hasta llegar a **=======**, luego procede el segmento de código que se encuentra en la rama **auxiliar** cerrando finalmente con **>>>>>>>**. Lo último que queda por hacer es escoger con cuál sección de código nos quedaremos, guardamos, la subimos, y solucionado el conflicto, en este caso como el conflicto ocurre porque el archivo **hola.txt** tiene un contenido distinto en **auxiliar** que es donde queremos realizamos **merge**, entonces simplemente nos quedamos con la sección de código que hacía referencia al branch **auxiliar**, quedando de la siguiente manera:



Realizamos de nuevo los pasos anteriores y finalmente queda resuelto el conflicto.

5. Evaluación

1. Crearse una cuenta en **Git** https://github.com, la cual utilizará posteriormente para el proyecto.
2. Crear un nuevo repositorio denominado **Laboratorio_06_ATI**.
3. Localmente, crear un directorio de trabajo llamado **Actividad_06_ATI** y realizar **git init** sobre el mismo.
4. Agregar a este directorio los archivos ubicados en el portal de asignaturas, en la sección **Otros** con nombre **Archivos_Lab_06**.
5. Realizar al menos tres modificaciones sobre cada archivo utilizando tres branch distintos: "**html**", "**css**" y "**javascript**" (que sean coherentes y que no sean copias) y subirlas a su repositorio remoto (se evaluará que hayan realizado commit correctamente).
6. Cada estudiante debe realizar modificaciones sobre el repositorio remoto de al menos **dos** de sus compañeros, y cada modificación la realizarán con un **branch distinto**, es decir, crearán a su vez, dos branch "**modificador_01**" y "**modificador_02**".
7. Deben realizar al menos un **merge** (forma libre, excluyendo la que se encuentra en el enunciado del presente laboratorio).
8. Generar un **conflicto adrede** y **resolverlo** (deben explicar brevemente la causa del conflicto y cómo lo resolvieron).
9. Deben tomar capturas de pantalla por cada uno de los puntos anteriormente expuestos, y por cada punto su **respectiva justificación** en caso de ser necesario (al menos para las modificaciones, la creación de branch y los conflictos).
10. Utilizar Trello https://trello.com/, y colocar las actividades realizadas en su respectivo kanban (tablero) con la duración en las que fueron realizadas incluyendo aspectos adicionales abordados en el laboratorio, para posteriormente enviar dicha gestión al preparador mediante el uso de compartimiento por enlace.

6. Entrega

Fecha de entrega: Cada grupo de laboratorio tendrá exactamente 1 semana para desarrollar la actividad, es decir:

Grupo Lunes → Entrega el lunes de la semana siguiente

Grupo Martes → Entrega el martes de la semana siguiente
Grupo Jueves → Entrega el jueves de la semana siguiente
(Subir al portal de asignaturas 2).

Nota: ***las copias serán penalizadas con la nota mínima en el laboratorio total, es decir, si el estudiante asistió y realizó su actividad, a pesar de obtener 5 puntos, si se copió obtendrá 0 puntos en la nota final de ese laboratorio.***

Formato de entrega: [ATI][Laboratorio_6]<Nombre_Apellido>.zip

"Comentar el código es como limpiar el cuarto de baño; nadie quiere hacerlo, pero el resultado es siempre una experiencia más agradable para uno mismo y sus invitados" ***- Ryan Campbell***