Taller 1 **-** Git + GitHub

# Objetivos

* Familiarizarse con los conceptos y operaciones de un control de versiones distribuido.
* Aprender a administrar un depósito de código alojado en un repositorio.
* Hacer uso de la línea de comandos para ejecutar las operaciones Git sobre un repositorio

# Introducción

Los sistemas de control de versiones permiten llevar un registro de todos los cambios realizados a cualquier artefacto generado durante un proceso de desarrollo, en particular artefactos de código. Adicionalmente a los cambios, se incluyen metadatos, es decir datos de contexto relacionados con el cambio tales como la fecha del cambio, el autor del cambio, y un mensaje de “commit” ingresado por el autor que está orientado a describir el porqué de los cambios. Ejemplos de sistemas de control de versiones (SCV) ampliamente usados por la comunidad son Subversion – SVN (<https://subversion.apache.org/>) y Git (<https://git-scm.com/>).

Los SCVs son un conjunto de utilidades que normalmente se pueden invocar vía línea de comandos, a través de clientes locales como TortoiseSVN (<https://tortoisesvn.net/>) o Source Tree (<https://www.sourcetreeapp.com/>), plugins embebidos en los IDEs. También existen herramientas que corren del lado del servidor remoto para efectos de alojamiento y administración de los repositorios; ejemplos de estas herramientas son Subversion Edge (<http://www.collab.net/products/subversion>), ProjectLocker (http://projectlocker.com/), GitHub (<http://github.com>), GitLab (<https://about.gitlab.com/>), y BitBucket (<https://bitbucket.org/>). Las dos últimas ofrecen servicios de alojamiento y administración tanto para repositorios privados como públicos de forma gratuita.

En el caso particular del curso ISIS1225 trabajaremos con el Sistema de Control de Versiones **Git,** el entorno de desarrollo (IDE) **Visual Studio Code** (VS Code) y con el administrador de repositorios **GitHub.** Para efectos de manejo local de operaciones se sugiere la línea de comandos, pero en dado caso que prefiera usar un cliente con interfaz gráfica se sugiere usar la extensión que incluye VS Code que incluye **GitHub Desktop** (<https://desktop.github.com/>).

Para más información acerca de las diferencias entre los SCVs centralizados y distribuidos se sugiere leer las siguientes páginas en internet:

* <http://blogs.atlassian.com/2012/02/version-control-centralized-dvcs/>
* <https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control>

Para más información acerca de los beneficios de Git, se sugiere visitar la siguiente página:

* <https://git-scm.com/about>

# Creación de una cuenta en GitHub

Ingrese al siguiente enlace <https://github.com/> y cree una cuenta en **Sign up**, recuerde usar su correo **uniandes** para hacerlo. Si usted ya tiene una cuenta de GitHub con su correo uniandes, puede pasar al siguiente paso.

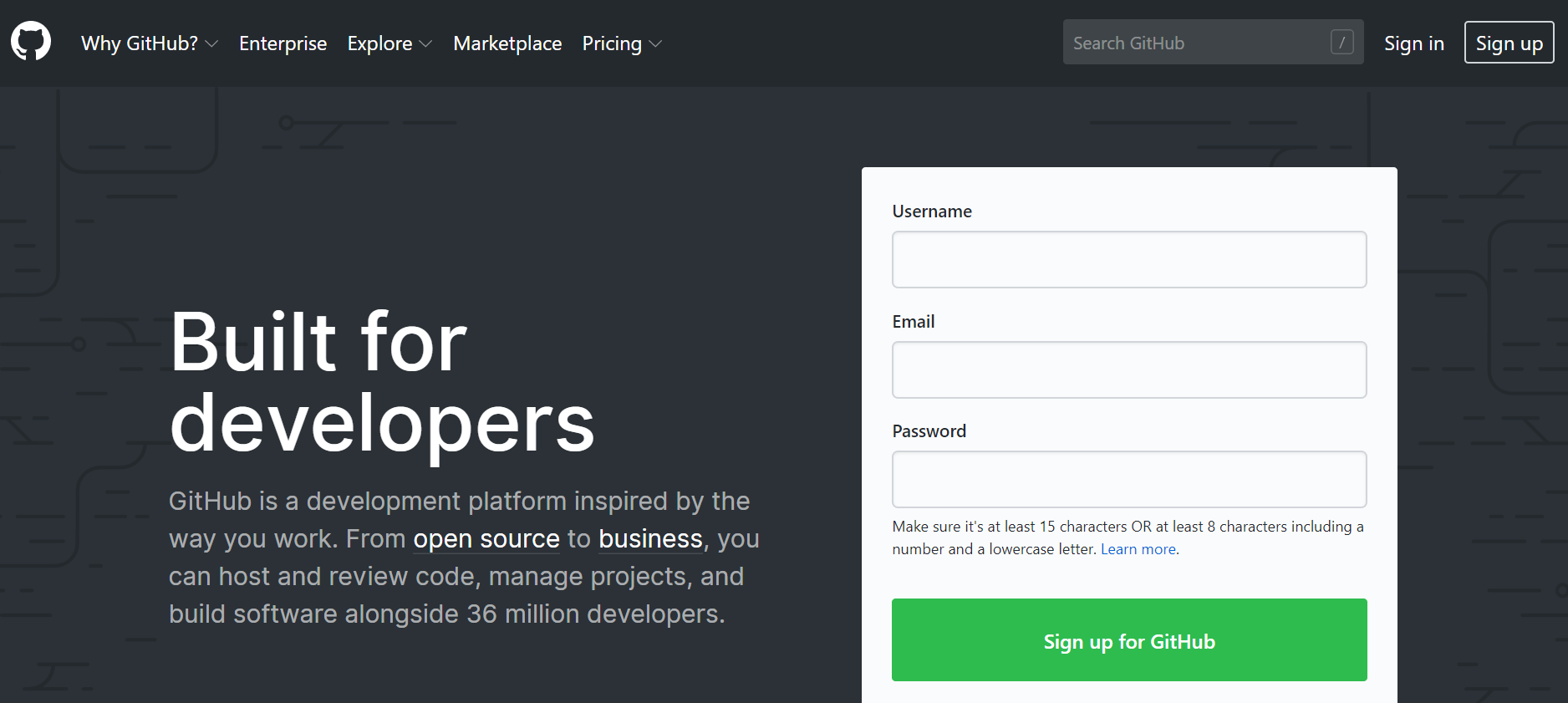


Figura 1. Pantalla inicial de GitHub

# Ingrese a su cuenta de GitHub

Para ingresar diríjase al **Sign in** de la página inicial e ingrese sus credenciales con las que creo su cuenta, recuerde que debe ser creado con su correo Uniandes.

Al crear una cuenta en GitHub, usted tendrá la posibilidad de alojar repositorios públicos y privados de forma remota, que puede ser **clonados** localmente (es decir copiados a un espacio de trabajo local como su computador) por usted o por otros colaboradores. Adicionalmente usted cuenta con la posibilidad de clonar proyectos de otros usuarios directamente a su cuenta GitHub (es decir en el servidor remoto); esta última operación se conoce como **Fork,** y también se puede realizar en el caso de un proyecto suyo, es decir, hacer *fork* de un proyecto propio consiste en crear una copia del proyecto, pero en el servidor remoto (GitHub)**.**

Para más detalles acerca de las diferencias entre las operaciones **fork** y **clone** se sugiere leer las siguientes guías**:**

* <https://help.github.com/articles/fork-a-repo/>
* <https://stackoverflow.com/questions/10963878/how-do-you-fork-your-own-repository-on-github>

# Fork: Realizar una copia del código de ejemplo a su depósito personal en GitHub

Se va a trabajar en el repositorio 202010 disponible en el usuario isis1225. La URL de acceso a este repositorio es <https://github.com/isis1225/202010.git>. Este repositorio contiene un proyecto ejemplo que ilustra una implementación sencilla que sirva de base para los siguientes talleres y retos del curso. El proyecto contiene las siguientes cinco carpetas:

* **ADT**: archivos Python con la definición de los Tipos Abstractos de Datos.
* **App**: aplicación Python cliente que usa las ADTs y ordenamientos para dar solución a talleres y retos.
* **Data**: archivos con los datos (csv, json, txt, etc) usados en el taller o reto. Esta carpeta NO se debe versionar.
* **DataStructures**: archivos Python con las estructuras de datos básicas (listas enlazadas y arreglos).
* **Sorting**: archivos Python que implementan los algoritmos de ordenamiento.
* **Test**: pruebas unitarias en Python para validar el código desarrollado.

Dado que el proyecto fue creado por el usuario isis1225, usted no tiene permisos para trabajar directamente sobre ese repositorio. El primer paso para poder modificar el código del proyecto es copiar el proyecto a su cuenta en el repositorio remoto, es decir, usted debe ejecutar la operación ***fork***. Para esto, usted debe estar logeado en GitHub, luego en su browser, ingrese la URL del repositorio del taller <https://github.com/isis1225/202010.git>. En su browser se desplegará una página similar a la que se muestra a continuación:

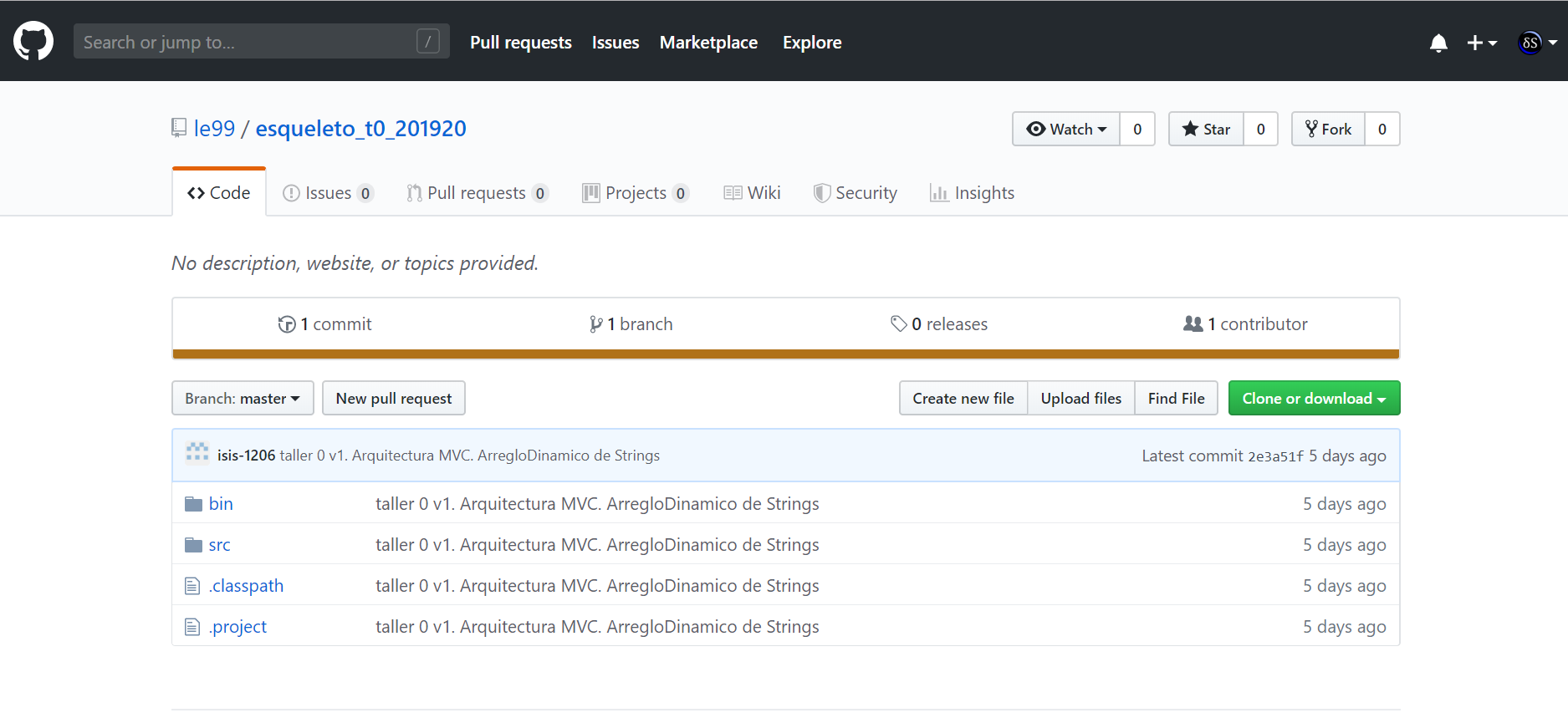


Figura 3. Página inicial del repositorio original del proyecto

Dado que usted no tiene los correspondientes permisos al repositorio solo podrá realizar un conjunto limitado de las operaciones. De nuestro interés son las que aparecen en la Figura 3, en particular la opción ***Fork***:

Para crear la copia remota en su cuenta GitHub, haga clic en el Botón ***Fork*.**

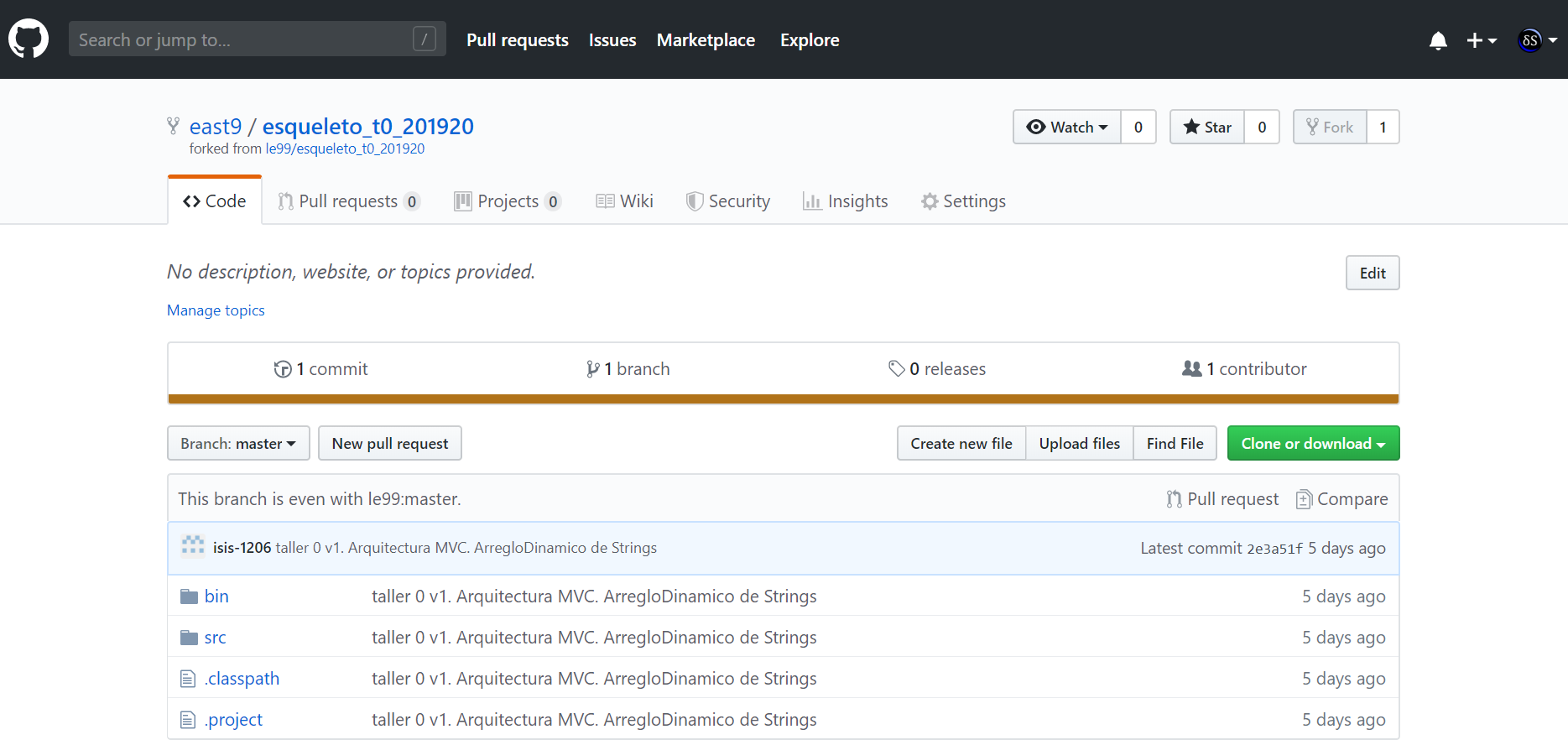


Figura 4. Página inicial del repositorio resultante del fork en su cuenta personal en GitHub, en este ejemplo “east9”

En *Settings,* *Options* puede cambiarle el nombre que quiere colocarle a su repositorio GitHub, por ejemplo, cambiarlo a copia\_202010.

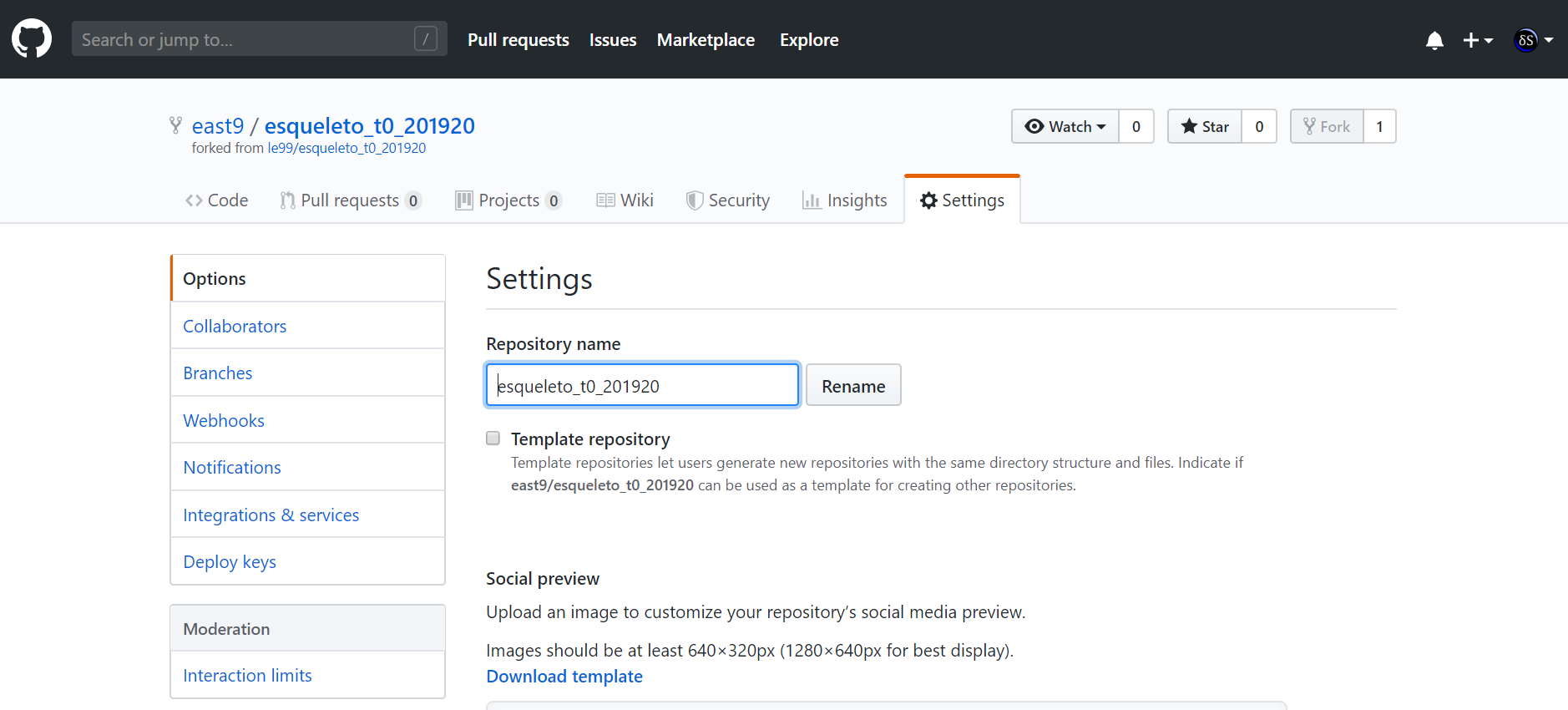


Figura 3. Página Settings / Options parta cambiarle el nombre al repositorio

La nueva URL (<https://github.com/usuario/nombre_repositorio.git>) es la dirección del repositorio en su cuenta GitHub. Este URL es el que se requiere para acceder al repositorio desde su computador local.

# Clonar el repositorio a su espacio de trabajo local

**NOTA**: Para las explicaciones que siguen, se va suponer que el repositorio al que le hizo el *fork* quedo en su cuenta GitHub, con nombre copia\_202010 y su URL es [https://github.com/su\_usuario/ copia\_202010.git](https://github.com/su_usuario/%20t1_202010.git)

Si Ud. le coloco otro nombre solo tiene que reemplazar copia\_202010 por el nombre dado.

Para hacer uso de las operaciones de Git en su espacio de trabajo local, su computador debe tener instalado Git. Para instalar Git vaya a la página de descargas <https://git-scm.com/downloads> y haga la respectiva instalación.

Al terminar la instalación puede abrir la terminal de comandos en UNIX/Linux/Mac, como se muestra en la Figura 7 donde podrá verificar la versión de su Git instalado haciendo uso del comando **git --version**. Para usar Git en Windows, se debe hacer a través de la consola “Git bash”, la cual viene incluida con el paquete de instalación para Windows.

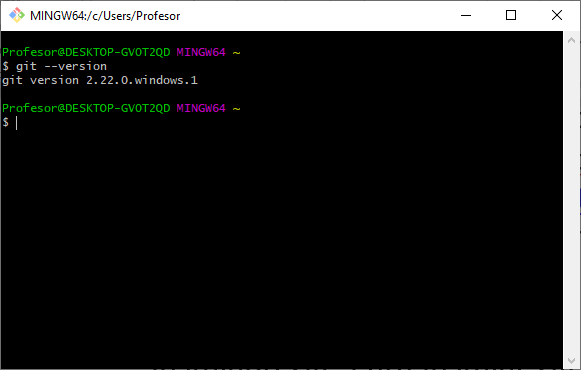
3

Figura 7. Consola y comando git --version

Una vez instalado Git en su equipo, podrá clonar el repositorio copia\_202010.git en su cuenta GitHub a su máquina local.

El proceso de copia local de un repositorio (es decir clonación) requiere de una URL que se pasa como parámetro al comando ***clone*** de Git que se ejecuta en la máquina local.

En este caso la URL del repositorio a clonar es el que se encuentra en la caja HTTPS de su repositorio con la estructura: <https://github.com/su_usuario/copia_202010.git>, donde <su\_usuario> corresponde al usuario asociado a su cuenta GitHub.

Una vez tenga el comando completo, cree un directorio local (en su computador) llamado **Talleres** en donde usted sacara la copia del repositorio. Ingrese a este directorio tal como se muestra en la Figura 9, y luego ejecute el comando "**git clone <URL-Repositorio-a-clonar>**" como se muestra en la Figura 8.

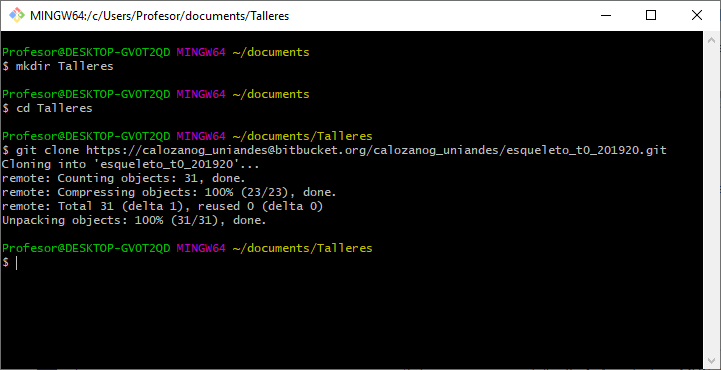


Figura 8. Creación de carpeta local y ejecución de comando git clone con la URL del repositorio en GitHub

Es posible que en este punto deba ingresar su *contraseña* GitHub. Al finalizar la clonación del repositorio debe tener un resultado similar al siguiente:

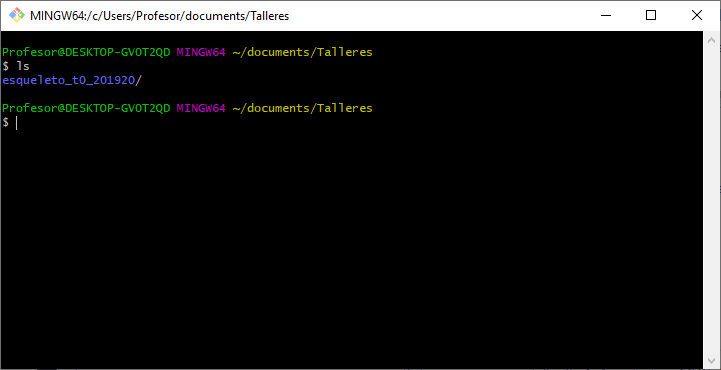


Figura 9. Clonación local del repositorio remoto. El comando *ls* en UNIX/Linux/MAC muestra el contenido de un directorio; en Windows se debe utilizar el comando *dir*

Una vez clonado el repositorio, ingrese a la carpeta copia\_202010 que se creó en su computador, usando el comando *cd* copia\_202010. Usted verá las carpetas del proyecto ya descritas utilizando el comando *dir* (windows) o *ls* (linux), como se muestra en la Figura 10. Revise el contenido de las carpetas usando el explorador de archivos de su sistema operativo.

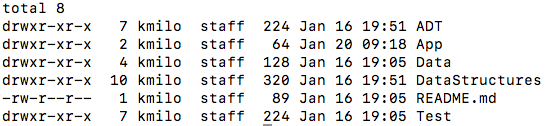


Figura 10. Contenido proyecto

El proyecto dado ofrece cuatro opciones al usuario: leer un archivo, crear una lista de palabras a partir del archivo leído, contar los elementos de la lista y ordenar los elementos de la lista. Para ver el funcionamiento de dichas opciones, inicie explorando el archivo app.py de la carpeta App. Ese programa Python hace uso de módulos incluidos en las demás carpetas.

Agregue un archivo a la carpeta copia\_202010 con nombre README.txt el cual debe contener su nombres, apellidos y código separados por espacios, ejemplo: “Pedro Pablo Perez Rodriguez 20001091”. Ahora diríjase a la terminal o Git Bash e ingrese el comando **git status**[[1]](#footnote-1); este comando es para verificar el estado de los archivos y el espacio (workspace, index, repo local) donde se encuentran los archivos/cambios:

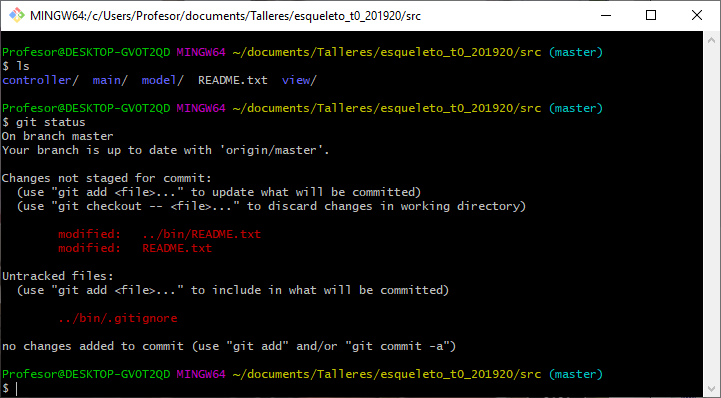


Figura 11. Comando git status

Git status

Notará que el archivo README.txt aparece como “untracked”; esto quiere decir que su archivo está en el workspace pero aún no ha sido incluido en el índex de Git (stage). Para agregar el archivo al index debe utilizar el comando **git add README.txt** y luego verifique el estado de nuevo con el comando **git status**:

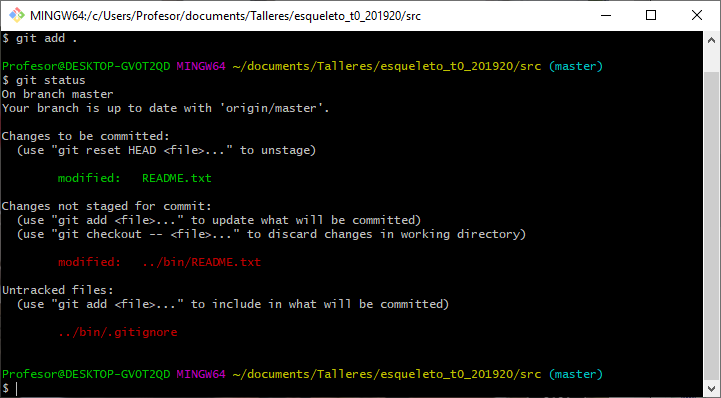


Figura 12. Comandos git add y git status

El comando **git add** puede ser usado con un archivo, listas de archivos/carpetas separadas por espacio, carpetas, o directorio actual.

Una vez los archivos/cambios están en el índex de git, estos pueden ser enviados al repositorio local con la operación **commit** usando el comando:

**git commit -m "Comentario sobre la actualizacion registrada en repositorio local"**

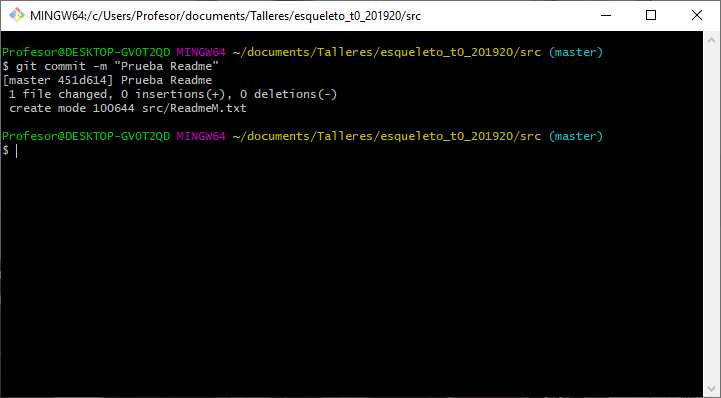


Figura 13. Comando git commit

Finalizado el **commit** puede subir los cambios al repositorio remoto (GitHub) con la instrucción **git push;** usted deberá proporcionar su password:

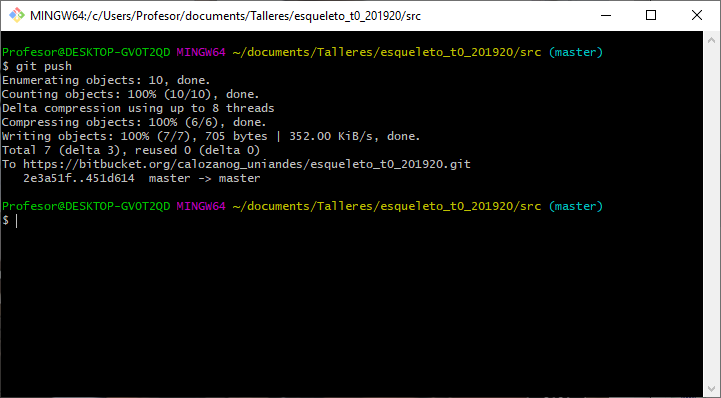


Figura 14. Git Push (autenticación)

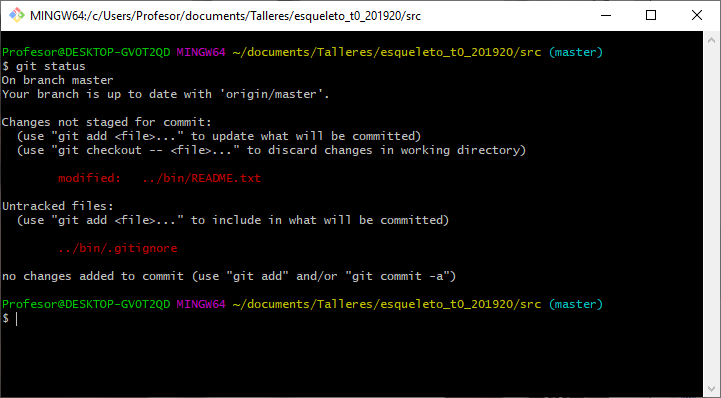


Figura 15. Git Push (finalizado)

Una vez terminado el **push**, puede ir a su navegador en la página de GitHub ***Code – Commits*** y notar que usted realizó un *commit* como se muestra en la Figura 17.

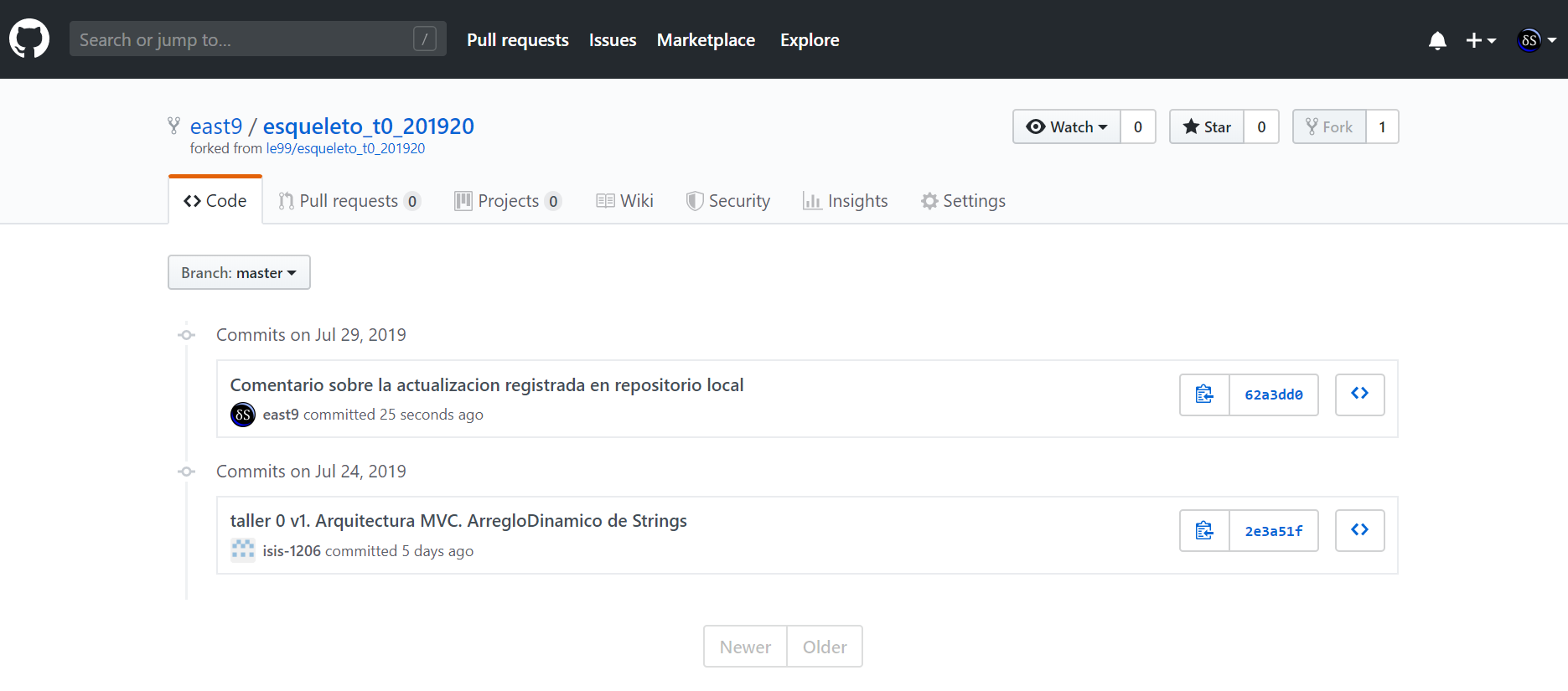


Figura 16. Commit realizado.

Más información acerca de los comandos para trabajar localmente con archivos se puede encontrar en <https://confluence.atlassian.com/bitbucket/work-on-local-source-files-223217905.html>

# Cambiar el nombre de su repositorio

Usted puede cambiar el nombre de su repositorio en la página *Settings,* *Options* de su repositorio.

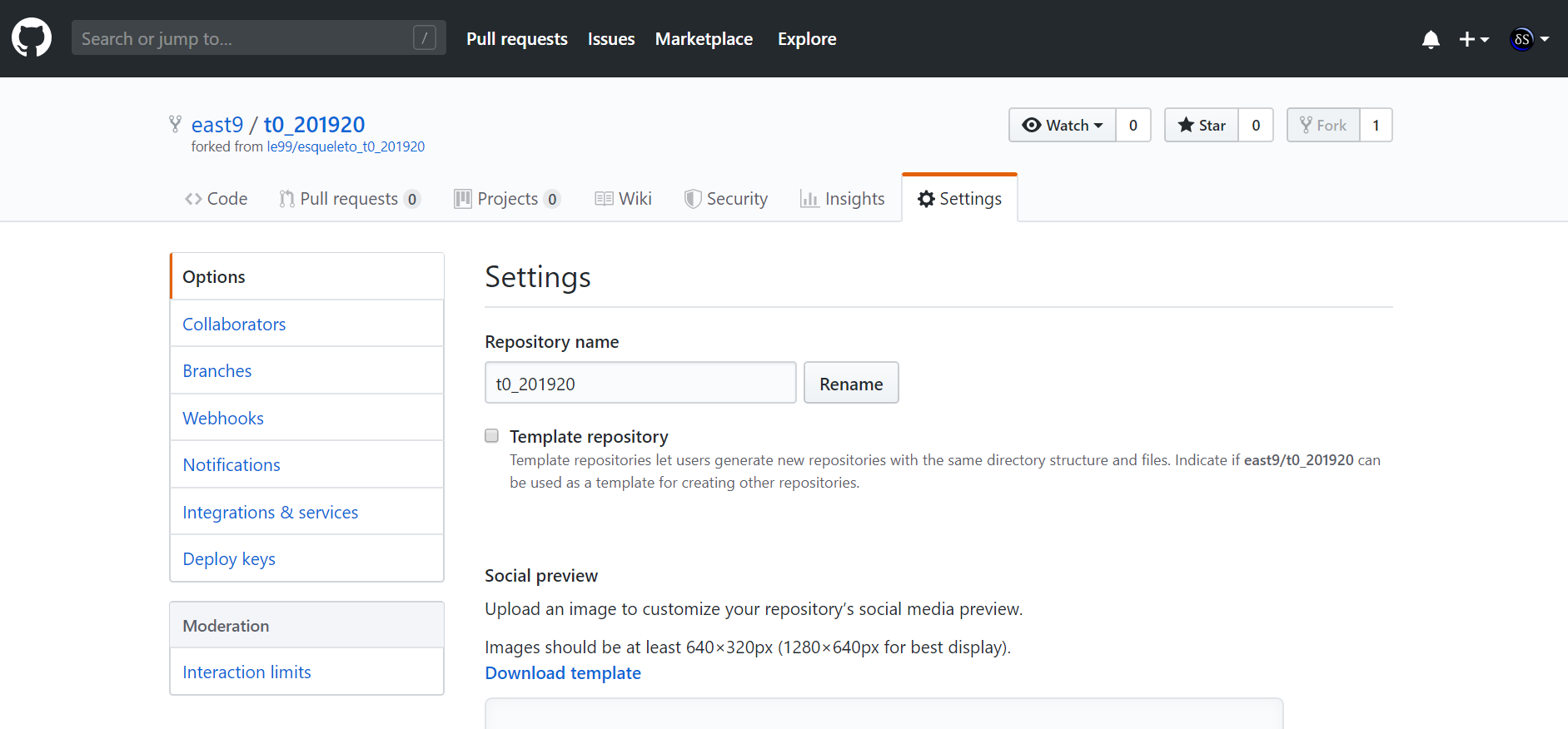


Figura 3. Página Settings / Options parta cambiarle el nombre al repositorio

# Actualice nombre del repositorio en su espacio de trabajo local

Dado que usted cambió el nombre del repositorio en GitHub, se deben actualizar las referencias locales al nombre anterior. Una forma sencilla de hacerlo es clonando de nuevo el repositorio con la nueva URL. Para eso vaya a la página *Code* y haga click en el boton “clone or download” en la caja HTTPS aparece la URL del repositorio con el nuevo nombre con la estructura:

https://github.com/usuario/t0\_202010.git

donde *usuario* corresponde al usuario asociado a su cuenta GitHub.

El identificador del repositorio corresponde al identificador del taller (t1) y al semestre de realización (202010).

Nuevamente ejecute el comando clone de git con esta nueva URL en el directorio Talleres en su máquina local:

git clone https://github.com/usuario/t0\_202010.git

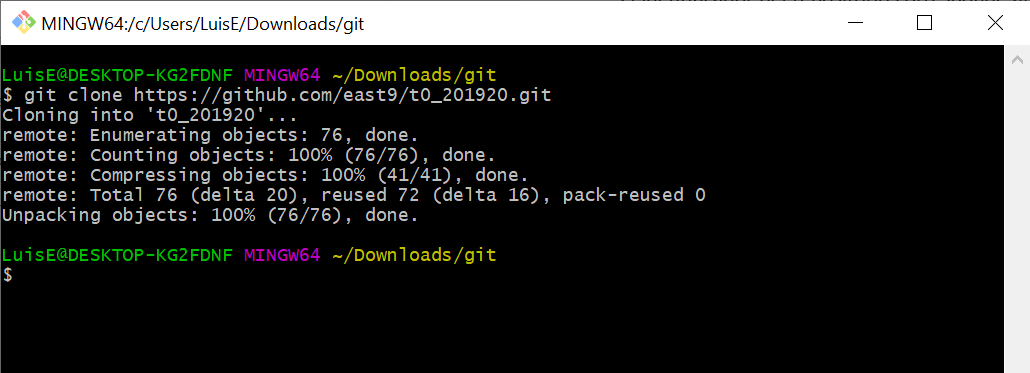
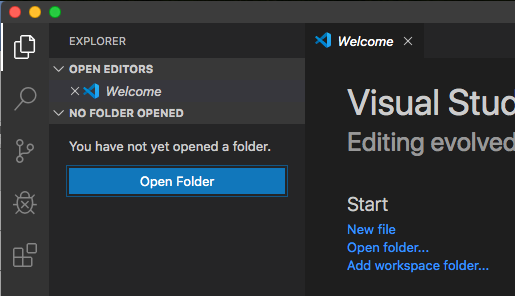


Figura 18. Clonación de repositorio

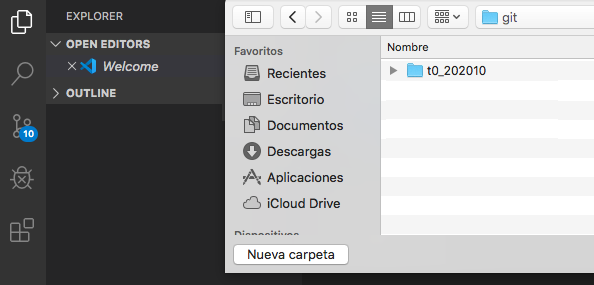
Al finalizar usted tendrá dos subcarpetas (copia\_202010, t0\_202010) en su carpeta talleres. Cada carpeta tiene una copia del proyecto original. La carpeta t0\_202010 está asociada al repositorio t0\_202010 en GitHub.

# Abrir el código desde Visual Studio Code (VSCode)

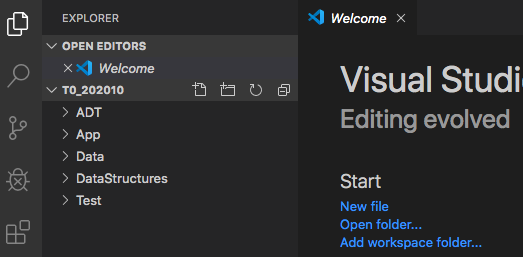
Abra VSCode y seleccione la opción de *Explorer* y luego *Open Folder*, como se muestra en la figura.



Seleccione el directorio t0\_202010 creado en los pasos anteriores del taller y clic en abrir.



Al finalizar la operación, le debe aparecer el contenido del folder t0\_202010.



# Agregar cambios al proyecto

Ahora que usted está familiarizado con el uso de Git, GitHub y VSCode, está listo para hacer cambios al proyecto. Como parte del taller usted debe modificar la lectura del archivo para cargar el archivo **README.txt** y adicionar Strings a una lista simplemente enlazada.

Abra la carpeta del proyecto (carpeta t0\_202010) en VSCode y explore el código del proyecto. El modulo *app.py* en la carpeta *App* es el punto de entrada y encargado de la interacción con el usuario. Esta app básicamente ofrece un menú de opciones para leer un archivo, crear una lista, contar y ordenar los elementos de la lista. Modifique la función de carga del archivo (***loadTxtFile***) ubicada en app.py para que lea el archivo README.txt de la ruta respectiva y retorne una lista con las palabras contenidas en el archivo (Strings separados por espacios). Finalmente, verifique que la opción de lectura del archivo en la app funciona correctamente.

Al finalizar los cambios, verifique el estado usando **git status**, y envíe los datos al repositorio remoto con los comandos **git commit** y **git push**.

# Modifique el proyecto para almacenar y ordenar las palabras en una lista.

Modifique la función de contar palabras (***wordCount***) para que retorne la cantidad de elementos de la lista leída desde el archivo. Modifique la función de ordenamiento de la lista (***sortList***) para que ordene ascendentemente las palabras dentro de dicha lista llamando a la función …..

Finalmente, valide que las opciones de leer, contar y ordenar elementos funcionan correctamente. Use de los comandos git status, git add, git commit, git push, etc, para enviar sus cambios al repositorio remoto. Verifique que en github están los últimos cambios realizados.

# Compare los tiempos de ordenamiento de la lista

Descargue los datos del reto actual a la carpeta *data* del proyecto, recuerde que no deben versionarse dichos datos. Modifique la función de carga para leer el archivo del reto usando el separador que corresponda (coma, punto y coma, espacio, tabulación, etc). Modifique la función de creación de lista (*buildList*), y el nodo de la lista enlazada (*listnode*), dentro de la carpeta *DataStructures,* para que ahora no solo almacene *info*, sino cada uno de los campos o columnas contenidos en el archivo del reto. Como le define el enunciado del reto. Modifique la función *printElement* para que imprima el contenido del nodo indicado, dado que ahora no solo se almacena un String, sino los campos de un elemento del reto.

Pruebe cargando diferentes cantidades de datos (10, 100, 1000, 10000) enviando el parámetro a la función *buildList* y registre los tiempos que toma construcción de la lista, y el ordenamiento de dichos elementos. Para ello, tome la información que se imprime en consola para la ejecución de cada función: Construir lista y ordenar lista.

Al finalizar haga uso de los comandos git status, git add, git commit, git push, etc, para enviar sus cambios al repositorio remoto. Verifique en GitHub que el *commit* fue exitoso. Para hacer la entrega del taller usted debe seguir las instrucciones del documento “Guía de Creación de Repositorios Github”.

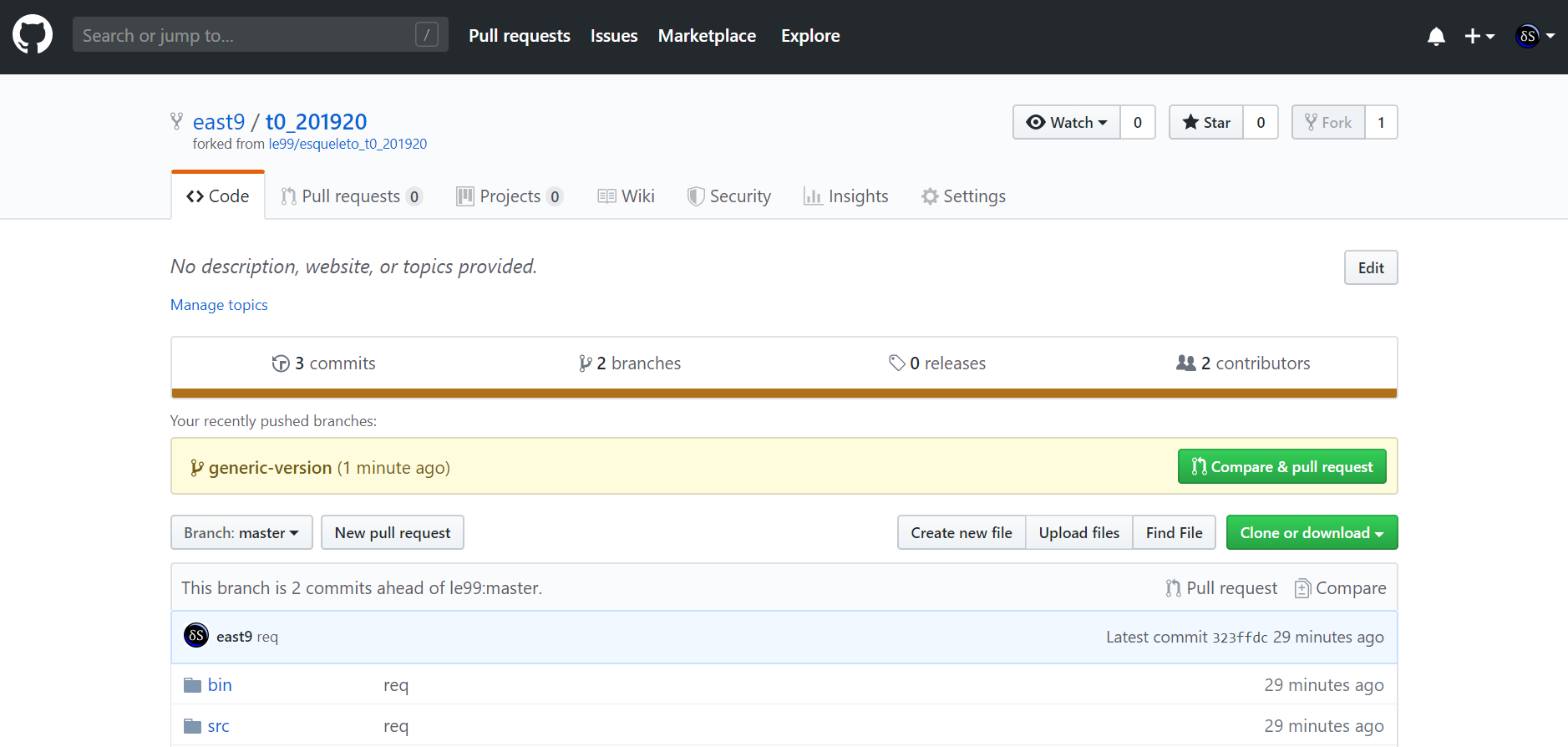


Figura 23. Estado final del repositorio con 2 ramas

# Algunos comandos en GNU/Linux (funcionan bajo Git Bash)

|  |  |
| --- | --- |
| **Comando** | **Descripción** |
| **pwd** | El comando "pwd" informa cual es el directorio actual |
| **ls** | Use el comando "ls" para **listar los contenidos** del directorio actual.  Puede ver todos los archivos ocultos con el comando "ls -a". |
| **cd** | Utilice el comando "cd" para **cambiar de directorio** a otro dado como parámetro.  Si no le pasamos ningún parámetro, por defecto nos situará en nuestro directorio  *home*.  Para volver al directorio inmediatamente superior (en el árbol de jerarquías) al actual, se debe introducir el comando "cd ..".  Generalmente, el directorio destino se puede pasar de forma absoluta o relativa. De forma relativa significa que partiremos del directorio actual y subimos un directorio en la jerarquía usando "..". Por ejemplo, si estamos en el directorio /usr/bin/ y queremos ir al /root/, deberíamos introducir el siguiente comando: “cd ../../root” (la secuencia ../../ permiten subir dos niveles en la jerarquía de directorios, a partir de la cual ya podemos acceder a /root/). De forma absoluta siempre partimos de la raíz, de manera que el comando que utilizaríamos para el ejemplo anterior sería: “cd /root”. |
| **mkdir** | Utilice el comando "mkdir" para **crear** un directorio que le indiquemos como parámetro |
| **rm** | Use el comando "rm" para **borrar** los archivos que le indiquemos como parámetro |
| **rmdir** | Utilice el comando "rmdir" para **borrar** el directorio que le indiquemos como parámetro. Si quisiéramos borrar completamente un directorio y todo su contenido, podríamos utilizar “rm -r <directorio a borrar>” |
| **cp** | Para **copiar archivos** de un lugar a otro tenemos el comando cp, al que siempre debemos indicar el fichero o directorio origen y el lugar o nombre de destino, aunque sea en el directorio actual.  Se necesitan dos argumentos: el primero es la ubicación del archivo a copiar, el segundo es dónde copiar. |
| **mv** | Para **mover** **archivos** de un lugar a otro tenemos el comando mv. Necesita dos argumentos, al igual que el comando cp. |
| **find** | Es el comando más versátil para realizar una búsqueda. Nos permite filtrar los archivos para encontrar desde los que tienen un determinado nombre, los modificados o creados a partir de una cierta fecha, los que tienen ciertos permisos, entre otros. |
| **locate** | Se trata de otro comando para buscar, pero, a diferencia del anterior, utiliza una base de datos interna que se actualiza periódicamente y nos permite hacer búsquedas bastante más rápidas. |
| **man y --help** | Para **saber más sobre** un comando y cómo usarlo, use el comando man. Este comando muestra las páginas de manual del comando; por ejemplo, "man cd" muestra las páginas de manual del comando cd.  Escribir el nombre del comando y el argumento --help lo ayuda a mostrar las formas en que se puede usar el comando (por ejemplo, cd --help). |

1. https://git-scm.com/docs/git-status [↑](#footnote-ref-1)