Universidad Andrés Bello®
Conectar · Innovar · Liderar

DESARROLLO DE LA INTERFAZ DE USUARIO ANDROID

Fundamentos de Git

Respaldar nuestro código fuente es una necesidad crítica, ya que en programación es nuestro principal activo, sobre todo cuando trabajamos en proyectos de gran envergadura y con equipos variados.

Pero además de almacenar nuestro código fuente, cuando trabajamos en equipo con más desarrolladores, necesitamos gestionar nuestro código para que no tengamos problemas al momento de desarrollar un software. Es muy común que dos o más desarrolladores trabajen en un mismo módulo y muchas veces modifiquen los mismos archivos de código fuente. En ese caso también es común que surjan conflictos de código y es necesario solucionarlos para que un desarrollador no borre los cambios que hizo otro. Esto es lo que se conoce como gestión de versiones de código fuente.

Git es un sistema de control de versiones gratuito y de código abierto, creado originalmente por Linus Torvalds en 2005. A diferencia de los antiguos sistemas centralizados de control de versiones, como SVN y CVS, Git está distribuido: cada desarrollador tiene el historial completo de su repositorio de código de manera local. De este modo, la clonación inicial del repositorio es más lenta, pero las operaciones posteriores, como commit, blame, diff, merge y log son mucho más rápidas.

Git nos permite versionar nuestro código, controlar el flujo de trabajo y crear espacios de trabajo seguro mientras desarrollamos.

Git incluye las funcionalidades de crear ramas y fusiones y reescribir historiales de repositorios, lo cual ha dado como resultado muchas herramientas y flujos de trabajo innovadores y eficaces. Las solicitudes de incorporación de cambios son una herramienta popular con la que los equipos pueden colaborar en las ramas de Git y revisar con eficacia el código de los demás. Git es el sistema de control de versiones más utilizado en el mundo hoy en día, y se considera el modelo actual de desarrollo de software.

Según la encuesta anual realizada por stack overflow el año 2021 git es el amplio líder de los sistemas de control de versiones, ocupando más del 90% de las preferencias de los desarrolladores y es considerada una herramienta fundamental para cualquier desarrollador. Para más información acerca de esta interesante encuesta sobre diferentes temas de desarrollo de software pueden visitar el siguiente enlace: https://insights.stackoverflow.com/survey/2021/#most-popular-technologies-tools-tech-prof.

Instalando Git
Vamos a instalar Git en nuestro computador. Ahora veamos cuál necesitas dependiendo del sistema operativo que tengas (Mac/Linux, Windows).

El primer paso es verificar si ya tenemos instalado git en nuestro sistema. Esto lo podemos realizar escribiendo el comando git --version en nuestra terminal. Si está instalado veremos un mensaje como se muestra en la Figura 1.

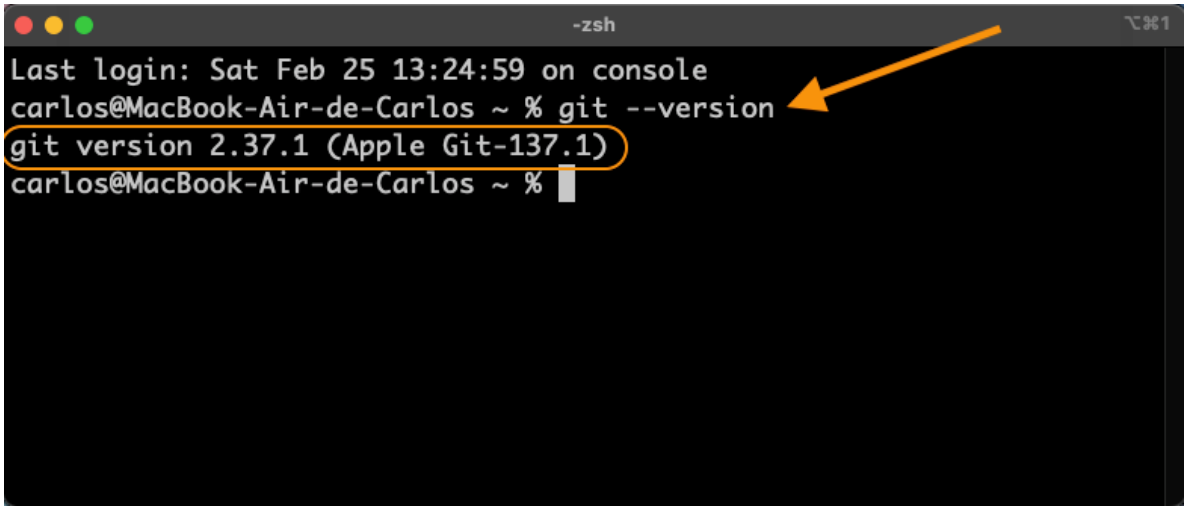

Figura 1 – Versión de Git instalada en el Sistema Operativo.

Si estás en Windows y la palabra terminal te parece algo extraña, puedes ver si te aparece algún ícono que diga GitBash. Si no aparece lo más probable es que no tengas Git instalado.
Nosotros vamos a ocupar la Interfaz de Línea de Comandos, más conocida como terminal para gestionar nuestro código fuente con Git. Los comandos que se van a utilizar son comandos de tipo Unix, por lo que en Windows cuando instalemos Git, también se va a instalar un terminal llamado GitBash.
Para saber más acerca de los comando elementales de un terminal de tipo Unix que ocupan los sistemas Mac y Linux recomendamos visitar los siguientes enlaces: https://programminghistorian.org/es/lecciones/introduccion-a-bash y https://www.softzone.es/linux/tutoriales/terminal-linux/.

En computadores Mac Git viene instalado por defecto. Pero si por algún motivo no lo tienes, sigue los siguientes pasos:
1. Entra al sitio de descarga de Git en el siguiente enlace: https://git-scm.com/downloads.
2. Descarga el archivo para Mac.
3. Ejecuta el archivo descargado y sigue los pasos del instalador.

En Linux, si no está instalado, debemos utilizar el gestor de paquetes que use tu distribución Linux. Para Ubuntu y derivados como Linux Mint el comando en el terminal sería:

```
sudo apt install git
```

En Windows, si no está instalado debes seguir los siguientes pasos:
1. Entra al sitio de descarga de Git para Windows en el siguiente enlace: https://git-scm.com/download/win.
2. Descarga el archivo dependiendo de tu versión del sistema operativo.
3. Ejecuta el archivo descargado y sigue los pasos del instalador.

Configurando Git
Ahora que tenemos Git instalado, nuestro siguiente paso será configurarlo en nuestro equipo.
Lo primero que debemos hacer es configurar nuestro nombre y correo en Git. Esto es necesario para poder hacer seguimiento de nuestro código.
Para ello usaremos los siguientes comandos dentro de la consola:

```
git config --global user.name "Nombre Apellido"
git config --global user.email correo@mail.com
```

Donde iría el nombre y el email del usuario.

Una vez ingresado cada comando ninguno de ellos nos da alguna confirmación, por lo que debemos confirmar si todo está bien con el siguiente comando:

```
git config --list
```

Y deberíamos ver dentro del listado de opciones obtenida los siguientes elementos:

```
user.name=Nombre Apellido
user.email=correo@mail.com
```

Si ves este mensaje es que todo está bien y estarías listo para usar Git.

Usando Git
No hay que olvidar que Git es una herramienta para gestionar nuestro código fuente, por lo que debemos ingresar a una carpeta donde esté nuestro proyecto.
Por ejemplo a continuación en la Figura 2 veremos cómo acceder a la carpeta de nuestro proyecto Android Hola Mundo.

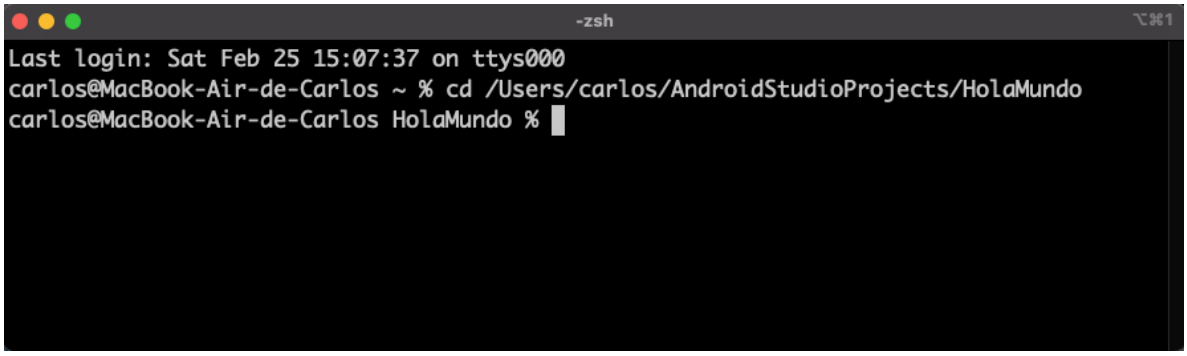

Figura 2 – Ingresando con comando cd a la carpeta HolaMundo

Una vez dentro de la carpeta HolaMundo, vamos a ejecutar el comando ls -a para poder ver el contenido de la carpeta. Hay algunos archivos o carpetas que comienzan con un punto (.). Esos archivos son especiales y en sistemas Unix están ocultos, es decir no se ven en un listado normal. En la Figura 3 veremos el contenido de esta carpeta.

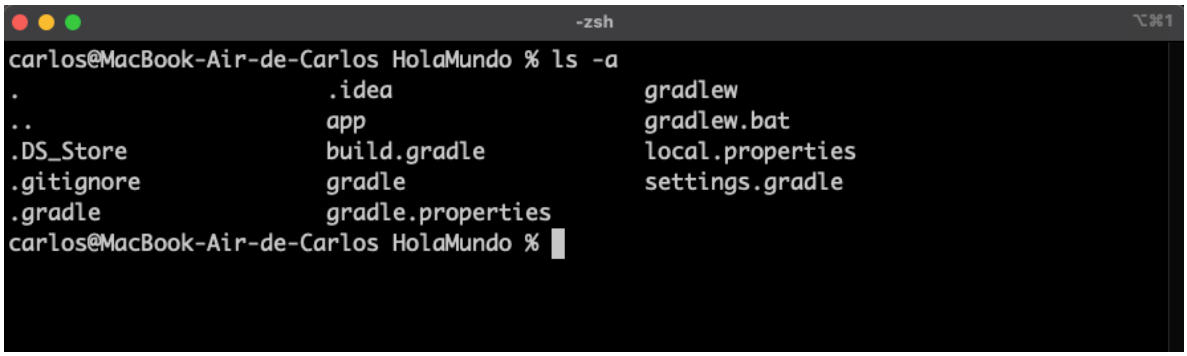

Figura 3 – Contenido carpeta HolaMundo

Ahora que estamos en la carpeta y confirmamos que es nuestra carpeta de proyecto vamos a inicializar Git para que nos cree un repositorio. Lo haremos con el comando git init.
En la Figura 4 vemos la ejecución del comando y la respuesta obtenida.

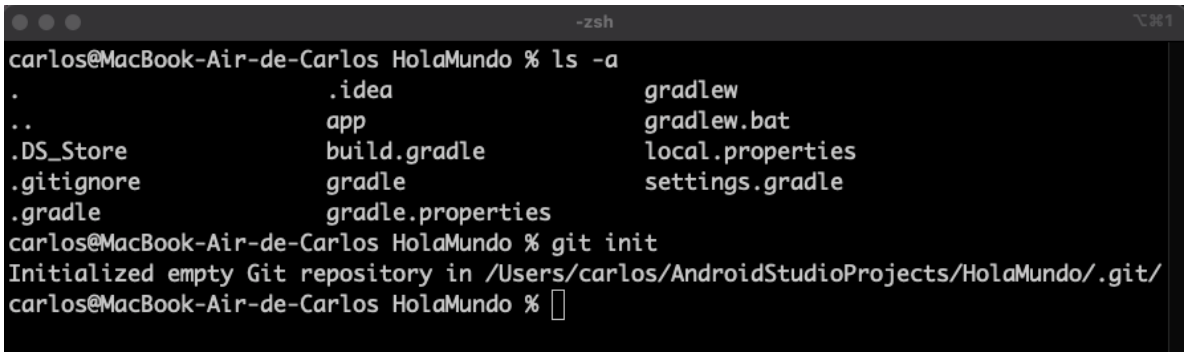

Figura 4 – Ejecución comando git init y la respuesta de creación exitosa de un nuevo repositorio.

Ahora que tenemos creado el repositorio en nuestra carpeta de trabajo debemos agregar los archivos al seguimiento de nuestro repositorio. Los archivos no se agregan de forma inmediata al repositorio y pasa por algunos estados. A continuación en la Figura 5 se muestra un flujo de trabajo con Git.

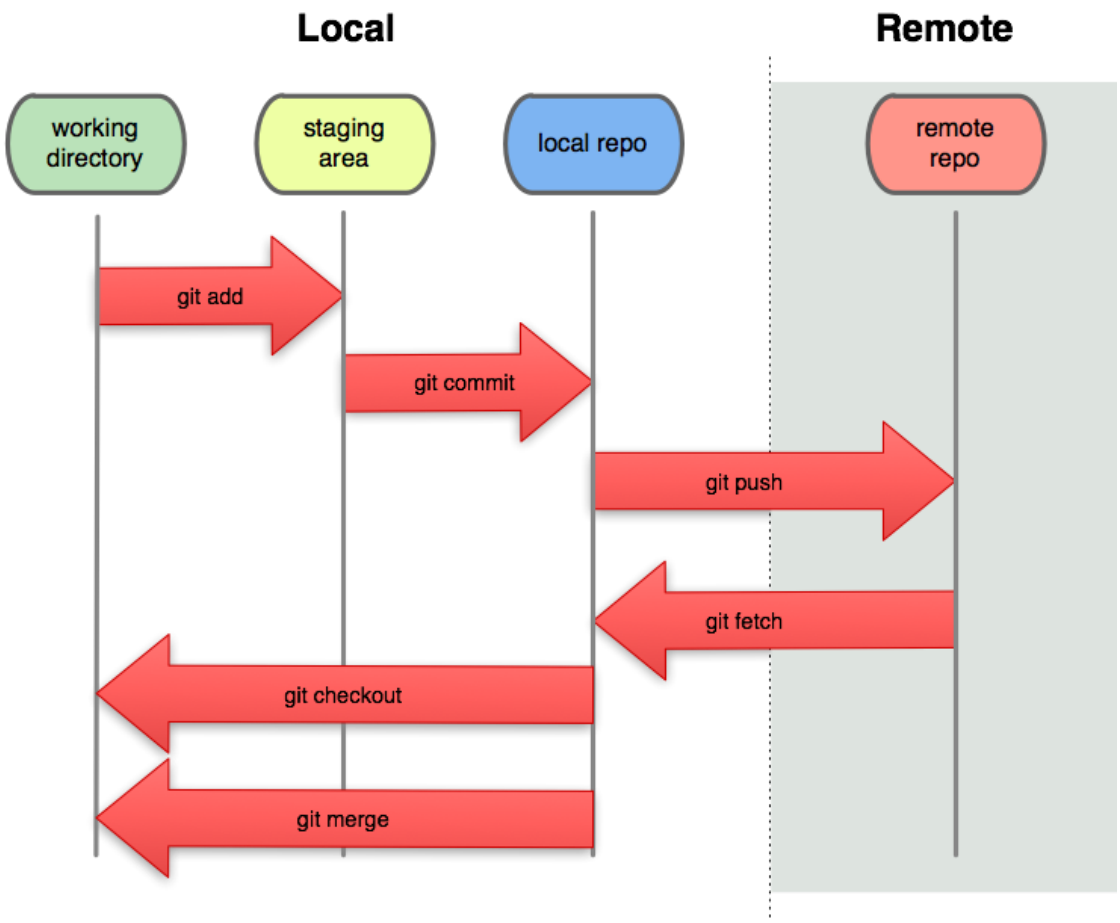

Figura 5 – Flujo de trabajo básico con Git.
Fuente: https://bluuweb.github.io/tutorial-github/guia/.

En este diagrama 'working directory' corresponde a nuestra carpeta del proyecto. Ahí están los archivos cue modificamos cuando estamos trabajando. 'stagin area' corresponde a un estado de Git donde el archivo está siendo 'seguido' por Git, está en preparación antes de ser confirmado. 'local repo' corresponde a nuestro repositorio local de Git- Aquí Git lleva un control de los archivos confirmados, es decir que se confirmó que estarían en el repositorio. 'remote repo' corresponde a un repositorio Git remoto, que está en un servidor donde nuestro repositorio Git local (que está en nuestro equipo) se sincroniza con un repositorio Git que está en un servidor. De esta forma podemos respaldar nuestro código fuera de nuestro equipo.

git status

Este comando nos ayuda a saber el estado de los archivos de nuestro repositorio. Con este comando podremos saber si nuestros archivos están en el repositorio y en qué estado están.
En la Figura 6 veremos la ejecución de este comando en nuestro recién creado repositorio en nuestro proyecto ya existente.

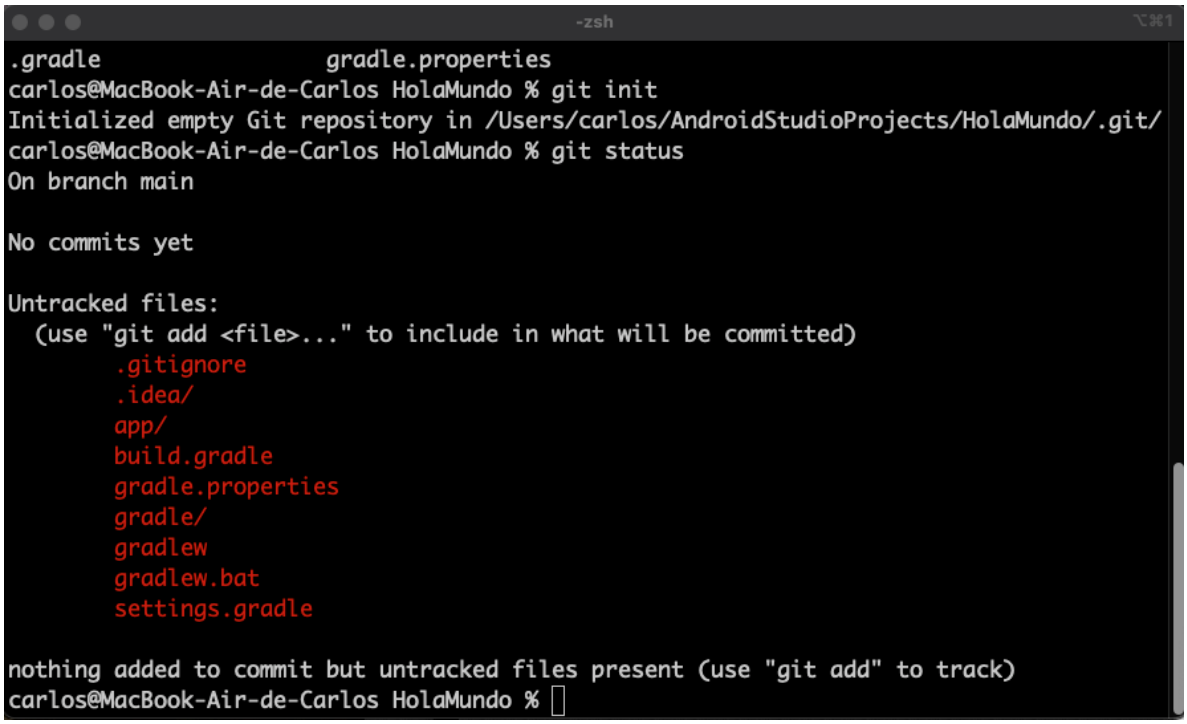

Figura 6 – Primer git status de nuestro proyecto HolaMundo.

Este primer git status nos da mucha información valiosa. Lo primero es que nos dice que estamos en la rama (branch) main. Esta es la rama creada por defecto, nuestra rama principal. Para obtener más información sobre las ramas puedes visitar el siguiente enlace: https://git-scm.com/book/es/v2/Ramificaciones-en-Git-%C2%BFQu%C3%A9-es-una-rama%3F.
Más abajo nos muestra un listado en rojo de archivos y carpetas que no estamos siguiendo (untraked). Y nos aconseja a través del comando git add agregarlos al seguimiento, ver Figura 5.

git add

Con este comando agregamos nuestros archivos creados y/o modificados a nuestro repositorio. Podemos agregar de a uno cómo nos sugiere git status o podemos agregar de varios a la vez utilizado git add . (el punto va) o git add --all.
En la Figura 7 veremos el resultado de agregar los archivos al seguimiento con git add . , cómo no nos da ningún resultado ejecutamos git status para saber su hubo cambios.

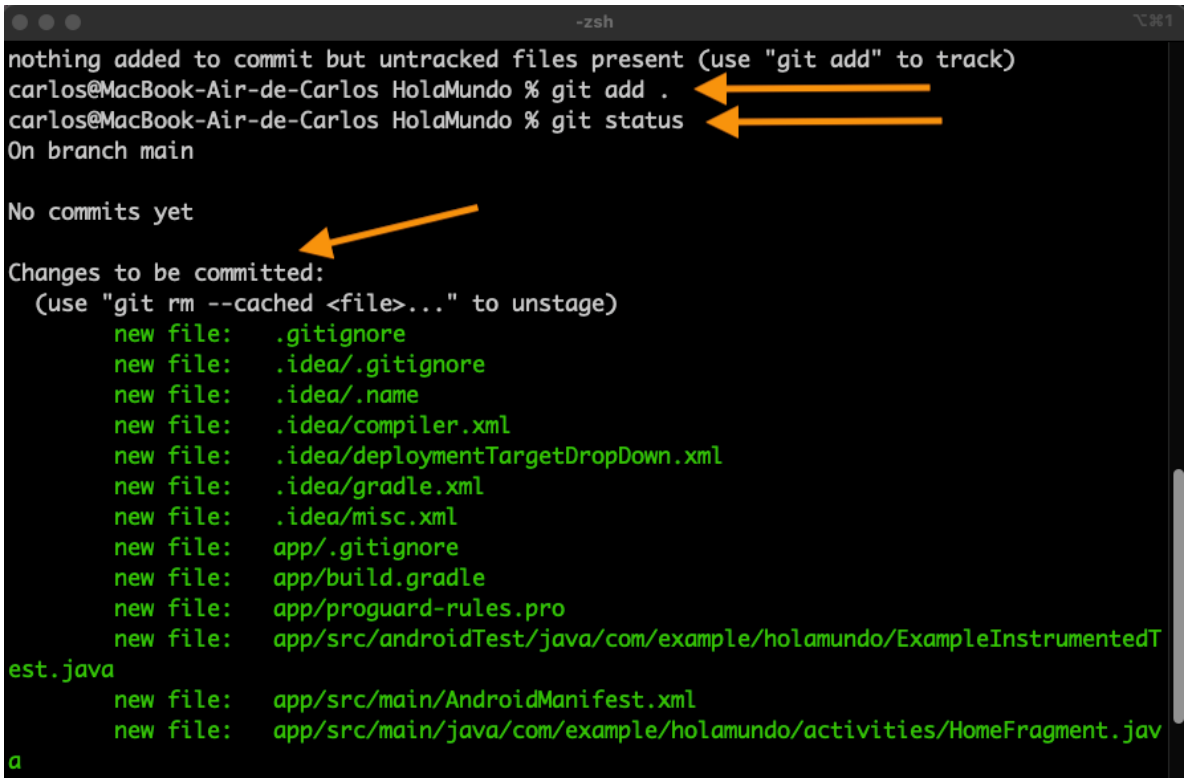

Figura 7 – git add . y luego git status para verificar estado del repositorio.

En la Figura 7 podemos ver que luego de agregar los archivos al seguimiento con git add los archivos (cada uno de los archivos existentes, inclusive lo de las carpetas) pasaron a color verde. También la flecha de más abajo nos apunta a un mensaje de cambios a ser confirmados. Ya que si no están confirmados no forman parte del repositorio.

git commit

Para que nuestros archivos se guarden en el repositorio se deben confirmar los cambios. Esto lo hacemos con el siguiente comando:

git commit -m “Nombre o descripción del commit”

A continuación en la Figura 8 veremos nuestro primer commit en nuestro proyecto HolaMundo.

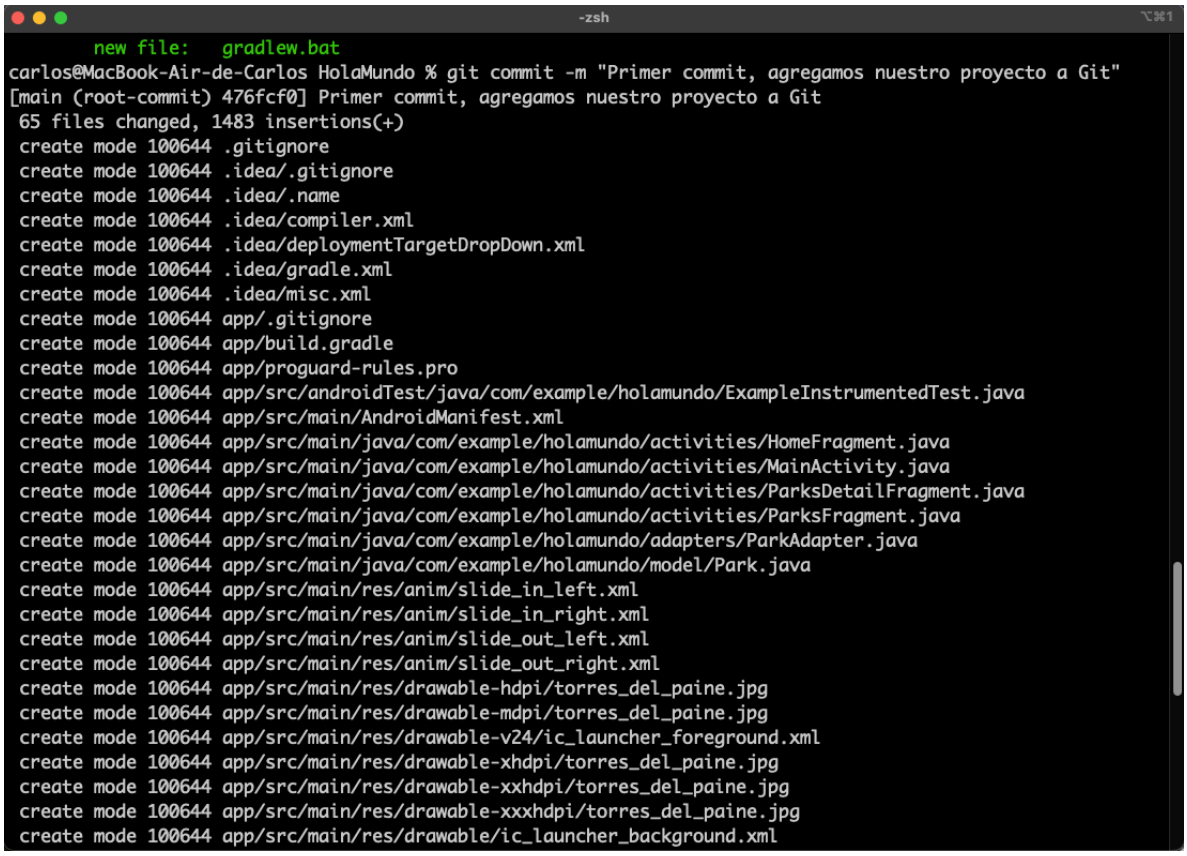

Figura 8 – Primer commit proyecto existente HolaMundo.

En la Figura 8 podemos ver que luego de ejecutar el comando git commit este responde con los archivos agregados al repositorio que en este caso son todos los archivos del proyecto.
Ahora vamos a ejecutar nuevamente un git status para ver cómo quedó nuestro repositorio. La figura 9 muestra el estado de nuestro repositorio después del commit.

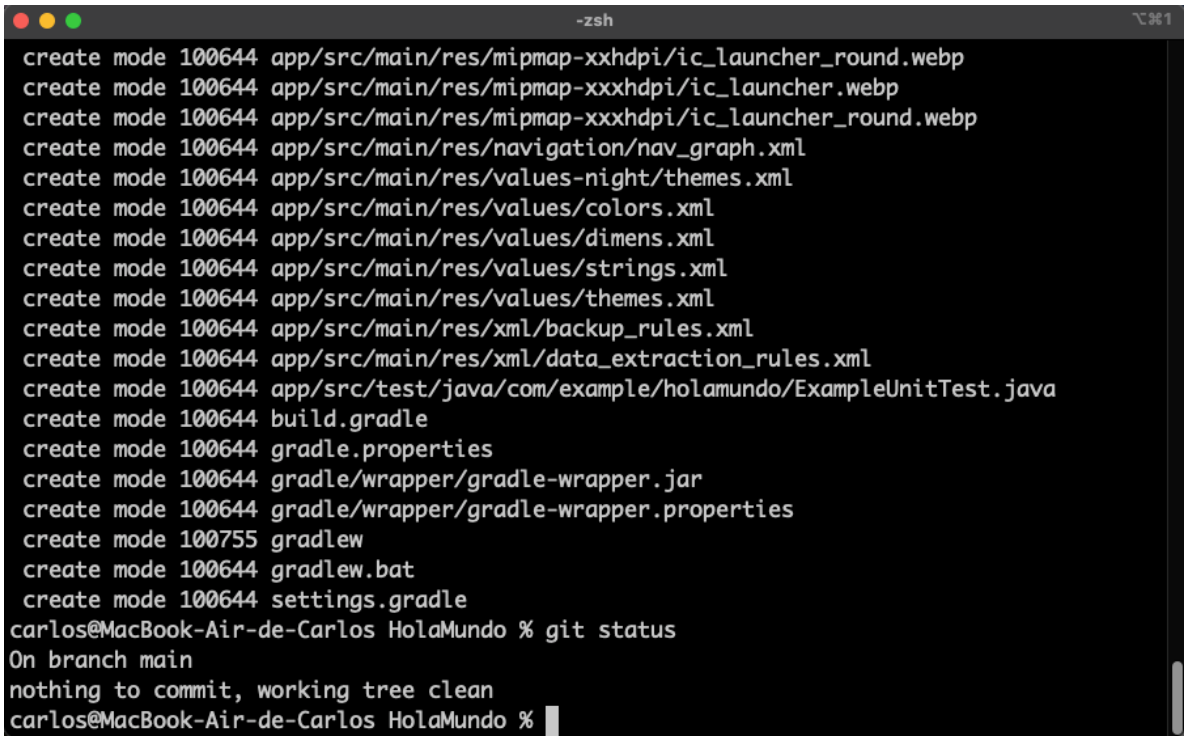

Figura 9 – Estado de nuestro repositorio luego del git commit.

Con los comandos ejecutados hasta el momento hemos completado nuestro flujo local tal como muestra el flujo descrito en la Figura 5.
Ahora nos queda configurar un repositorio Git remoto para completar un flujo típico de desarrollo con Git. Cómo se explicó anteriormente un repositorio remoto es un repositorio Git que se encuentra en un servidor y accedemos a él normalmente por Internet para sincronizarlo con nuestro código local si tenemos los permisos para hacerlo. Veremos más sobre repositorios Git remotos a continuación con GitHub.

Introducción a GitHub
Cómo vimos anteriormente nuestro flujo de trabajo con Git se completa con un repositorio Git remoto y es aquí donde entra en juego GitHub.

Existen varios tipos de repositorios remotos y empresas asociadas a proporcionarlos, las más usadas son: GitHub, Bitbucket y Gitlab. Todos funcionan de forma similar y lo más importante es que soportan Git, permitiéndonos gestionar con los mismos comandos que venimos usando hasta ahora desde nuestros computadores, independiente del servicio que utilicemos.

Como mencionamos anteriormente, GitHub es un gestor de repositorios remotos, lo que quiere decir que podemos almacenar una copia de nuestro código en sus servidores. Así podemos trabajar colaborativamente y respaldar nuestro trabajo.

GitHub es gratis y no tiene restricciones de la cantidad de repositorios que podamos crear. Así que si no tienes tu cuenta de GitHub creada debes crearla para poder continuar con el curso en GitHub.com utilizando el siguiente enlace: https://github.com/.

Configuración de GitHub

Una parte importante de conexión con cualquier servidor es la autenticación, es decir, el procedimiento informático que permite asegurar que un usuario es auténtico es decir es quien dice ser.

Existen varias formas de hacerlo, pero la más fácil y segura es utilizar el protocolo SSH y sus llaves pública - privada.

Las llaves, son un medio por el cual podemos identificar nuestro equipo con un servidor o página específica, incluso sin tener que ingresar una contraseña. Esto funciona mediante dos llaves, una privada y otra pública. La privada se queda en nuestro equipo (no debe ser compartida con nadie) y la pública es la que se ingresa en el lado remoto.

Como posteriormente vamos a subir nuestros cambios a los servidores de GitHub, la idea es identificar nuestro equipo en su servicio, mediante nuestra llave SSH . Esto nos permitirá conectarnos de forma segura y rápida.

Hay que indicar que esta no es la única forma de conectarnos con GitHub, pero es la más utilizada por lo segura y fácil de utilizar una vez configurado.

Entonces nuestro primer paso será revisar si ya tenemos llaves generadas o si necesitamos crearlas.

Obteniendo llaves SSH

La forma de obtener las llaves SSH puede variar dependiendo del Sistema Operativo y su configuración. Para obtener más información en caso de no estar cubierta por este documento se puede visitar el siguiente enlace: https://docs.github.com/es/authentication/connecting-to-github-with-ssh.

El primer paso consiste en verificar en nuestro sistema si ya las tenemos generadas: En el caso de tener Linux o MAC abre la terminal, en el caso de Windows abre git bash.

Una vez al interior de la terminal ingresa el siguiente comando:

```
ls -al ~/.ssh
```

A continuación en la Figura 10 veremos cómo se ve la ejecución de este comando en el caso que ya tengamos llaves previamente creadas.

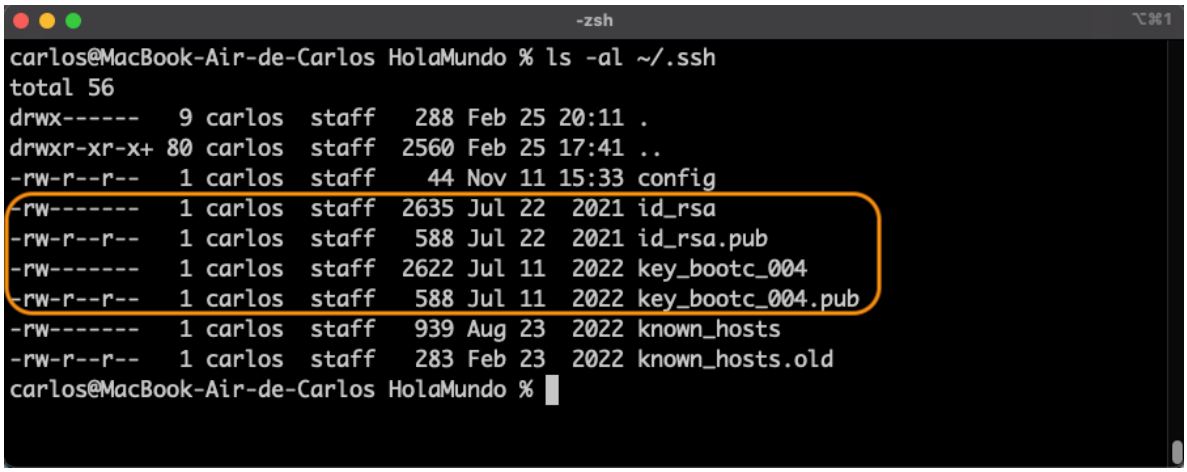

Figura 10 – Llaves SSH, estas aparecen en el recuadro con borde naranjo.

Como se ve en la Figura 10 aparecen varias llaves SSH. Las llaves públicas aparecen con la extensión .pub y el contenido de esta es la que ocuparemos más adelante para subirlos a GitHub.

En caso que al ejecutar el comando anterior aparezca un error porque no existe ~/.ssh es porque aún no tenemos ninguna llave SSH y debemos generarla.

Para generar el par de llaves debemos ejecutar el siguiente comando:

```
ssh-keygen -t rsa -b 4096 -C "Tu_email_cuenta_github@example.com"
```

A todo lo que nos pregunte este comando debemos apretar enter. No debemos agregar ningún password no nada adicional, solo aceptar con enter.
Si seguimos los pasos anteriores se creará el archivo **id_rsa.pub** e **id_rsa** . El .pub es nuestra llave pública, mientras que el archivo id_rsa es nuestra llave privada y debe quedar dentro de nuestro computador.
En la Figura 11 veremos el terminal con el proceso. El comando es levemente distinto, ya que creamos un par de llaves con nombre base id_rsa_android, esto es porque ya teníamos creada la llave antes. Es importante aceptar todo con la tecla enter.

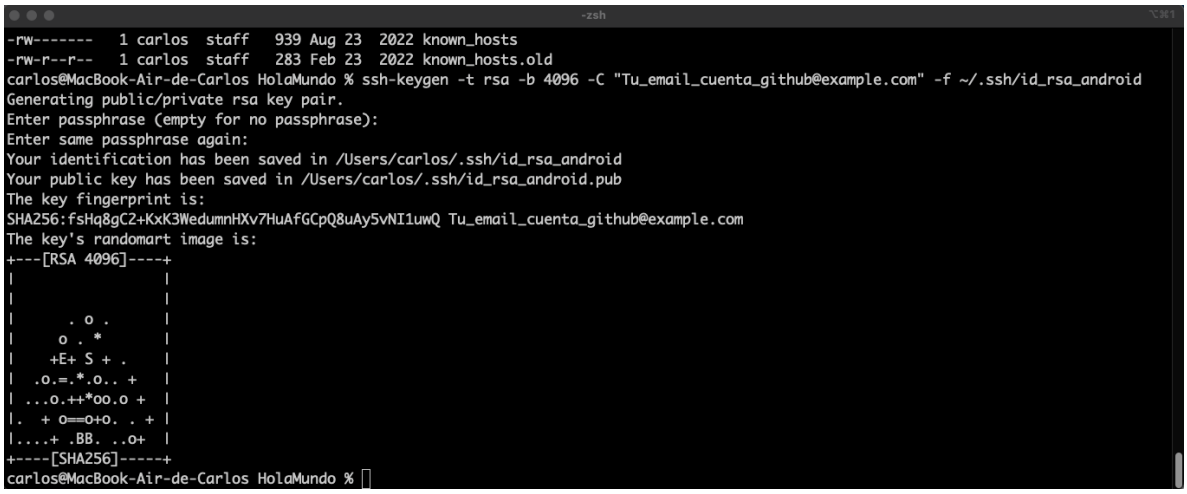

Figura 11 – Creación de llaves SSH.

Ingresando llave pública SSH en GitHub

Debes ingresar con tu cuenta a GitHub. Si no tienes cuenta es el momento de crearla.
Una vez dentro de la cuenta de GitHub, vamos a añadir la clave en nuestra cuenta.

Para poder visualizar el contenido de la llave pública podemos utilizar el comando cat de la consola. Una vez visualizado el contenido lo podemos copiar para luego pegarlo en el sitio de GitHub como se indica unos pasos más adelante. A continuación mostramos un ejemplo del comando cat.

```
cat ~/.ssh/id_rsa.pub
```

En la Figura 12 podemos ver el contenido de la llave pública que creamos en la Figura 11.

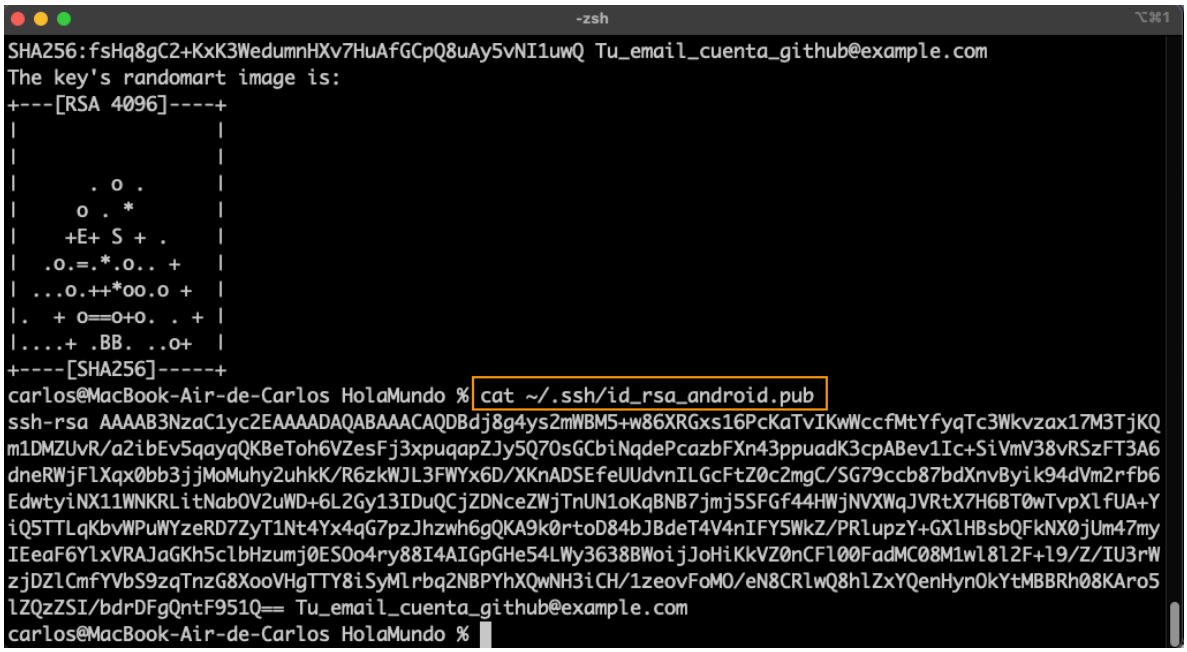

Figura 12 – Comando cat que permite ver el contenido de la llave pública para poder copiarla.

Ahora que tenemos el contenido, dentro del sitio de GitHub vamos a ir a la pestaña de settings.

- Luego seleccionaremos clave SSH y GPG.
- Veremos un botón a la derecha que nos indica añadir una nueva clave pública.
- Copiamos el contenido de la llave publica que obtuvimos con el comando cat.
- Vamos a asignarle un nombre y luego pegaremos la clave copiada.

A continuación en la Figura 13 se muestra el sitio de GitHub con las opciones antes mencionadas para una cuenta en particular.

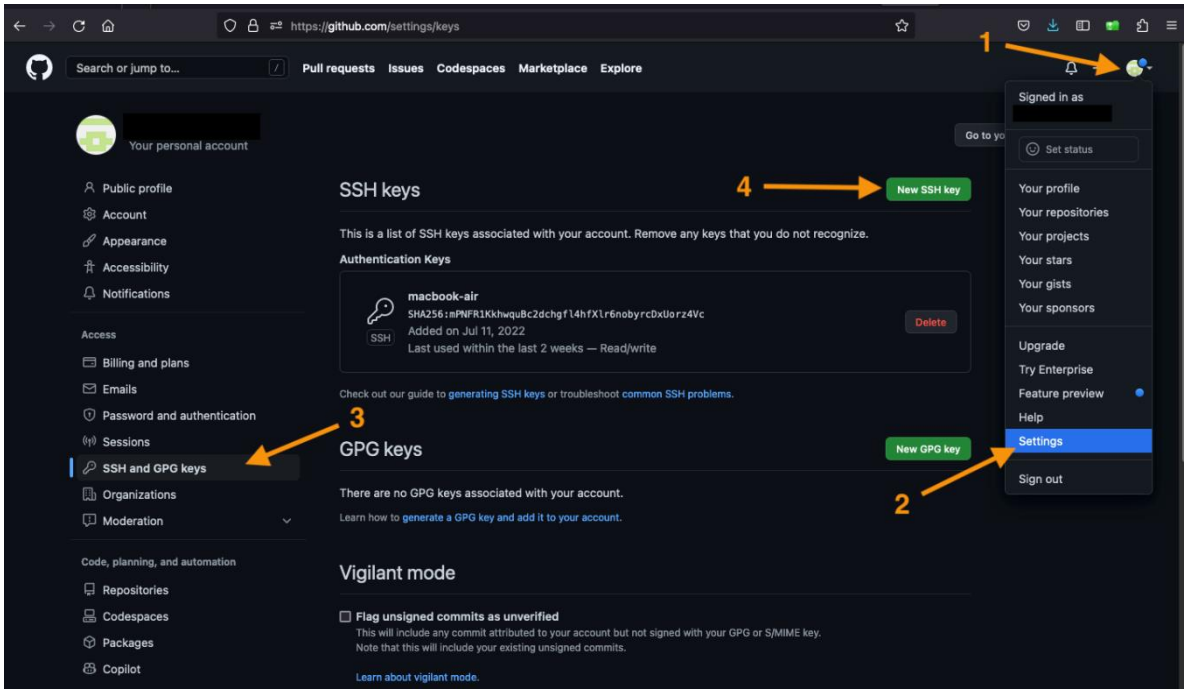

Figura 13 – Agregar llave SSH sitio GitHub. Flechas indican el orden de la navegación.

Para probar el acceso debes ejecutar el siguiente comando:

```
ssh -T git@github.com
```

Lo que debería devolver un mensaje similar al siguiente:

```
Hi nombre-usuario-github! You've successfully authenticated, but
GitHub does not provide shell access.
```

Si no aparece este mensaje deberías repasar el proceso. También puedes visitar el siguiente enlace: https://docs.github.com/es/authentication/connecting-to-github-with-ssh.

Conectando con el repositorio remoto

Ahora llegó el momento de configurar nuestro repositorio remoto. Para ello debemos desde el sitio de GitHub una vez autenticado, desde el sitio, en la barra de la izquierda hay un botón verde con el texto New. Ahí podemos crear nuestro repositorio como se muestra en la Figura 14.

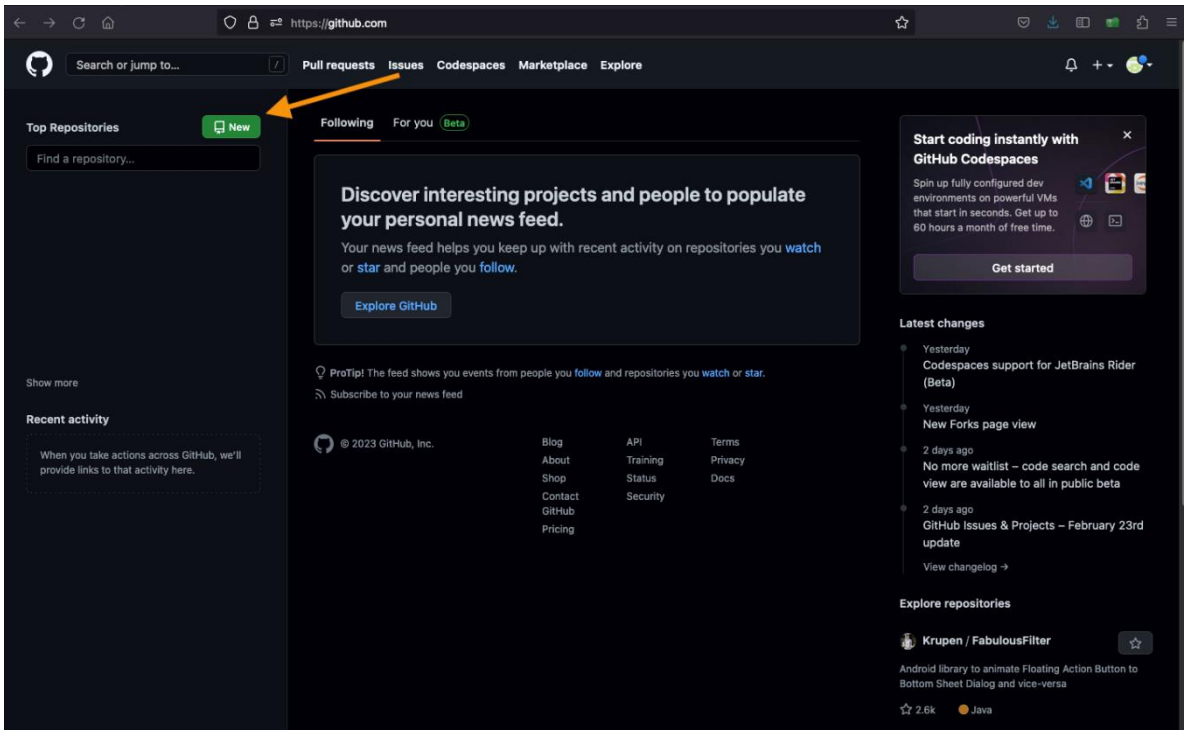

Figura 14 – Dentro del panel del usuario flecha naranja apunta a botón New para crear un repositorio.

Luego se va a desplegar un formulario para crear el nuevo repositorio. Escribir el nombre del repositorio, se puede agregar una descripción, nosotros no lo vamos a hacer en este momento y luego hacemos clic en crear repositorio. Es importante no modificar ninguna de las otras opciones para que nos cree un repositorio vacío. Esto tiene mucha importancia cuando como es nuestro caso ya tenemos un repositorio con un proyecto y código. Así no tenemos conflictos.
En la Figura 15 podemos ver como creamos el repositorio remoto para nuestro proyecto Hola Mundo de Android.

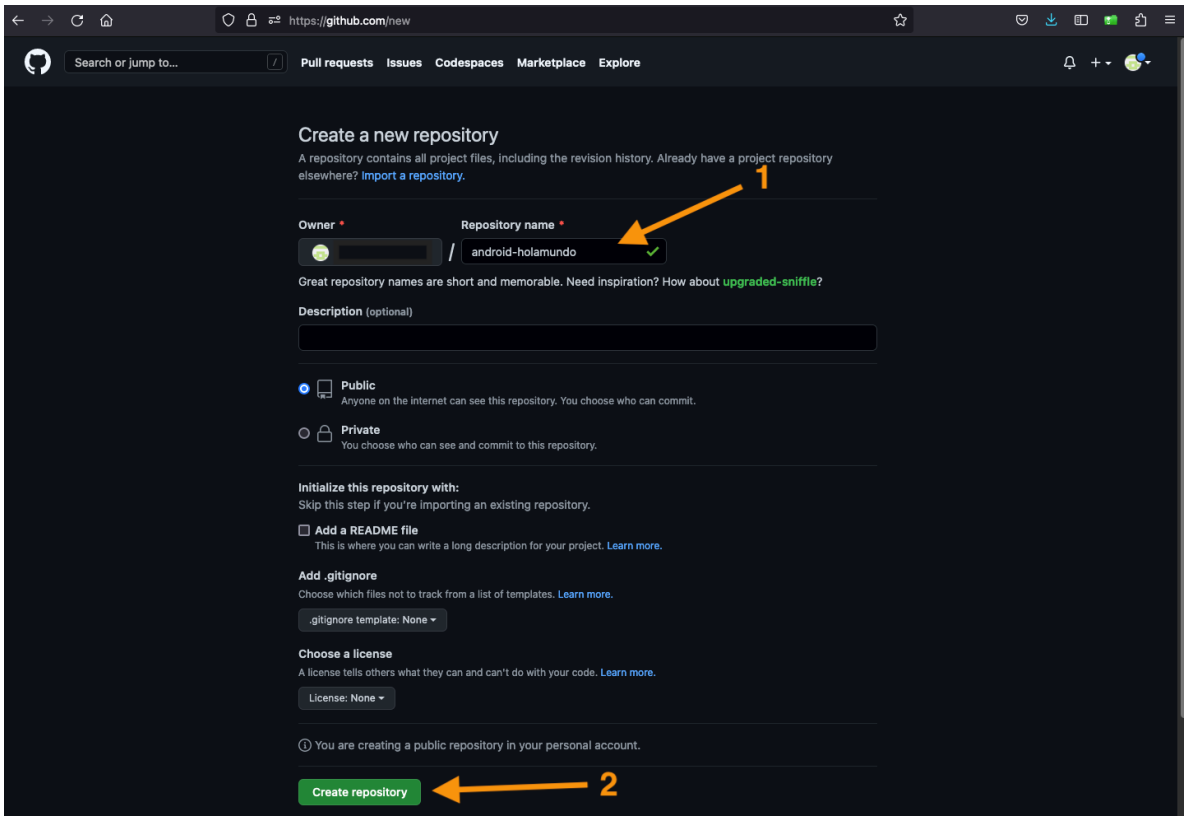

Figura 15 – Creación de repositorio remoto.

A continuación va a aparecer una página con la información de nuestro repositorio y cómo poder crear un repositorio local con los datos de nuestro repositorio remoto en caso de no tener uno. También más abajo aparece la opción cuando ya tenemos un repositorio local que queremos enlazar que es nuestro caso. En la figura 16 podemos ver esta página.

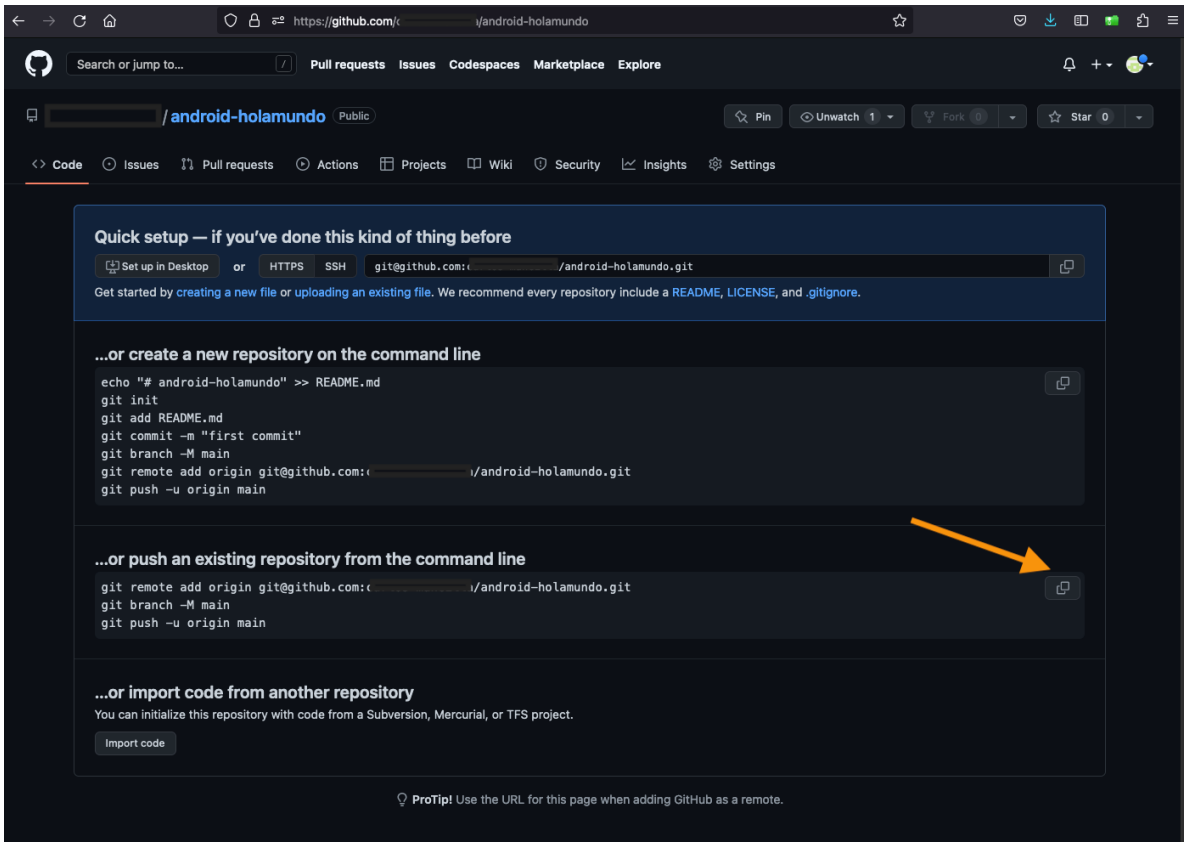

Figura 16 – Página con instrucciones para subir nuestro código. Nosotros vamos a copiar los comandos de la segunda opción como muestra la flecha naranja.

Ahora vamos a copiar los comandos de la segunda opción en nuestro terminal que estamos dentro del repositorio local HolaMundo como hemos estado trabajando. A continuación en la Figura 17 vamos a ver el resultado de la ejecución de estos comandos.

Figura 17 – Agregando repositorio remoto a repositorio local y sincronización entre repositorios.

Lo que hicimos en con los comandos que copiamos de GitHub fue:
1.- Agregar el repositorio remoto a nuestro repositorio local.
2.- Asignar el nombre main al repositorio principal. Esto no era necesario ahora, ya que nuestro repositorio principal ya se llamaba main. Esto se hace porque hace unos años atrás el repositorio principal se llamaba master, pero ahora se llama main.

3.- Subir los cambios al repositorio principal con el comando git push. El flag -u solo se usa en las configuraciones e indica que este es el repositorio remoto principal.

Ahora podemos refrescar la página de GitHub y vamos a ver que ya está arriba nuestro código fuente tal como se muestra en la Figura 18.

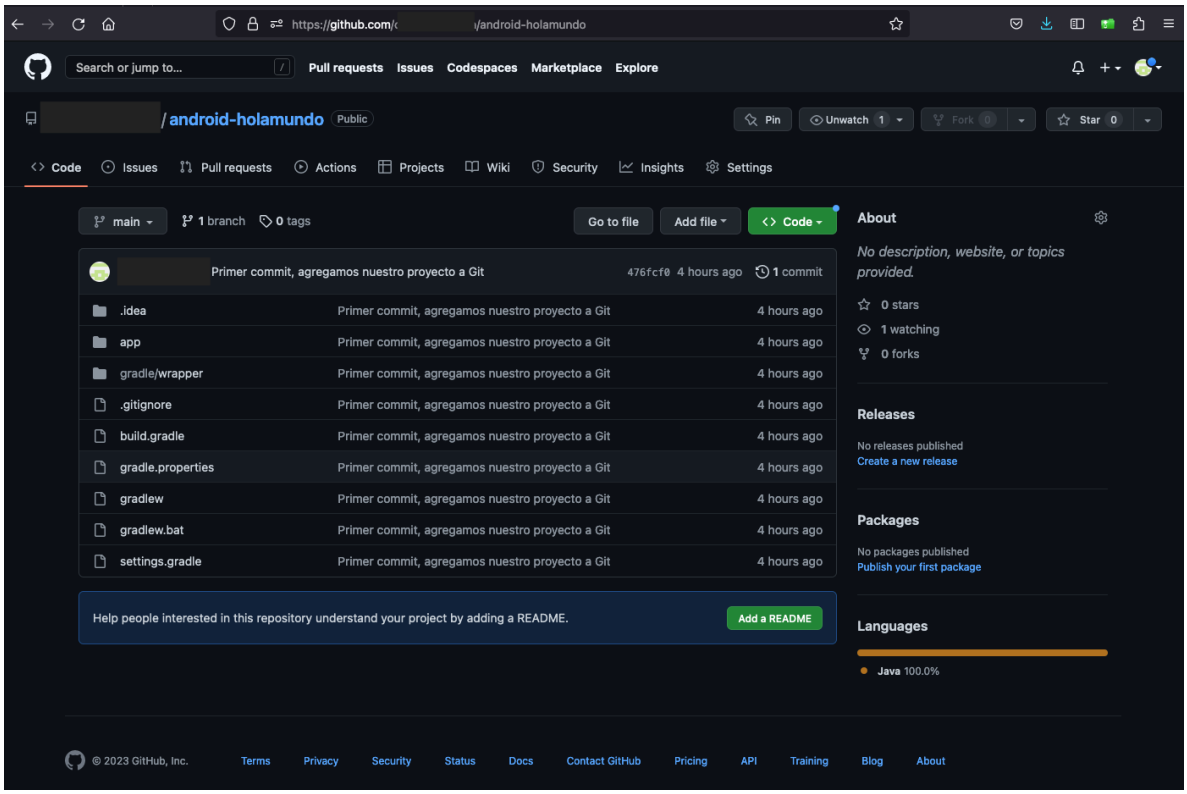

Figura 18 – Repositorio remoto con nuestro código fuente.

Ahora que ya tenemos sincronizados los cambios ya podemos trabajar normalmente con el flujo completo.

Subiendo y bajando cambios hacia y desde GitHub

Para subir los cambios una vez que estén confirmados (ya hecho el commit) se hace con el siguiente comando:

```
git push origin main
```

Donde origin es el nombre de nuestro repositorio remoto y main es la rama a la que estamos subiendo los cambios. Esto en el supuesto que nosotros estemos también en la rama main.
En nuestro caso debería ser así ya que solo tenemos una rama y es la main.

Para bajar cambios desde el repositorio remoto a nuestro repositorio local debemos usar el siguiente comando:

```
git pull origin main
```

Recuerda que el nombre origin corresponderá al repositorio remoto registrado en tu proyecto y main se refiere a la rama main del repositorio remoto.

Descarga de un repositorio de GitHub

Para descargar un repositorio ya creado en GitHub para que podamos trabajar con el existe el comando git clone. A continuación mostramos un ejemplo.

```
git clone [dirección del repositorio remoto]
```

A continuación en la Figura 19 vemos cómo obtener la ruta de un repositorio para clonar.

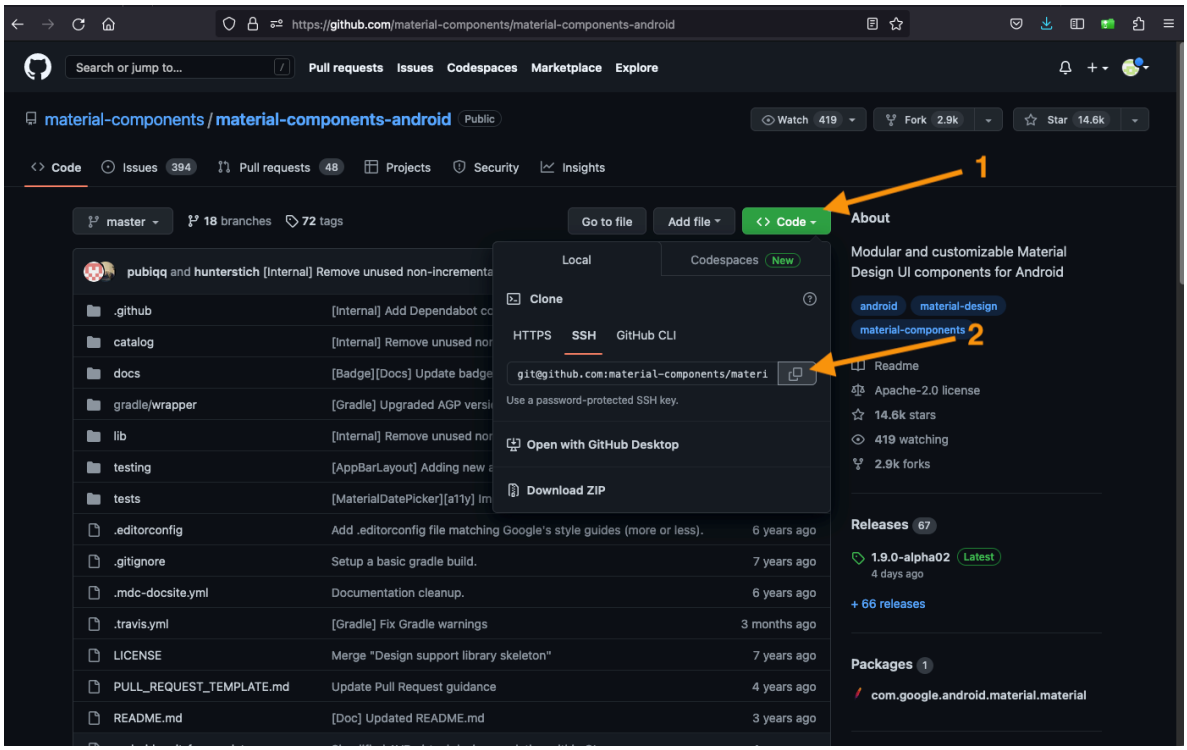

Figura 19 – Obtención de ruta para usar con el comando git clone.

Con esto hemos terminado de revisar el uso de Git y GitHub. Para más información sobre el uso de Git se recomienda visitar el siguiente enlace: https://git-scm.com/book/es/v2 y para más información sobre GitHub se recomienda visitar el siguiente enlace: https://docs.github.com/es.

Git y Android Studio

Cuando nuestro código fuente del proyecto no está gestionado con Git ni ningún otro sistema de gestión de versiones en la esquina inferior izquierda aparece un tab que al hacer clic se despliega una ventana interna con la opción de crear un repositorio local desde Android Studio tal como se muestra en la Figura 20.

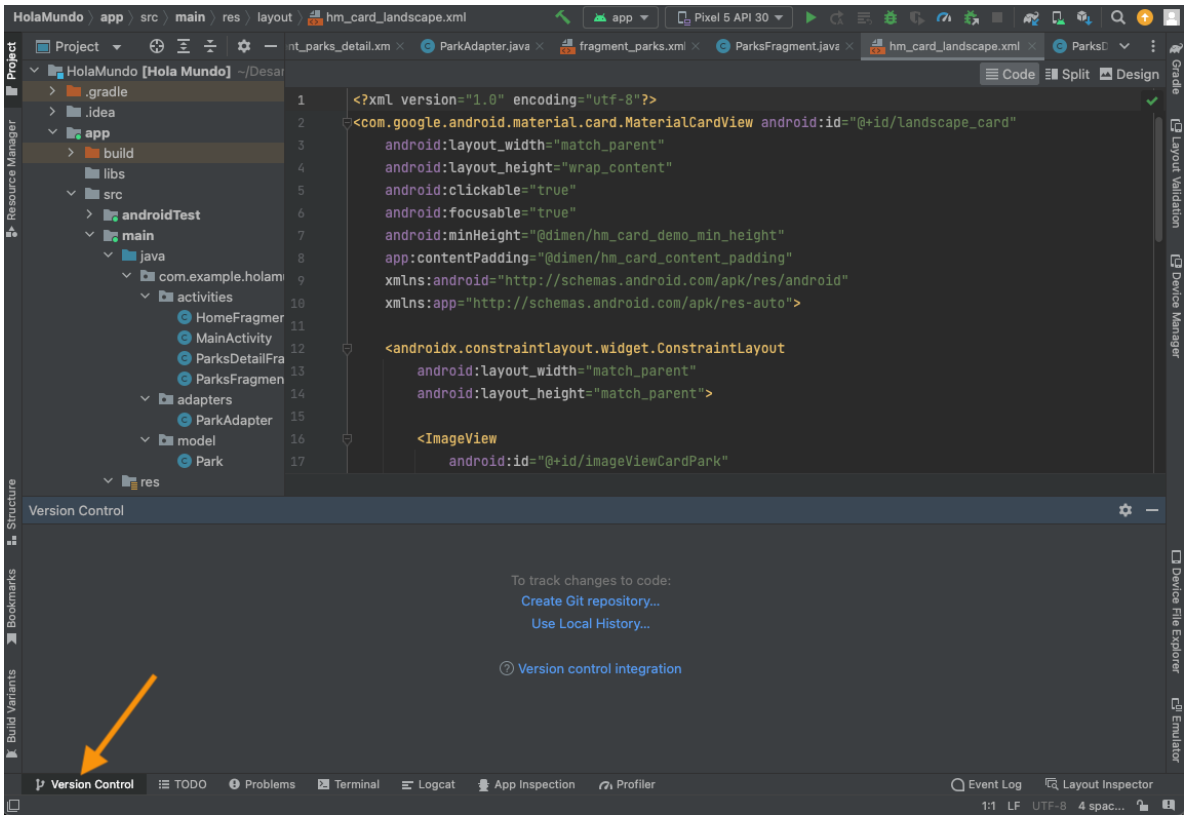

Figura 20 – Proyecto sin control de versiones.

En cambio cuando estamos con Git, en vez de decir Version Control la pestaña cambia de nombre a Git. Al desplegarse nos muestra por un lado los repositorios local y remoto y en otro panel nuestros commits. Desde esta ventana podremos gestionar nuestro repositorio. Esto se muestra en la Figura 21.

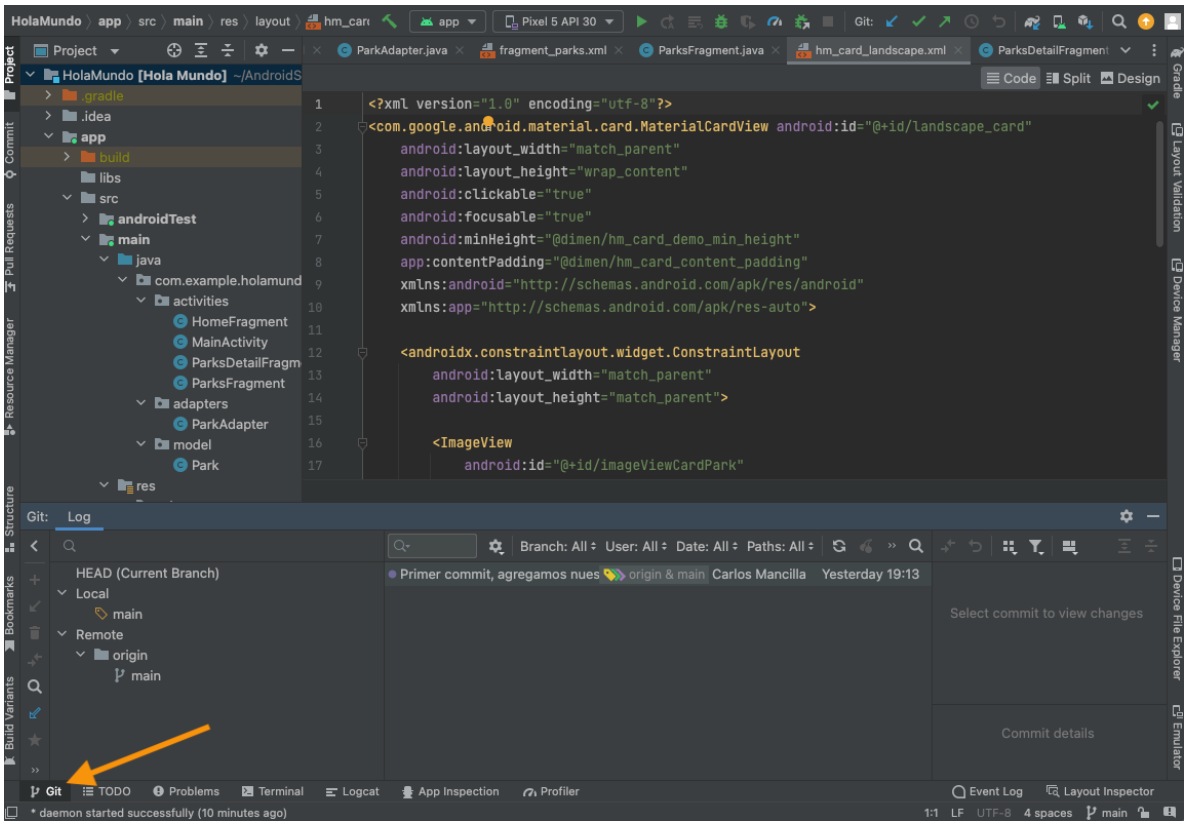

Figura 21 – Tab Git para gestionar desde Android Studio nuestros repositorios.

A continuación vamos a modificar el archivo ParksFragment.java donde solo agregamos un comentario. Luego que el archivo se modificara vimos que en Android Studio las letras del nombre del archivo quedaron de color azul. También en un terminal aparte corrimos el comando git status y mostró en rojo el archivo modificado como también un archivo de configuración nuevo de Android Studio que no estaba en seguimiento. Esto lo podemos ver en la Figura 22.

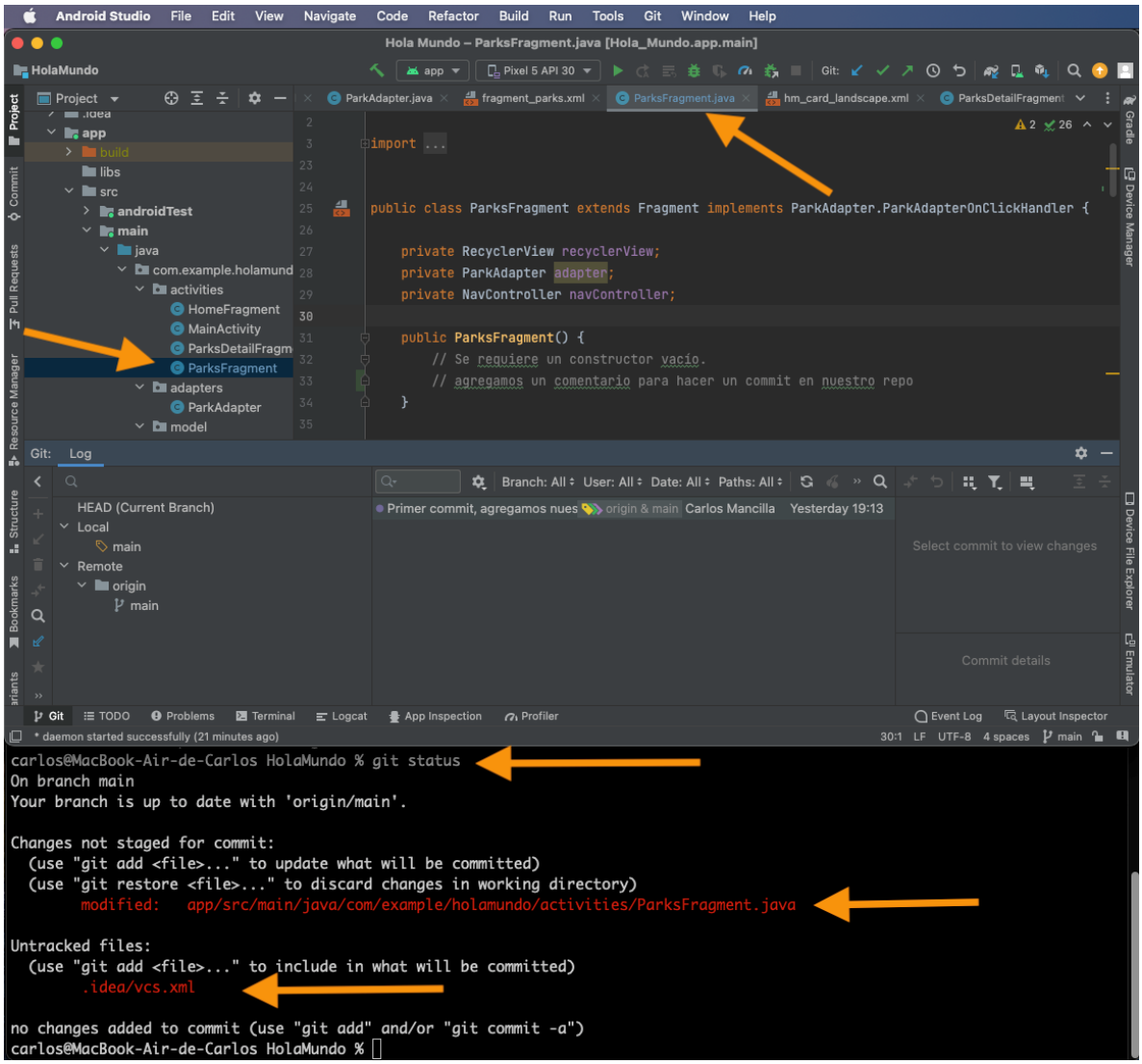

Figura 22 – Modificación archivo en Android Studio visto también en un terminal independiente con el comando git status.

Como vimos en la parte de comandos de Git, para agregar un archivo al seguimiento se usa el comando git add. En este caso Android Studio nos ofrece por ejemplo seleccionado el archivo y luego haciendo clic con el botón derecho un menú de opciones de Git en la que está el add para agregar este archivo al seguimiento como se aprecia en la Figura 23.

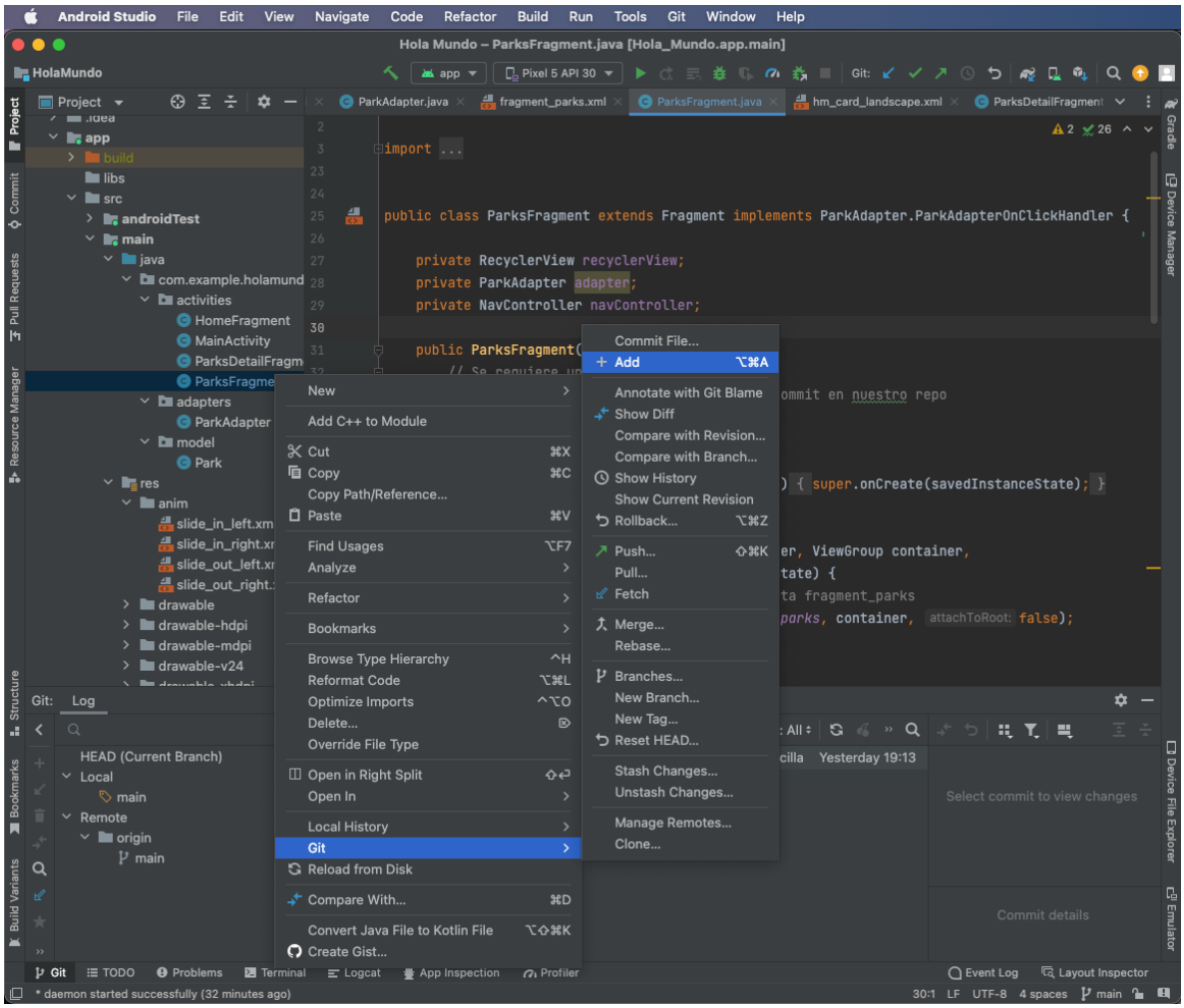

Figura 23 – Agregar a seguimiento de Git archivo ParksFragment.java

Luego de esto vamos a confirmar por un terminal con un comando git status si los archivos fueron agregados, lo que fue efectivamente así y lo vemos en la Figura 24.

```
carlos@MacBook-Air-de-Carlos HolaMundo % git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   .idea/vcs.xml
        modified:   app/src/main/java/com/example/holamundo/activities/ParksFragment.java

carlos@MacBook-Air-de-Carlos HolaMundo %
```

Figura 24 – Comando git status en un terminal para verificar seguimiento de archivos.

Ahora vamos a hacer el commit desde Android Studio. Esto lo podemos hacer desde la barra de herramientas superior donde aparecieron unos botones para el uso de Git. Luego nos aparece una ventana donde aparecen los archivos a confirmar. Verificamos los cambios en una sección de la ventana más abajo. Luego agregamos en comentario del commit y al final hacemos clic en el botón Commit. En la Figura 25 podemos ver cómo vamos a hacer este commit y la secuencia la mostramos con las flechas naranja.

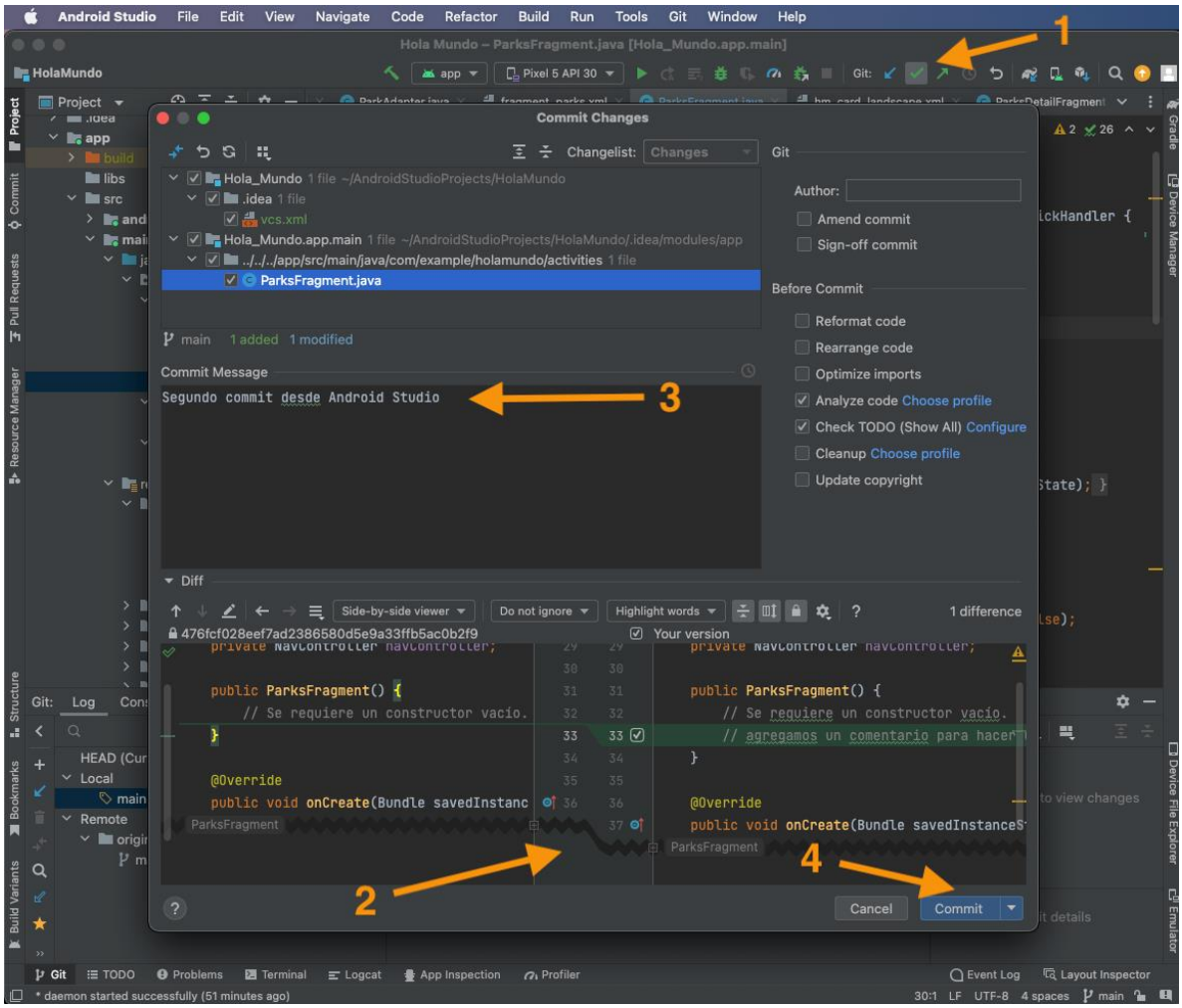

Figura 25 – Commit desde Android Studio.

Luego puede aparecer una ventana para confirmar el commit, uno debe hacer clic en commit nuevamente para que se realice. En la Figura 26 vemos esto realizado. También vamos a ver que en el Branch main del repositorio Local apareció una flecha verde apuntando hacia arriba a la derecha. Esto indica que debemos hacer un git push a nuestro repositorio.

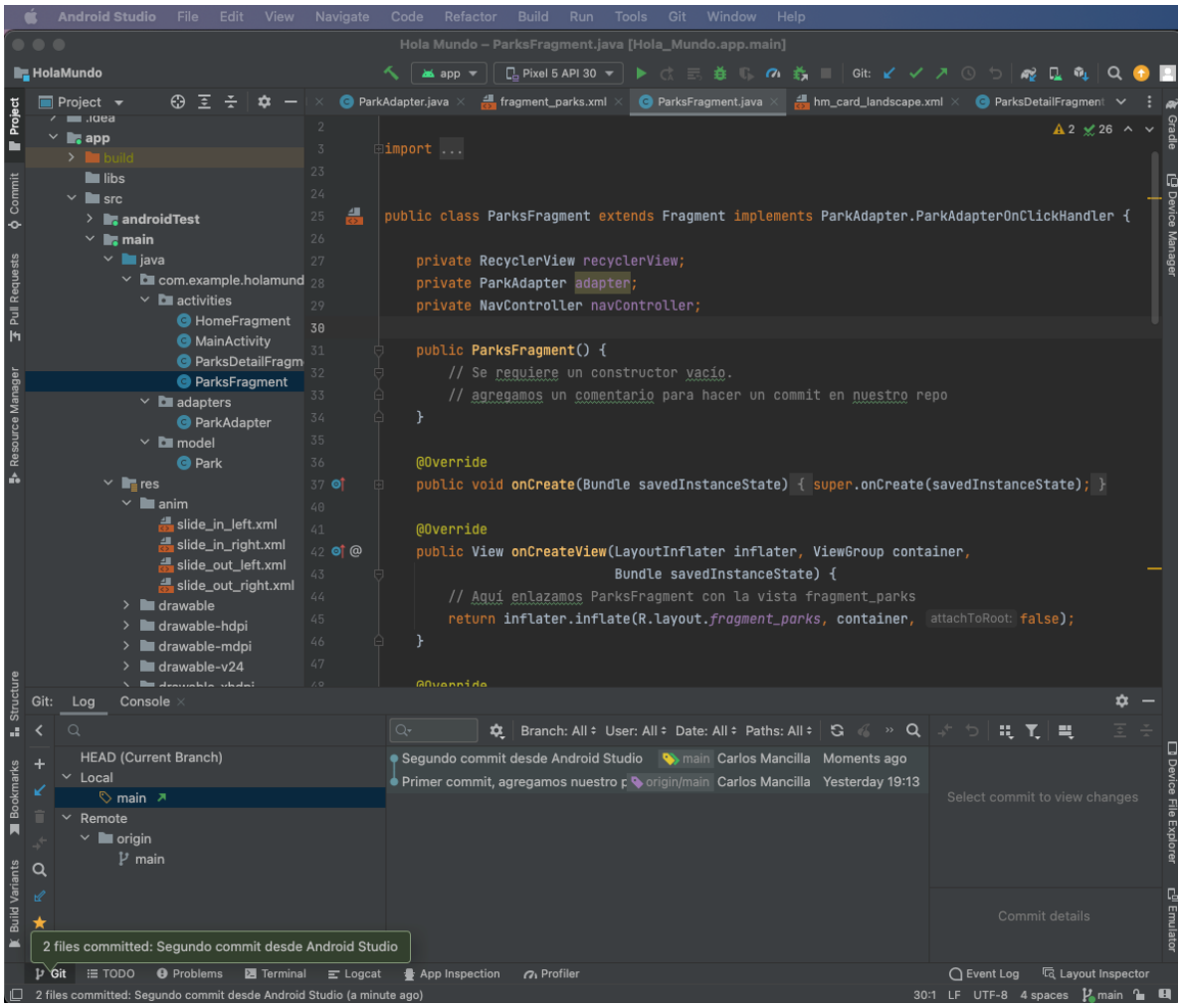

Figura 26 – Vista luego de ejecutar el commit.

A continuación vamos a hacer un git push a nuestro repositorio remoto en Android Studio en la barra de herramientas superior está el botón push, luego de hacer clic en él va a aparecer una ventana para hacer el git push al repositorio remoto. En la Figura 27 vemos la secuencia.

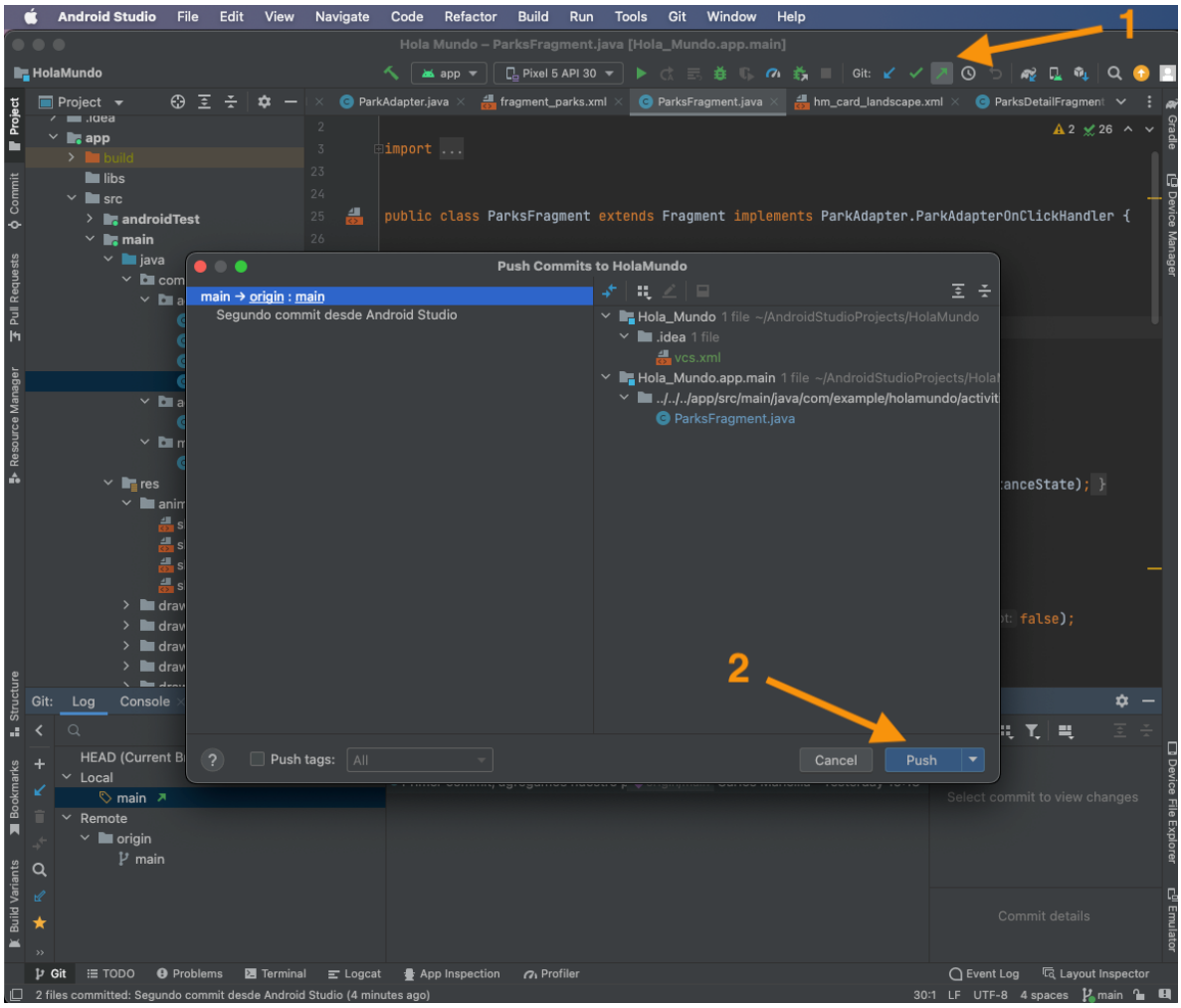

Figura 27 – Haciendo git push desde Android Studio.

Para más información de cómo usar Git desde Android estudio puedes visitar el siguiente enlace: https://www.jetbrains.com/help/idea/2021.3/version-control-integration.html.

Hasta aquí hemos cubierto Git, GitHub y el uso de Git en Android Studio. Ya estás listo para iniciar la gestión de código con Git.

Anexo: Referencias

[1] Sitio oficial Git
Referencia: https://git-scm.com/book/es/v2

[2] Sitio oficial GitHub
Referencia: https://docs.github.com/es/authentication/connecting-to-github-with-ssh

[3] Sitio oficial Android Developers
Referencia: https://developer.android.com/?hl=es-419