# GIT y GITHUB

Git es una herramienta para control de versiones y GitHub es una plataforma para compartir proyectos con otras personas

mSitios de descarga

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

## Configuración de GIT y GitHub

Se puede hacer a través del comando de Windows (cmd) o el Git Bash del programa, al poner el comando **git** aparece todos los comandos del mismo o se puede utilizar la ayuda con ***git --help***

### configuración global:

es la configuración inicial donde se declara los parámetros del usuario que está utilizando.

1. git config --global user.name “nombre” *//nombre del usuario*
2. git config --global user.email “correo electrónico” *//mail del usuario*
3. git config --global color.ui true *//colores identificatorios*

### Configurar las llaves SSH

Es la forma de conectar *git* de nuestra cuenta (pc) con *github* de la nube, a través del protocolo SSH. Es decir, crear un puente entre ambos, para configurar las llaves se utiliza el comando **ssh-keygen -t rsa -b 4096 -C “mi mail”**, crea una llave que se guardara en usuarios/nameUser/.ssh, generando 2 archivos (***id\_rsa*** y ***id\_rsa.pub***).

Estos archivos los vamos a ingresar en la web de mi GitHub *en setting -> SSH and GPG keys -> new SSH keys*, pondremos un título a modo informativo de la computadora y la llave se hará a través del comando de línea (de la consola de git) **clip <~/.ssh/id\_rsa.pub** (sería como el comando copiar de Windows), y se pegará en el cuadro de la llave (key), la cual se generará para enlazar ambas partes.

## Crear repositorio en GitHub

En nuestro usuario de la página web, vamos a ***Repostories***, elegir el botón ***new***, se elige el nombre del repositorio y se puede ingresar una descripción (opcional), elegir el tipo (public/private), si se elige la segunda opción debe ser pago, y si queremos generar un archivo de texto plano con los datos del mismo, checar **Add a README file**, se debe seleccionar salvo que se esté importando un repositorio. Ultimo paso, crear el repositorio.

### Clonar el repositorio a la computadora

Dentro del repositorio de GitHub

1. seleccionar el botón **code**
2. Clonar por medio de SSH (copiar el dato del repo), también se puede hacer por HTTPS o GitHub CLI.

Dentro del Git Bash (***alt***+***F8*** para limpiar la pantalla)

1. Crear una carpeta con ***mkdir*** (conviene crear una carpeta donde guardar todos nuestros trabajos de GitHub) porque la raíz es nuestra carpeta de usuario (ejemplo C:\Users\christian).
2. Entrar en la carpeta con ***cd nameFolder***
3. ejecutar el comando ***git clone git@github.com:Christian/curso-GitHub.git*** (el dato copiado del SSH de GitHub.

Así dentro de la carpeta creada aparecerá nuestro repositorio creado en GitHub.

1. Ingresar a la carpeta del repositorio con ***cd repositorio***, nos daremos cuenta que estamos en GitHub porque en la ruta aparecerá *(main)* o la rama que estemos trabajando.

### Crear repositorio desde cualquier carpeta de Windows

En la carpeta que queremos crear un repositorio con la tecla shift, hacer click derecho y seleccionar “Abrir ventana de comando aquí” o “Git Bash Where” y abrirá la consola de comandos. También se puede utilizar el comando cd por cada nivel de carpeta para llegar hasta la deseada.

Para crear un repositorio se usa el comando git init y creará una carpeta .git (esta carpeta no se tiene que manipular), en caso de que no exista .git, se debe utilizar el comando git init -b main y creará el mismo.

### Archivo .gitignore

Es un archivo que se utiliza para ignorar una serie de archivos (archivos de configuración, de entorno, librerías, etc.) en el proyecto cuando se realiza una confirmación (commit), por ejemplo: .psd, contacto.html, etc.

La sintaxis del archivo .gitignore es la siguiente:

* En cada línea se establece el archivo, directorio o patrón a ignorar.
* Líneas vacías o en blanco no se consideran.
* que comienzan con #, se consideran comentarios.
* que comienzan con !, permite realizar excepciones (niega el patrón).
* que terminan con /, solo se aplicará sobre directorios.
* utilizar \* como comodín.
* patrones a través de expresiones regulares.

### Comandos básicos de Git

**git status** ver el estado del directorio de trabajo y del área del entorno de ensayo. Permite ver los cambios que se han preparados, los que no y los archivos en los que Git no va a realizar seguimiento.

**git commit** **-m “comentario”** confirma (hace una captura) de una instantánea de los cambios preparados en este momento. las opciones son las siguientes: **-m** permite escribir un comentario (debe ser menor a 50 caracteres), el cual va seguido entre comillas dobles.

* **-a** confirma todos los archivos modificados
* **-am** confirma los archivos modificados y agrega un comentario.

Siempre es preferible realizar un ***commit*** cuando hay un gran cambio.

**git push origin main** se utiliza para cargar el contenido del repositorio local (nuestra computadora local) a un repositorio remoto, ***origin*** es el nombre que recibe el repositorio remoto y ***main*** la rama donde vamos a insertar los cambios.

**git add queAgrego** (archivo, imagen, etc), añade cambio del directorio de trabajo en un entorno de ensayo, indica que quiere incluir actualizaciones en un archivo en concreto en la próxima confirmación (commit). tiene opciones:

* **-u** añade al remoto solo aquellos archivos que están siendo motorizados por Git (solo sube esos archivos)
* **-A** añade al índice (remoto) los archivos borrados, modificados y nuevos.

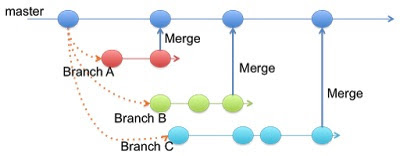
si se quiere agregar más de un archivo, en vez de poner el nombre del mismo ponemos un punto (.)

**git fetch** se utiliza para descargar el contenido de un repositorio remoto a nuestra computadora (solo se clona una vez el repositorio a nuestro sitio local).

**git pull origin main** es a la inversa de push, carga el contenido del repositorio remoto al repositorio local.

## Ramas (branches)

Es una línea de desarrollo distinta a la principal, sirve para trabajar en forma paralela al *trunk* (línea principal) y si dos o más personas trabajan sobre lo mismo no se pisen el trabajo uno a otro, es decir se crea una “sucursal” del proyecto.



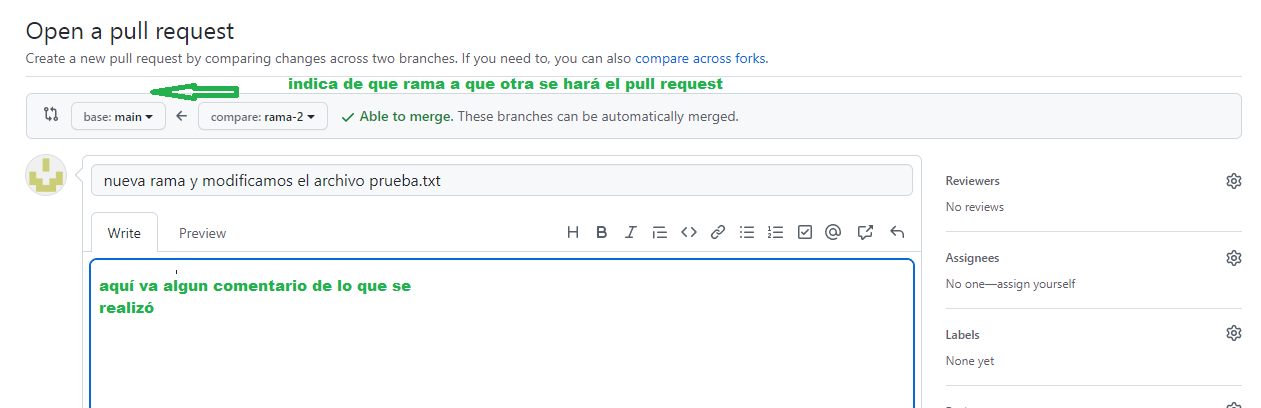
Se utiliza el comando ***git branch nombreRama***, esto solo la crea, ***git branch -d nombreRama*** borra la rama (agregar el parámetro ***-d***). Para trabajar con la rama se utiliza el comando ***git checkout nombreRama*** (con esto en Git se modifica la línea de donde estoy parado cambia *main* por la rama). Si quiero crear una nueva rama y posicionarme sobre ella debo agregar el parámetro **-B** a checkout (ejemplo: *git checkout -B newBranch*)

Cuando se utiliza el comando ***git commit origin nombreRama***, se modifica el *main* por el nombre de la rama, porque estamos trabajando sobre ella y no sobre el principal.

Con esto se puede trabajar con un mismo archivo y tener varias versiones (de cada desarrollador o ramas) o trabajar con distintos archivos, que al final se van a terminar juntando en la rama principal (trunk)

## Pull request

Es una petición para integrar las propuestas (nuestras) o cambios de código en un proyecto, generalmente unir las ramas que lo hace el propietario del repositorio. Los **pull request** permiten no solo llevar de forma más ordenada las tareas en etapa de desarrollo, sino también crear propuestas o cambios para integrar posteriormente al proyecto.



Cuando se realiza el ***pull request*** (solicitud de extracción) en la solapa **Files changed** muestra las diferencias que hay en los archivos que se va a insertar en la rama principal.

Se puede poner un comentario por cada línea modificada, haciendo click en el signo + o -, abriendo un recuadro para tal fin.

En el icono  sirve para agregar una sugerencia.

A veces se puede hacer un ***pull request*** pero un ***merge*** (conflicto sobre un pedazo de código), ejemplo: desarrollador-1 agrega un condicional (if) al código, pero el desarrollador-2 borra ese condicional, entonces GitHub no sabe a quién hacer caso. En la solapa de *pull request* aparecen los conflictos de interés.

Solución, poner los últimos cambios en mi *main* (local) con *git* *fetch*, después un *git* *checkout* a mi rama. Con **git merge** (en *git bash*) va aparecer los conflictos, y lo solucionamos. Paso siguiente hacer un *commit* (indicando que se resolvió el conflicto) y posteriormente un *push* a main.

#### Guía para abrir un Pull Request

* Haz un *fork* del repositorio con el que deseas contribuir haciendo click en el botón de la cabecera del repositorio. Deberás haber hecho *login* en tu cuenta de GitHub.
* Cuando esté completado se abrirá la página con tu copia del repositorio bifurcado, pero de momento solo existe en GitHub. Deberás *clonarlo* en tu equipo local.
* Ahora puedes realizar los cambios editando el código o la documentación en el repositorio clonado a partir del *fork*
* Cuando estés satisfecho con los cambios, sigue el procedimiento normal de trabajo con *commit* y *stage*.
* En el momento en que tus cambios sean completos, y corrijan o mejoren la característica que tenías pensado aportar, ya sea código o documentación, puedes hacer *push* al repositorio.
* En estos momentos tus cambios estén en el repositorio bifurcado, no en el original, de manera que debemos proceder a comunicar los cambios. Si entras en el repositorio, verás una nueva *bandera* indicando que se ha hecho *push* en una nueva rama y que esta rama se puede añadir al repositorio original.
* Haz click en *Compare and Pull Request*, se abrirá una página de discusión. En ella podrás incluir un título y una descripción opcional. Es importante que aportes toda la información necesaria para comprender por qué hiciste los cambios y en qué se basa tu mejora. El mantenedor del proyecto necesita poder determinar si el cambio es útil y/o necesario (recuerda que el mantenedor del proyecto puede tener en mente otras mejoras o cambios que pueden ser generadores de conflicto con tus aportaciones).
* Envía el *pull request*

### Unir (Merge)

Permite tomas las líneas independientes, creadas en las ramas (branches) e integrar en una sola rama (generalmente la principal). Hace una fusión de las ramas. el comando a utilizar es ***git merge ramaAgregar***.

## Historial de repositorios

Se puede ver a través del comando **git log** el historial de un repositorio ordenado cronológicamente. Este comando es muy versátil y muestra la historia del repositorio en distintos formatos, dependiendo de los parámetros que se le pasen.

Algunos de los comandos que se utilizan en el historial del repositorio son:

* **git log** trae todos los *commit* (registros de confirmación) realizados.
* **git log --oneline** trae todos los *commit* en una única línea.
* **git log --stat** muestra los archivos que se modificaron en cada confirmación, número de líneas añadidas o eliminada y un resumen de lo anterior.
* **git log --patch** muestra los archivos modificado, la ubicación de líneas modificadas y los cambios específicos.
* **git log [núm. de SHA]** trae un *commit* especifico (ejemplo: *git log 7752b22*).
* **git log --graph** muestra como un gráfico, este comando tiene opciones **--decorate** () y **--all** ()
* **git reset --hard** regresa al último *commit* realizado (resetea todos los archivos implicados).
* **git rm --cached nombreDelArchivo** eliminar archivos sin eliminar su historial del sistema de versiones. Si se desea recuperar ese archivo, ir al último *commit* antes de haber borrado el mismo.

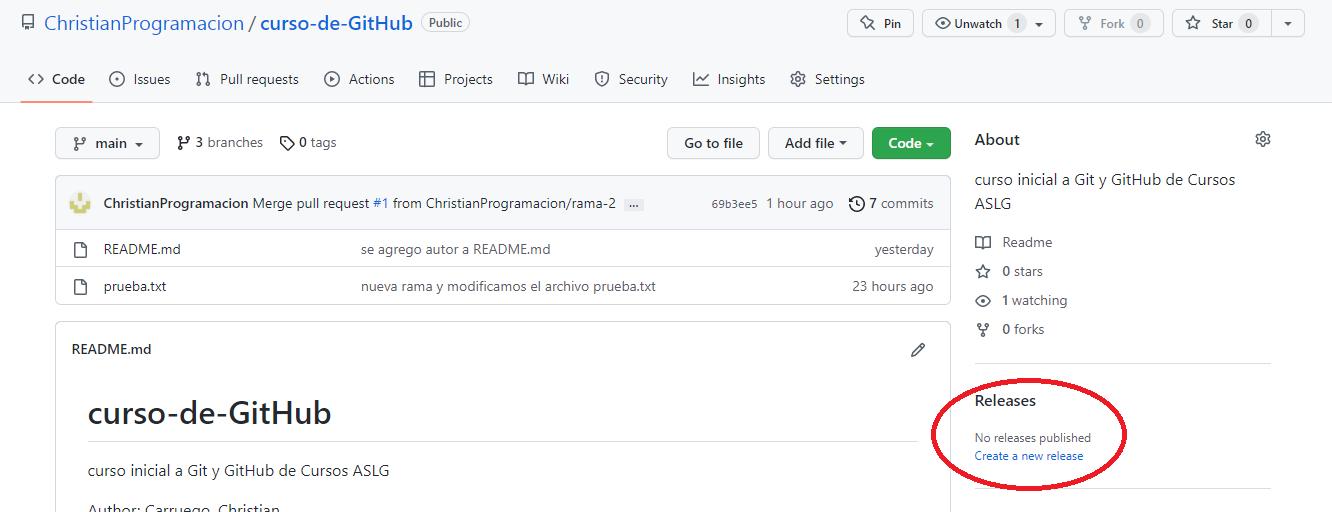
### Comparación de modificaciones

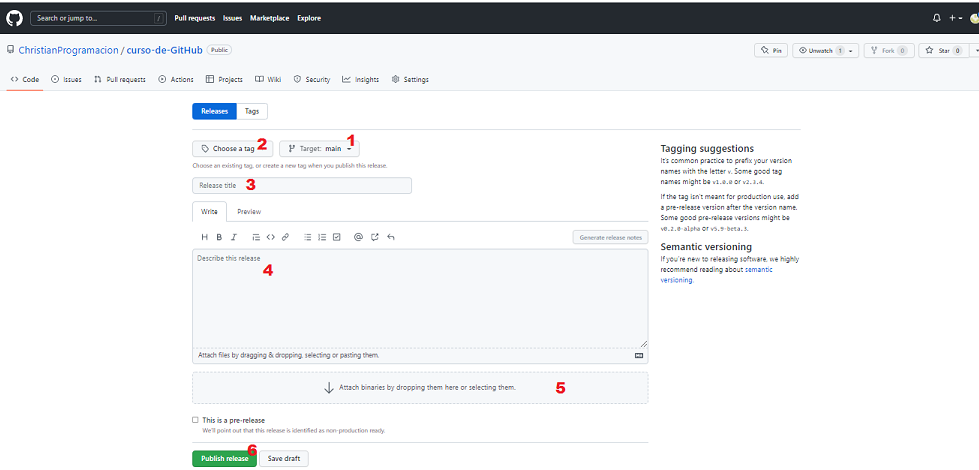
el comando **git diff** muestra las diferencias en los orígenes de datos (confirmaciones, ramas, archivos, etc.). Se suele utilizar junto a *git status* y *git log* para analizar el estado actual de un repositorio de Git.

* **git diff** muestra las diferencias de los archivos guardados con las modificaciones pendientes de confirmar (commit).
* **git diff identificadorCommit nombreArchivo** muestra las diferencias entre los dos commits.

## Release

Se utiliza en GitHub, sirve para liberar, es como cuando ya está listo el proyecto, es como cuando en etapa de desarrollo, listo para la etapa de producción. Es como publicar el proyecto.





1. indica de cual branch (rama) va a salir.
2. elegir una etiqueta (ejemplo: v1.0 -no debe tener espacio en la etiqueta-)
3. título del reléase (descripción).
4. descripción del proyecto.
5. agregar un archivo binario, por ejemplo, un ejecutable (.exe, .apk, etc.), en caso que se pudiese.
6. generar el Release
7. Se genera el proyecto, creando archivos para su descarga.

## GitHub Issue

es una nota en un repositorio que trata de llamar la atención sobre un problema. Puede ser un error a corregir, una petición para añadir una nueva opción o característica, una pregunta para aclarar algún tema que no está correctamente aclarado o muchas otras cosas diferentes.

Por cada ***Issues*** es conveniente que se abra un *branch* para que se resuelva el error.

#### Guía para abrir un Issue

* Asegúrate de que tu petición no se ha hecho ya con anterioridad. Evita duplicar peticiones.
* Si nadie ha hecho un issue con tu petición. Haz click en la pestaña **Issues** de la barra lateral.
* Haz click en el botón de New Issue
* Dale un título suficientemente descriptivo y un texto aclaratorio de tu petición. Cuanto más claro seas, mejor.
  + Explica el problema, cuál es el resultado esperado y cual es el realmente obtenido.
  + Detalla uno a uno los pasos para repetir el problema. Incluye detalles como el sistema operativo, el navegador utilizado, la librería, o las versiones del software implicados en el mismo.
  + Pega los mensajes de error o el contenido de los logs en el mismo pull request o en un Gist. En caso de pegarlo en el propio texto abre y cierra con tres acentos franceses así: ``` que permiten que el código se muestre correctamente.
* Haz click en Submit New Issue cuando hayas terminado. Ahora este Issue tiene una URL permanente que puedes referenciar para compartir.

## GitHub Profile

Es para personalizar el perfil del usuario, es como crear una bibliografía del usuario. Se puede crear un archivo de README a la raíz de un repositorio públic con el mismo nombre que el nombre del perfil.

Las personas que lo visiten podrán ver la siguiente información:

* ver una cronología de tu actividad de colaboración, como las propuestas y las solicitudes de extracción que has abierto, las confirmaciones que has realizado y las solicitudes de extracción que has revisado. Puedes elegir mostrar solo las contribuciones públicas o también incluir las contribuciones privadas, anonimizadas.
* Repositorios y gists que te pertenezcan o en los que contribuyas. Puedes exhibir lo mejor de tu trabajo si fijas los repositorios y gists en tu perfil.
* Los repositorios que hayas marcado como favoritos y organizado en listas. Una descripción general de tu actividad en organizaciones, repositorios y equipos en los que eres más activo.
* Las insignias y los logros que resaltan tu actividad y muestran si utilizas GitHub Pro o si participas en programas como el de la Bóveda de Código del Ártico, Patrocinadores de GitHub o el Programa de Desarrollador de GitHub.