GIT y GITHUB

Hoy en día es imprescindible aprender la tecnología GIT .En todos los proyectos se utiliza esta tecnología de repositorio (lugar común donde cohexisten las diferentes versiones de un programa).

Antiguamente nosotros usábamos un diario para ir anotando las cosas importantes de nuestra vida , y todo esto en un línea de tiempo, por ejemplo mi primer día en el colegio , mi primer día en la universidad , alguna fecha de cumpleaños importante , etc…

Y donde en un momento dado yo pudiese consultar que es lo que hice en una determinada fecha.

Este historial también se puede llevar en el mundo del software.

Un software no es algo que se crea en un día, tiene una vida, pasa por versiones la 1.0 , la 2.0, la 2.1 , etc…,incorporamos nuevas personas ,corregimos errores, añadimos nuevas funciones, etc…

Es necesario llevar también un diario donde se registre la vida de un proyecto, de una aplicación. ¿Qué se hizo la noche del 5 de septiembre de 2005 en el proyecto?

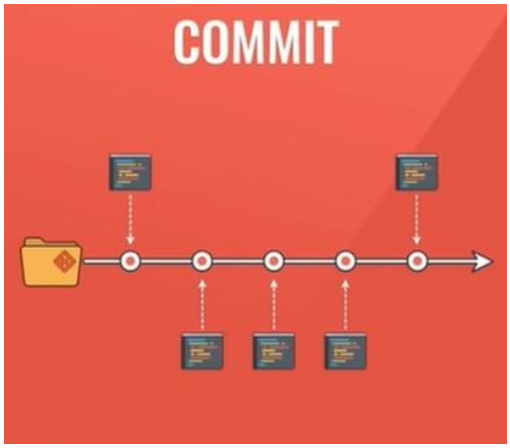
Para ello disponemos de lo que denominamos un sistema de control de versiones (VCS).Para registrar la vida de un proyecto. En esencia es lo mismo que el diario, solo que no es un cuaderno **es un software, y además permite trabajar en grupo.**

Cada desarrollador que está en este proyecto puede agregar a ese historial los cambios y correcciones, nuevas características que está añadiendo, por lo tanto el VCS , tiene un **historial** de todo lo que se ha hecho y **quién** lo ha hecho. Esto es Git.

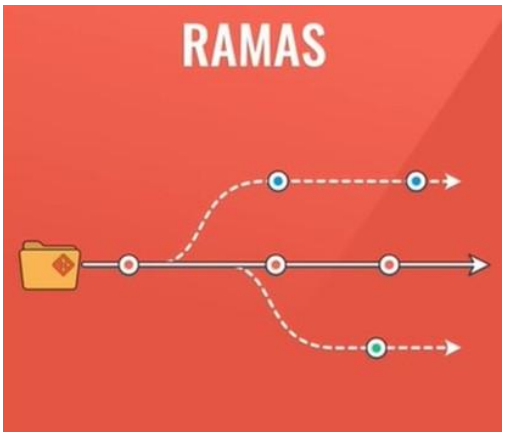
Vamos a echar un vistazo a la terminología más importante que vamos a usar en esta tecnología.

Git es un software de control de versiones que permite registrar todo el historial de cambios de un proyecto.

Un **repositorio** es todo proyecto que está siendo seguido por Git. Ya tiene un historial de Git en el que se están registrando sus cambios.



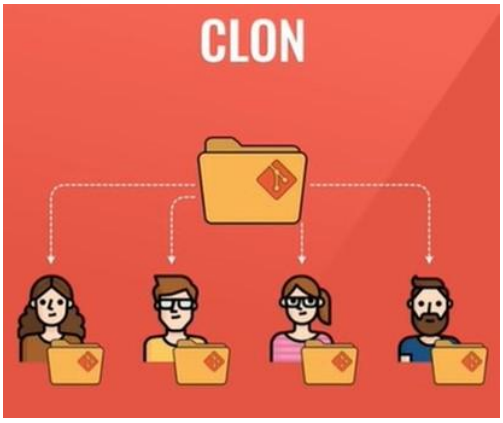
Un **commit** es cada uno de los cambios registrados en el historial de Git. Cada desarrollador debe mandar esos cambios a Git , no se hace de manera automática.



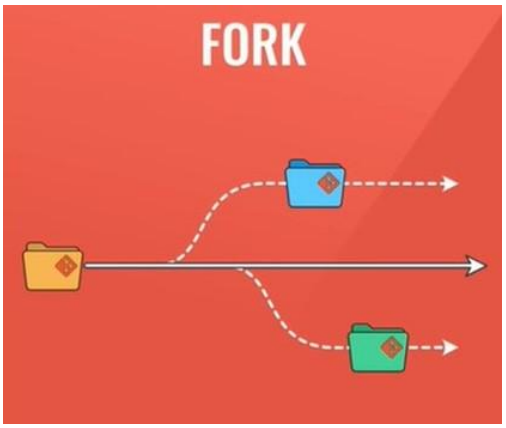
Las **ramas** son bifurcaciones , nuevos caminos que toma el proyecto. En Git todo se trabaja por ramas. Hay una rama que es la principal que suele llamarse **Máster**, donde está el proyecto que 'sale' al público, la versión en producción.

Cada vez que se quiere trabajar en una característica nueva o corregir algo se saca una rama, de tal manera que puedes trabajar en un ambiente aislado, es una copia del proyecto pero separada.

Trabajamos en ese ambiente aislado de tal manera que si algo se rompe no se compromete el proyecto. Si al final todo va bien esa rama la unificas con el proyecto principal y si no la puedes eliminar sin ningún problema.



Un **clon** es una copia exacta del repositorio. Cuando un programador se integra a un equipo de trabajo lo primero que debe de hacer es clonar el repositorio en su equipo local. Cada uno de los desarrolladores tiene un **clon** del repositorio en su equipo local.



Un **fork** a diferencia de un clon y de una rama es un proyecto completamente diferente que se crea a partir de otro. Por ejemplo las distribuciones de Linux, todas se basan en el mismo Kernel pero a partir de ahí todas toman su diferente camino. Un fork es un proyecto que se basó en otro pero que es diferente.

El creador de Git fue Linus Torvalds el creador del kernel de Linux. En 2005 Torvalds trabajaba con una comunidad de desarrolladores muy extensa y usaban un sistema de control de versiones llamado BitKeeper.

BitKeeper es un sistema de control de versiones distribuido para el código fuente de los programas producidos a partir de BitMover Inc. y se distribuye bajo la licencia Apache 2 a partir de la versión 7.2ce, versiones anteriores están solo bajo licencias propietarias.

BitKeeper es producido por Bitmover Inc., una compañía privada con sede en Campbell (California), propiedad del CEO Larry McVoy, que diseñó previamente TeamWare. BitKeeper compite principalmente con otros sistemas tales como **Git** y **Mercurial.**

Torvalds decidió crear su propio sistema de control de versiones (Git) debido a que la licencia de BitKeeper dejó de ser gratuita y quería seguir con la filosofía de Linux es decir licencia open source.

¿Por que Git se ha convertido en el sistema de control de versiones más usado en el mundo habiendo otros? Vamos a echar un vistazo a las principales características de este sistema.

Git es distribuido. Cada uno de los desarrolladores tiene una copia idéntica del repositorio en su equipo local. No necesitan conectarse a un servidor central para trabajar, tampoco necesitan conexión a internet para trabajar en cada momento, además es más rápido. Otra cosa interesante es cada desarrollador tiene un backup del proyecto, si algo pasara si algo fallara cada uno tiene una copia del proyecto lo cual lo pone a salvo.

Como vimos antes las ramas son nuevas bifurcaciones, nuevos caminos que toma el proyecto para no comprometer la rama principal, para mantener la integridad en la rama principal.

Las ramas se utilizan para nuevas características, para hacer pruebas, para corregir cosas , etc… Pero esas ramas luego tienen que integrarse a la rama principal. Sale una nueva rama, el programador está trabajando en una nueva característica y una vez finaliza esa rama tiene que integrarse con la rama principal (**Merge**).

En esa fase pueden suceder bastantes problemas y conflictos.

Git agrega un checksum una suma de comprobación a cada uno de los archivos, a cada uno de los commits de tal manera que hay una seguridad total de que cada uno de los desarrolladores tenga los mismos datos que los demás, si no habría datos corruptos y el sistema podría entrar en problemas.

Vamos a echar un vistazo al flujo de más trabajo básico que existe en Git.

Siempre partimos de nuestra área de trabajo que también es llamada **working directory**. Git init , desde la terminal cuando creamos un repositorio desde cero**. Git clone y la URL del repositorio**, clonamos un repositorio y tenemos una copia en local.

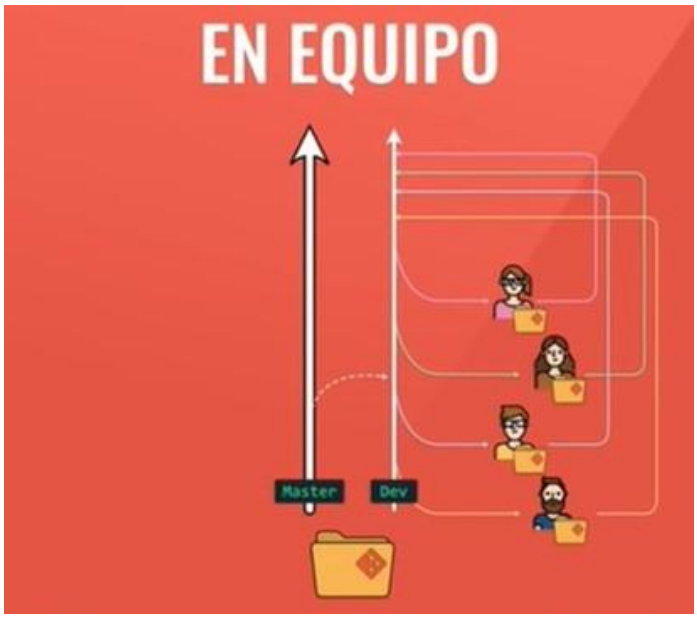
Empezamos a trabajar y mandamos los cambios a través de **commit.**

Hay que enviarlos manualmente , cada commit suele representar una funcionalidad específica, por ejemplo un cambio en la interfaz, la corrección de un problema, y el programador **debe escribir** de qué se trata este commit.

Los cambios **no** van directamente al repositorio , sino que van a una etapa intermedia, el **staging area** o área de preparación **.**

Cuando estoy realizando cambios en un archivo o varios archivos todos ellos los voy mandando al staging área con el comando **git add** , no los mando directamente al repositorio hasta que no estemos seguro de que se hicieron todos los cambios necesarios en los archivos, mientras tanto están en espera en el staging área.

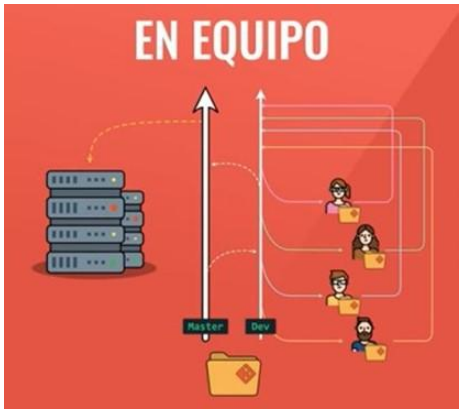
Una vez realizados todos los cambios ya lo envío al repositorio con el comando **git commit** y le mando un **mensaje** indicando que lo estoy enviando por ejemplo porque estoy corrigiendo un determinado error.



Cuando trabajas en equipo la forma básica(hay otras formas más complejas) es tener dos ramas la rama **Máster** que es la rama principal , esa no la debemos tocar, y la rama **Dev** que es la rama con la que vamos a trabajar.

Máster está prohibida para todo el mundo salvo para los líderes de proyecto. Desde la rama Dev cada uno de los desarrolladores va a sacar sus propias ramas para trabajar en lo que le toque a cada uno .

Las ramas deben volver e integrarse de nuevo a Dev (**Merge**) **aquí podríamos tener algún conflicto y los líderes del proyecto podrían devolvérnoslo para que lo volvamos a revisar de nuevo**. Hay sistemas más avanzados y automatizados que detectan los conflictos y no nos dejan hacer el Merge avisándonos de que se produjo algo que no fue bien. **Lo importante es que todo esto se hace en Dev y no en Máster.**



Si todo ha ido bien integramos Dev a Máster y es el momento de enviar todo a producción.

Vamos a ver cuáles son las herramientas más importantes para trabajar con Git.

La herramienta más importante es la terminal o línea de comandos. A pesar de que existen muchos programas con interfaz gráfica que agilizan el trabajo pero cuando ya sabemos Git. **No se aprende Git con una interfaz grafica**.

Vamos a aprender Git con la línea de comandos utilizando todos los comandos Git como:

init

add

commit

merge

push

pull

branch

status

etc…

Una vez ya dominamos la terminal podemos aprender a manejar un programa de interfaz gráfica que va a acelerar nuestro flujo de trabajo del día a día.

Hasta ahora hemos visto repositorios en local , pero como hacemos para equipos distribuidos alrededor del mundo? Lo que usamos son repositorios en la nube. Existen empresas que nos dan toda un plataforma para usar Git en la nube. La rama local sigue siendo la rama Máster y la rama remota la llamamos **Origin.**

Esta rama Origin puede apuntar a GitHub , Bitbucket o GitLab (estas son las 3 más importantes). Git es la tecnología y GitHub es la empresa que nos proporciona soporte de repositorios en la nube. GitLab ha entrado con mucha fuerza.

Git está integrado en muchos IDES y editores de código. VS Code de Microsoft va perfecto para GitHub ya que Microsoft compró GitHub. Intellij idea tiene una integración muy buena con Git , te proporciona un cliente en modo terminal y un cliente gráfico. Atom es el proyecto de GitHub que se desinfló un poco después de que Microsoft comprase GitHub pero sigue existiendo.

⚠ **Microsoft compró LinkdIn**

A diferencia de Google y Amazon, la presencia de Microsoft en el software corporativo ya es amplia, y algunas de las funciones de LinkedIn combinarán muy bien con sus productos de recursos humanos y planeación financiera. Microsoft también tiene como objetivo aumentar el software corporativo de Skype, la empresa de comunicaciones en línea que compró por 8,5 mil millones de dólares en 2011 y que junto a LinkedIn podría ser útil no solo para reclutar empleados, sino para organizar equipos.

Gran parte de la propuesta de LinkedIn ha sido su papel central en un futuro en el que habrá menos empleos para toda la vida y muchas más personas que trabajen de manera independiente, ofrezcan consultoría y presten otros servicios independientes en todo el mundo.

LinkedIn ya cuenta con datos sobre el tipo de trabajo que buscan y realizan las personas, lo que probablemente sea su mayor tesoro, además de tener uno de los equipos de científicos más prestigiosos en el sector de la tecnología. Esa información y esos científicos ahora trabajarán con otros colegas en áreas como la búsqueda en línea, videojuegos e inteligencia artificial.

Los datos no solo son importantes por naturaleza en la actualidad; son activos clave para mejorar el desempeño de los sistemas de inteligencia artificial. Los conjuntos de *big data* son ideales para obtener mejores resultados (también es la razón por la que muchas personas que trabajan en tecnología han comenzado a preocuparse por los datos que tienen las empresas más grandes).

Con la compra de **LinkedIn,** Microsoft no solo puede mejorar el desempeño de sus propios algoritmos, sino que toma el control de un gran recurso con potencial de datos y lo retira del mercado para Amazon y Google.

Version control system (VCS) for tracking changes in computer files

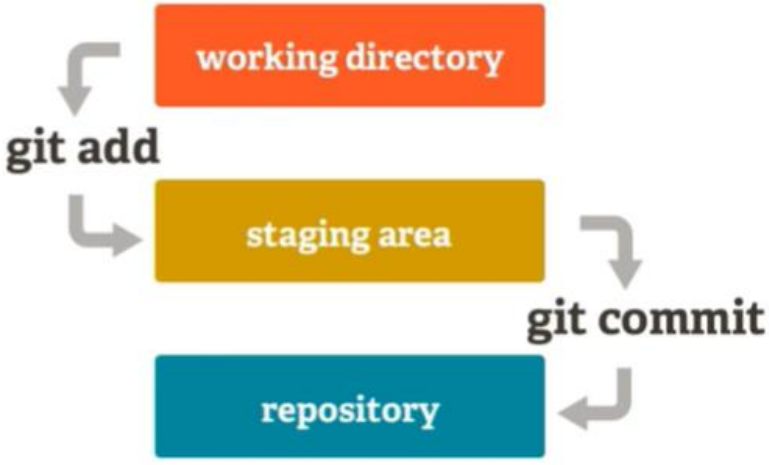
- Distributed version control
- Coordinates work between multiple developers
- Who made what changes and when
- Revert back any time
- Local & remote repos

Concepts of Git

- Keeps track of code history
- Takes "snapshots" of your files
- You decide when to make snapshots by making a commit
- you can visit any snapshot any time
- you can stage files before committing

Basic commands

- git init
- git add <file>
- git status
- git commit
- git push
- git pull
- git clone



GitHub

git pull →traer los cambios que han hecho otros desarrolladores.

git push→subir al directorio remoto.

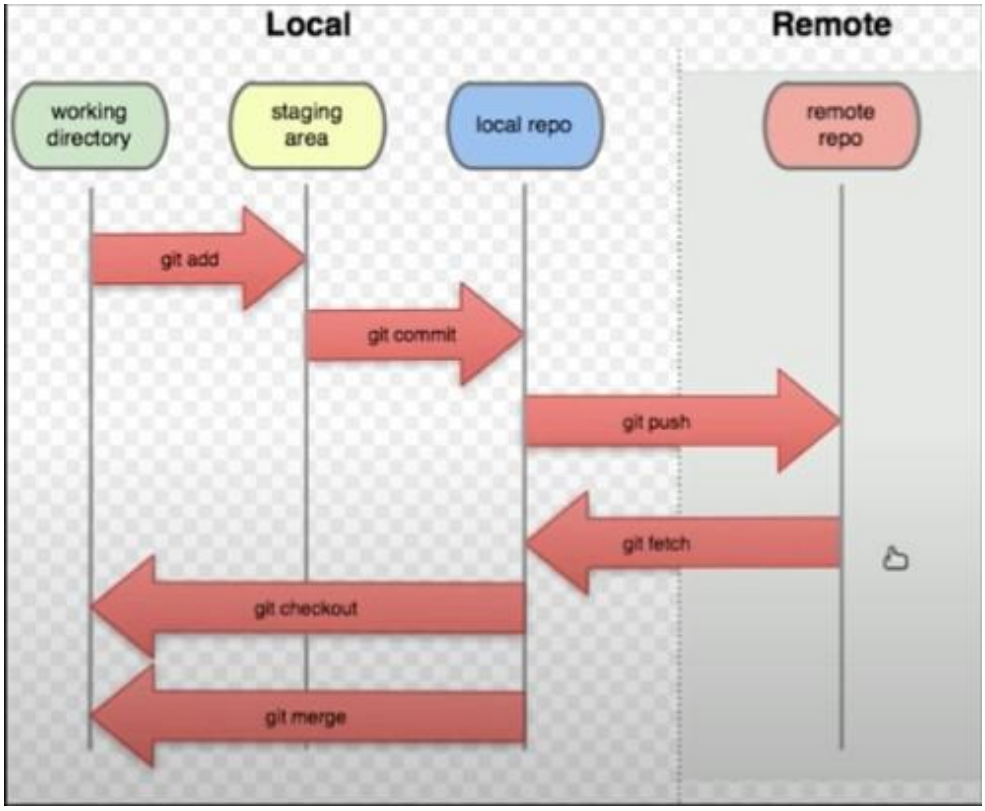
https://bluuweb.github.io/tutorial-github/guia/

En esta URL de arriba podemos encontrar un Tutoral de GitHub en castellano.

Página oficial de Git Hub:

https://git-scm.com/

Git es un entorno que nos va a permitir el control de versiones en nuestros proyectos para facilitar el trabajo colaborativo. Para que nuestro orden sea extraordinario.



Esquema del funcionamiento de Git

Nuestras carpetas locales donde estamos desarrollando el proyecto se representan en **working directory.**

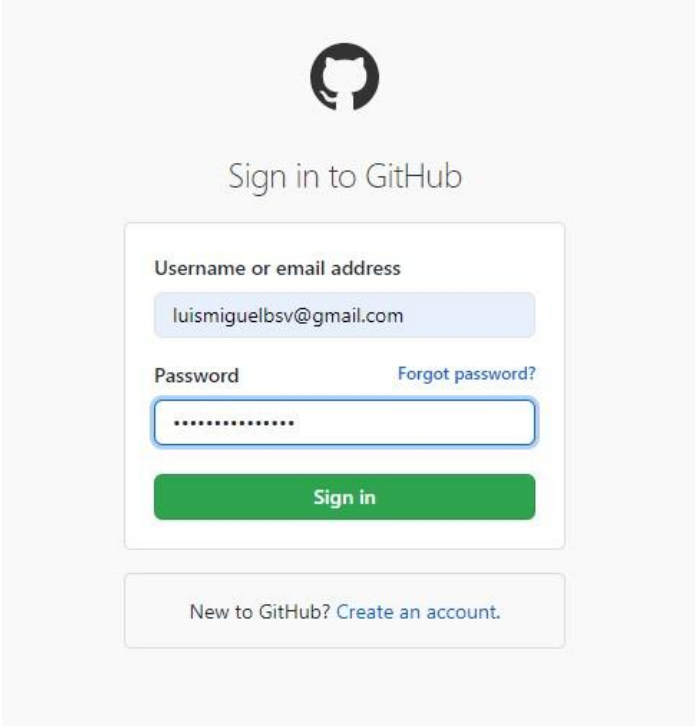
Cuando empezamos a realizar cambios se mandan todos los archivos (la mayor parte) en el **staging area** que es un área temporal que almacena los cambios que vamos realizando.

Y al guardar los cambios todo se almacena en el **local repo,** es un repositorio que guarda todos los cambios que hemos realizado.

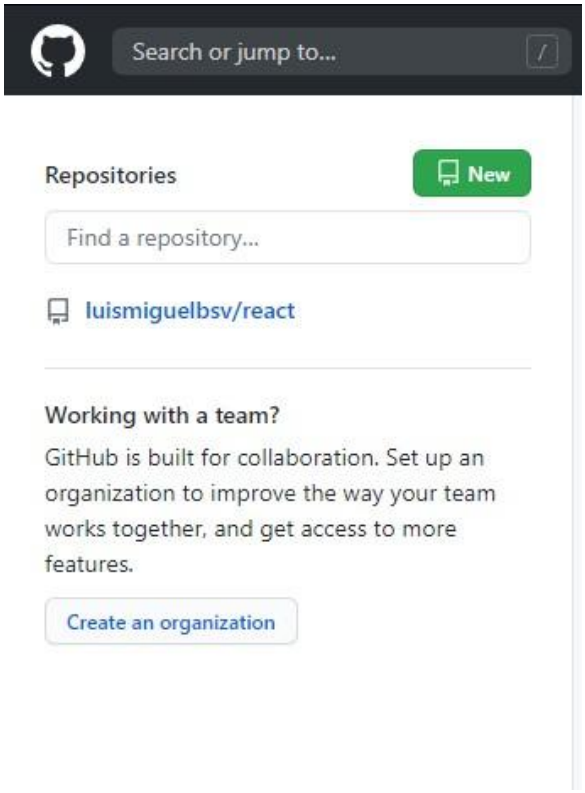
Lo primero que vamos a hacer es descargar Git desde su página web oficial https://git-scm.com/

Hacemos click en donde pone Download y una vez descargado la instalación es bastante sencilla , le damos next, next next.

Nos logueamos en la página de https://github.com/login

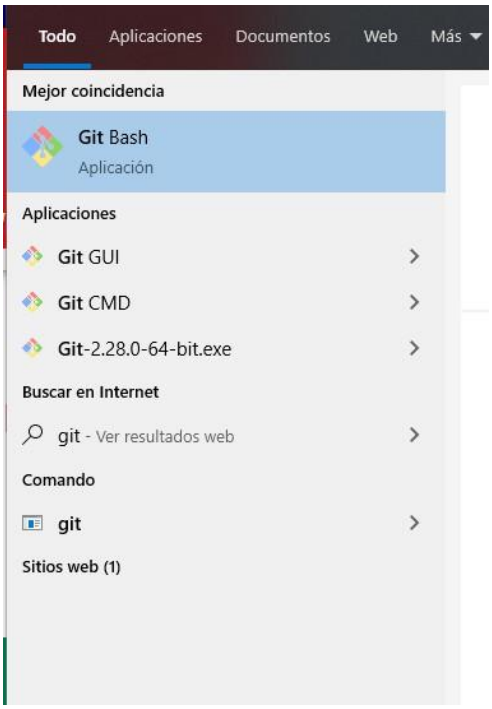


Creamos un repositorio en el botón de new o accedemos si ya tenemos uno creado al repositorio.



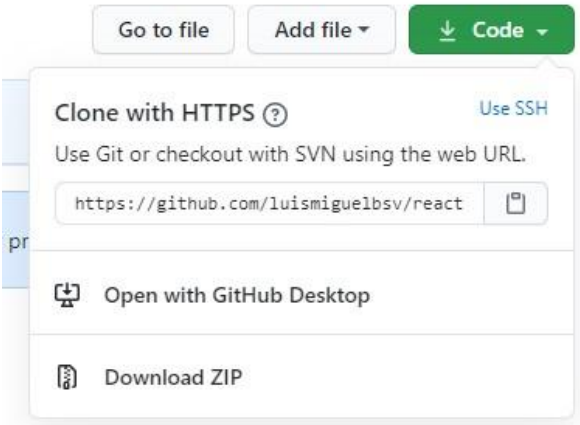
Vamos a abrir la herramienta GitBash que se instala cuando instalas GitHub.

La buscamos en la caja de búsquedas de Windows.



Al arrancarla nos aparece un terminal al estilo Linux que es la que vamos a utilizar para probar los comandos.

Vamos a copiar la URL desde la página web del repositorio con el fin de clonarlo en nuestro equipo:



Ahora nos situamos en el directorio /documents que es el que contendrá la carpeta react que es el nombre de nuestro repositorio en github. Usamos el comando **cd** para movernos previamente al documents.

Nota:

Nombres	Virgulilla, sombrero de la eñe, tilde de la eñe
Símbolo	~
Código ASCII	126
Entidad HTML	˜
Combinación en Windows	ALT + 126

cd / para ir al raíz

```
Usuario@DESKTOP-J3SL5U1 MINGW64 /
$ cd ~
```

Para movernos al directorio users

Y ahora para clonarlo escribimos:

$ git clone URL (la url de nuestro repositorio)