Manual básico de Git

Actualmente es necesario usar un software de control de versiones para el desarrollo colaborativo, sobre todo cuando se habla de proyectos grandes.

Un software de control de versiones es un grupo de aplicaciones para gestionar los cambios en el código fuente, dando la posibilidad de poder regresar a versiones viejas del mismo código fuente si así es requerido.

Esto se logra con un historial de versiones que va guardando el software en un repositorio, que puede ser una carpeta o conjunto de carpetas donde se almacenan estas versiones o cambios del proyecto y al cual tienen acceso los desarrolladores del proyecto.

El control de versiones es una potente herramienta colaborativa.

La necesidad de usar un software de control de versiones es que, el trabajo en equipo es muy complicado, sobre todo si se esta trabajando remotamente; si dos personas están modificando el mismo archivo, solo se preservaran los cambios de la última persona que guarde el archivo.

El software de control de versiones, facilita este trabajo en equipo, permitiendo que se varias personas puedan trabajar en el mismo proyecto al mismo tiempo evitando que se pierdan los cambios.

Ejemplos de software de control de versiones son: CVS, Tortoise, CVS, Clear Case, SourceSafe, etc

Virtudes de Git

Git es un software creado por Linus torvalds, el desarrollador del núcleo del S.O. Linux, tiene una curva de aprendizaje baja, es rápido, es eficiente al gestionar proyectos grandes, open source y permite el uso de ramas (branch).

Es necesario aprender a usar un poco la consola de los distintos sistemas operativos, para poder usar toda la potencia de git.

Comandos básicos de la consola de comandos de Windows msdos.

Existen muchos más comandos útiles pero estos son de los más utilizados

- **COMANDO**: MD
- **ACCIÓN:** Crea directorios
- **SINTAXIS:** MD [unidad:\ruta\] <nombre>

- **COMANDO:** RD
- **ACCIÓN:** Borra directorios (solo si se encuentra vacío)
- **SINTAXIS:** RD [unidad:\ruta\] <nombre>

- **COMANDO:** CD ; CD.. ; CD\
- **ACCIÓN:** Sirve para moverse por los distintos directorios
- **SINTAXIS:** CD <nombre de la carpeta> , CD.. , CD\

- **COMANDO:** DIR
- **ACCIÓN:** Muestra el contenido de un directorio
- **SINTAXIS:** DIR [unidad:\directorio]
- **PRINICPALES MODIFICADORES:** /P Pausa la pantalla para poder ver todo. /S Muestra también subdirectorios.

- **COMANDO:** MOVE
- **ACCIÓN:** Mueve ficheros de un directorio a otro
- **SINTAXIS:** MOVE <origen> <destino>
- **PRINICPALES MODIFICADORES:** /Y No pregunta por la confirmación de reemplazo

- **COMANDO:** DEL
- **ACCIÓN:** Elimina ficheros
- **SINTAXIS:** DEL [Unidad:\ruta\Archivo]
- **PRINICPALES MODIFICADORES:** /P Pide confirmación.

- **COMANDO:** EDIT
- **ACCIÓN:** Editor de texto
- **SINTAXIS:** EDIT <fichero>

- **COMANDO:** CLS
- **ACCIÓN:** Limpia la pantalla
- **SINTAXIS:** CLS

Principales comandos de Linux

Ls

Ls (de listar), permite listar el contenido de un directorio o fichero. La sintaxis es:

$ ls /home/directorio

El comando ls tiene varias opciones que permiten organizar la salida, lo que resulta particularmente útil cuando es muy grande. Por ejemplo, puedes usar *-a* para mostrar los archivos ocultos y *-l* para mostrar los usuarios, permisos y la fecha de los archivos. Así como para todos los comandos Linux, estas opciones pueden combinarse, terminando en algo como:

$ ls -la /home/directorio

Cd

Cd (de *change directory* o cambiar directorio), es como su nombre lo indica el comando que necesitarás para acceder a una ruta distinta de la que te encuentras. Por ejemplo, si estas en el directorio /home y deseas acceder a /home/ejercicios, seria:

$ cd /home/ejercicios

Si estás en /home/ejercicios y deseas subir un nivel (es decir ir al directorio /home), ejecutas:

$ cd ..

Mkdir

Mkdir (de *make directory* o crear directorio), crea un directorio nuevo tomando en cuenta la ubicación actual. Por ejemplo, si estas en /home y deseas crear el directorio ejercicios, sería:

$ mkdir /home/ejercicios

Mkdir tiene una opción bastante útil que permite crear un árbol de directorios completo que no existe. Para eso usamos la opción *-p*:

$ mkdir -p /home/ejercicios/prueba/uno/dos/tres

Cp

Cp (de *copy* o copiar), copia un archivo o directorio origen a un archivo o directorio destino. Por ejemplo, para copiar el archivo prueba.txt ubicado en /home a un directorio de respaldo, podemos usar:

$ cp /home/prueba.txt /home/respaldo/prueba.txt

En la sintaxis siempre se especifica primero el origen y luego el destino. Si indicamos un nombre de destino diferente, cp copiará el archivo o directorio con el nuevo nombre.

El comando también cuenta con la opción *-r* que copia no sólo el directorio especificado sino todos sus directorios internos de forma recursiva. Suponiendo que deseamos hacer una copia del directorio /home/ejercicios que a su vez tiene las carpetas ejercicio1 y ejercicio2 en su interior, en lugar de ejecutar un comando para cada carpeta, ejecutamos:

$ cp -r /home/ejercicios /home/respaldos/

Mv

Mv (de *move* o mover), mueve un archivo a una ruta específica, y a diferencia de *cp*, lo elimina del origen finalizada la operación. Por ejemplo:

$ mv /home/prueba.txt /home/respaldos/prueba2.txt

Al igual que *cp*, en la sintaxis se especifica primero el origen y luego el destino. Si indicamos un nombre de destino diferente, mv moverá el archivo o directorio con el nuevo nombre.

Rm

Rm (de *remove* o remover), es el comando necesario para borrar un archivo o directorio. Para borrar el archivo prueba.txt ubicado en /home, ejecutamos:

$ rm /home/prueba.txt

Este comando también presenta varias opciones. La opción *-r* borra todos los archivos y directorios de forma recursiva. Por otra parte, *-f* borra todo sin pedir confirmación. Estas opciones pueden combinarse causando un borrado recursivo y sin confirmación del directorio que se especifique. Para realizar esto en el directorio respaldos ubicado en el /home, usamos:

$ rm -fr /home/respaldos

Este comando es muy peligroso, por lo tanto es importante que nos documentemos bien acerca de los efectos de estas opciones en nuestro sistema para así evitar consecuencias nefastas.

Pwd

Pwd (de *print working directory* o imprimir directorio de trabajo), es un conveniente comando que imprime nuestra ruta o ubicación al momento de ejecutarlo, así evitamos perdernos si estamos trabajando con múltiples directorios y carpetas. Su sintaxis seria:

$ pwd

Clear

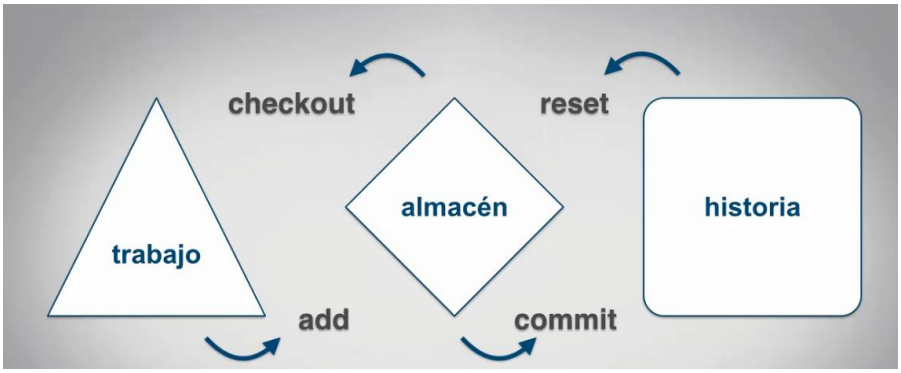
Clear (de limpiar), es un sencillo comando que limpiara nuestra terminal por completo dejándola como recién abierta. Para ello ejecutamos:

$ clear

Git se puede descargar desde la url https://git-scm.com, la ventaja de Git es que existen versiones para varios sistemas operativos.

La versión de Git para Windows, contiene una consola “tipo Linux” para realizar el trabajo con Git en ella.

Diagrama de funcionamiento de Git



Este es un diagrama básico de como trabaja Git; el área de trabajo o working directory es donde editamos nuestro proyecto.

El almacén también es llamado staging area y es donde se preparan los archivos con el comando git add, antes de ser confirmado con el comando git commit y que se guarde la versión en el repositorio.

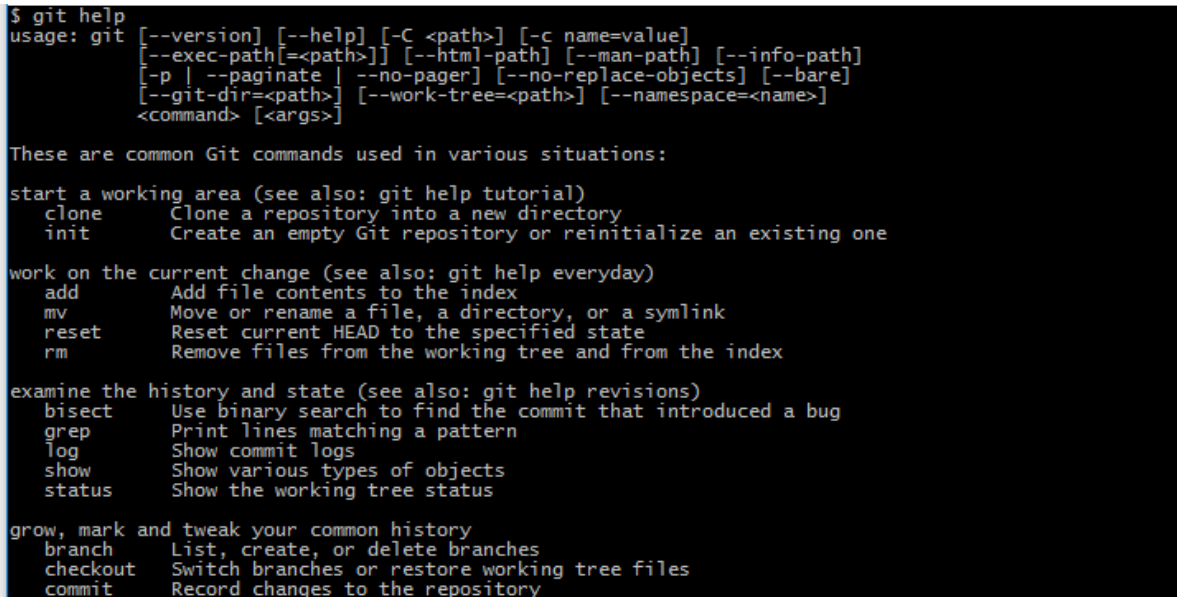
Una vez que se le da commit a los archivos, se almacenan sus versiones en el repositorio, ahí podemos regresar a versiones anterior en el staging area.

Al área de trabajo, almacén y a la parte donde se almacena el historial,se les conoce como los tres estados de trabajo de git.

Comandos de Git

Git help *comando de Git*

Te muestra ejemplos y una explicación del comando de Git, es el comando de ayuda.



Git config

permite obtener y establecer variables de configuración, que controlan el aspecto y funcionamiento de Git.

Git config –l

Para ver la configuración actual.

Los siguientes comandos son importantes al instalar git, ya que son datos que aparecen al hacer commit.

git config --global user.name "John Doe"
Para Establecer nombre de usuario al usar git

git config --global user.email johndoe@example.com
Para establecer tu correo en la configuración de git

También podemos configurar el editor que se usa por default para Git, el alias, etc.

Todos estos parámetros de configuración se guardan en el archivo de gitconfig.

Git Init

Git init crea un repositorio nuevo, inicializa todas las configuraciones para comenzar a trabajar en git.

Al crear el repositorio (carpeta de trabajo que almacenara los archivos), crea la carpeta .git, esta carpeta es la raíz del proyecto y contiene las configuraciones necesarias para trabajar.

Para trabajar en un directorio compartido, siempre se debe crear un repositorio con el comando:

git init –bare

Este comando crea un repositorio vacío pero omite el directorio de trabajo.

Es necesario crear una carpeta vacía para el trabajo colaborativo, también existe **git clone - -bare**.

Generalmente, los desarrolladores trabajan con clones de repositorios ya existentes (git clone).

Los archivos dentro de la carpeta oculta .git son:

HEAD, branches, config , description, hooks, info, objects, refs.

Git clone dirección: Nos traemos una copia de un repositorio existente a nuestra carpeta local para trabajar en ella.

Git add nombre Archivo

Prepara los cambios del directorio de trabajo al área de preparación para ser confirmados después con commit.

Es uno de los comandos más usados junto con commit, status y log.

Es el punto intermedio dentro del flujo de trabajo que es Editar, Preparar, Confirmar.

Con git add . Con el comando opcional [.], se suben todos los archivos disponibles para ser subidos de tu carpeta de working area.

Git commit

Es el comando para “confirmar” los cambios añadidos y preparados con el comando add en el staging area, al ejecutar el comando se guarda la versión en el repositorio y se pide que se guarde un mensaje que describa los cambios guardados en el archivo confirmado.

Después de hacer el comando git commit, se debe escribir un mensaje descriptivo del commit.

Se puede acelerar el proceso escribiendo el mensaje en el comando del commit de la siguiente manera: **git commit –m “mensaje”**

Cuando se trabaja en forma cooperativa se recomienda primero hacer commits atómicos, esto es primero commits de archivos que no están relacionados entre sí.

Esto permite encontrar errores de una forma más rápida.

Al poner el mensaje descriptivo del commit, hay que intentar ser muy aclarativos con el mensaje.

Commit es un comando que tiene muchos subcomandos que se pueden combinar.

git commit [-a | --interactive | --patch] [-s] [-v] [-u<mode>] [--amend]
[--dry-run] [(-c | -C | --fixup | --squash) <commit>]
[-F <file> | -m <msg>] [--reset-author] [--allow-empty]

[--allow-empty-message] [--no-verify] [-e] [--author=<author>]
[--date=<date>] [--cleanup=<mode>] [--[no-]status]
[-i | -o] [-S[<keyid>]] [--] [<file>…]

Los más usados son:

git commit –a

se agregan todos los archivos preparados.

Git commit –m

Se guarda en el repositorio el archivo y se le añade un mensaje.

Git commit –am

Se guardan todos los archivos preparados y se añade un comentario al commit.

***git rm* [-f | --force] [-n] [-r] [--cached] [--ignore-unmatch] [--quiet] [--] <file>…**
Se borra un archivo o carpeta del working directory y del stage.

Con **--cached** se borra del repositorio pero se deja en el working area.

git mv [-v] [-f] [-n] [-k] <source> <destination>
Mover o renombrar un archivo.

Git status

Informa el estado del árbol de trabajo.

Git diff

Muestra las diferencias entre commits, entre commit y el staging area, etc

Cuando usas el comando compara dos archivos a y b; donde a es el archivo que estaba y b el que cambió.

Con el signo + aparecen las líneas adicionales y con – las líneas que fueron borradas.

git diff –staged

se pueden ver las diferencias de archivos en el stage

Git diff [branch1] [branch2]

Muestra las diferencias entre dos ramas.

***git log* [<options>] [<revision range>] [[\--] <path>…]**
Muestra el historial del commit.

Git show [commit]

Muestra información del commit específico.

Git reset HEAD nombre del archivo

Marca el archivo para que no sea incluído en el próximo commit.

Git reset --soft HEAD

Deshace el commit y deja los cambios en el árbol local.

Git reset --hard HEAD

Restablece el árbol local a la versión anterior. Se pierden todos los cambios posteriores.

Git clean

Elimina los archivos desconocidos del árbol de trabajo local.

RAMAS EN GIT

Git checkout rama

Cambia de rama o reestablece archivos, crear ramas.

Git branch

Lista las ramas.

Git merge rama

Mezcla los archivos de la rama donde estas a la especificada.

Git pull

Descarga los cambios desde el repositorio.

Git push [alias] [branch]

Subimos los cambios al repositorio.

Git fetch

Trae los cambios pero no los fusiona.

Git remote

Lista los repositorios disponibles.

BANDERAS DE ESTADOS EN GIT

Cuando vemos el estado de los archivos en git, podemos encontrar las siguientes banderas .

M: Modified

C: Copy edit

R: Rename edit

A: Added

D: Deleted

U: Unmerged

git ignore

Para ignorar los archivos y que no aparezcan para ser preparados o para ser confirmados, se pueden incluir en el archivo exclude, dentro de la ruta .git/info/

git status

Muestra el estado del directorio de trabajo y del área de trabajo.

Aparece una ventana con los archivos preparados con git add en verde y los archivos no preparados en rojo.

Muestra mensajes de ayuda para continuar con el trabajo.

git log

lista las confirmaciones hechas sobre ese repositorio en orden cronológico inverso. Es decir, las confirmaciones más recientes se muestran al principio. Como puedes ver, este comando lista cada confirmación con su suma de comprobación SHA-1, el nombre y dirección de correo del autor, la fecha y el mensaje de confirmación.

Algunas de las opciones más usadas son:

Git log –n

Últimos n números de commits

Git log –oneline

Muestra en una línea todos los commits con el el número de comprobación abreviado.

Git rm

Para borrar ficheros en el repositorio.

Después de borrarlo se deb hacer commit, se borra en la raíz también.

Para git es lo mismo borrar y renombrar, solo que con renombrar se agregar un nuevo archivo.

***git mv* <options>… <args>…**
Mover y renombrar archivo

Git reset

Regresa a los cambios de un commit específico.

Git reset --soft numero commit

regresa los cambios del commit espec+ifico, dejando los archivos en el staging area.

Git reset –hard numero commit

regresa los cambios hasta el working area y se pierden todos los posteriores al commit especificado.

HEAD en git

Funciona como un apuntador de la carpeta o rama donde estamos ubicados trabajando.

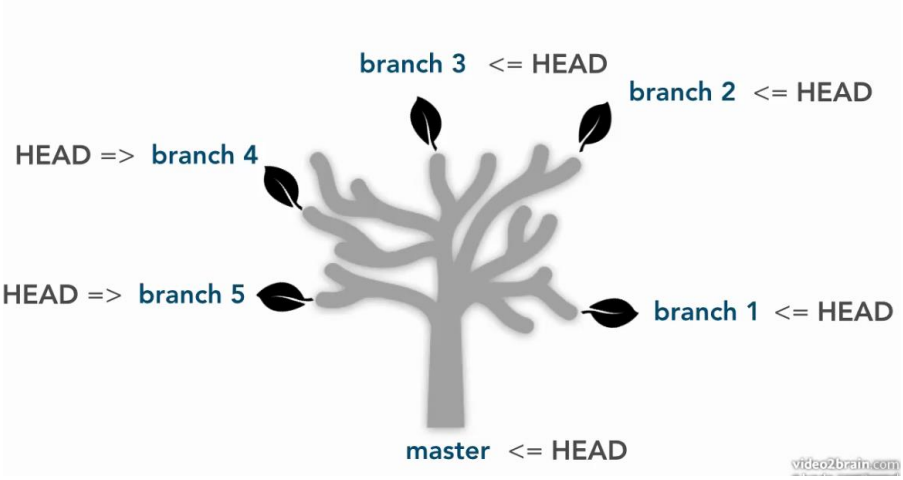
Ramas con git

La rama principal del proyecto se llama Master.

Uno de los puntos fuertes de git, es su manejo de ramas, lo hace de forma instantánea a diferencia de otros programas.

Cómo buena práctica, siempre deberíamos de crear otra rama para comenzar a trabajar, aparte de la rama Master.

Podemos clonar nuestro trabajo en la rama en la que estamos trabajando; cuando terminemos de realizar los cambios y de probarlos, podemos mezclar esa rama con la rama maestra.



Cuando hacemos commit, se guardan los HEAD de todas las ramas.

Git branch

Hace un listado de las ramas que tenemos.

Git branch[nombre de la rama]

crea una rama con el nombre indicado.

Git checkout [nombre de la rama]

Nos movemos a la rama indicada.

Git checkout –d [nombre de la rama]

Se crea una rama con el nombre indicado y nos posicionamos sobre ella.

Git branch –d [nombre de la rama]

Borramos la rama indicada, esto es necesario despúes de que ya fusionamos y terminamos de usar la rama indicada.

Merge en git

Git merge [rama]

Fusiona la rama en donde estamos ubicados con la rama indicada.

Para hacer un merge, nos ubicamos sobre la rama que recibirá los cambios y hacemos el comando

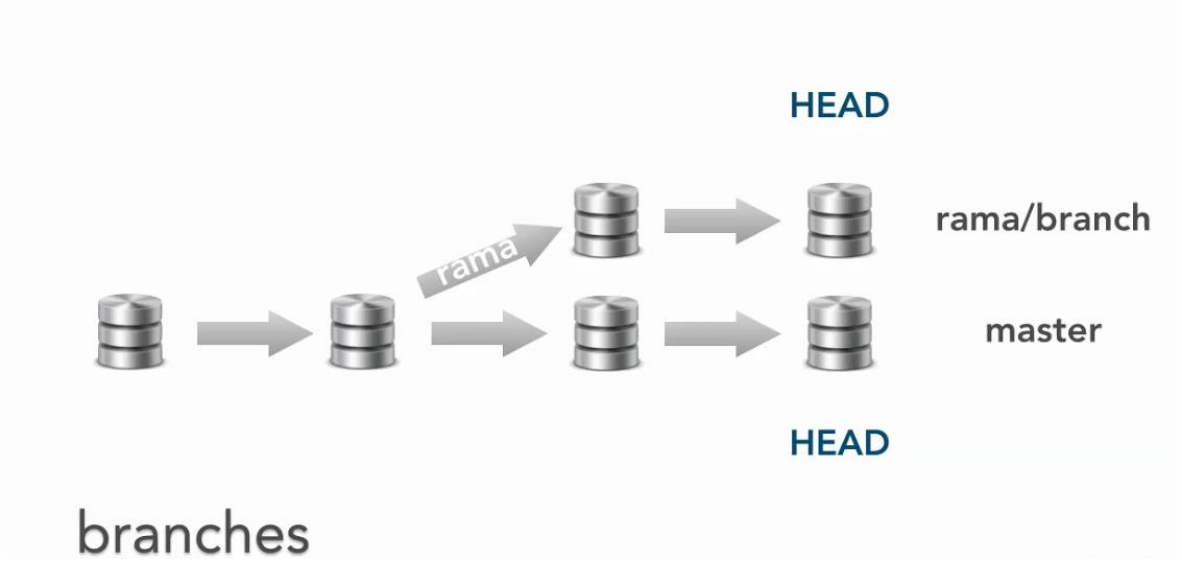
Git merge [rama de donde se traerán los cambios]

Al dejar de usar una rama es recomendable borrarla.

Es recomendable usar alguna herramienta grafica para trabajar con git y evitar confusiones.

En la página de git en la sección de downloads, en GUI clients, existen programas para usar git de manera gáfica.

Algunos de ellos son GitHub Desktop, Source Tree, gitg, etc.

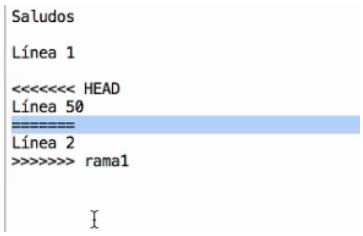


Deshacer Merge

Cuando se trabaja en forma cooperativa, se debe tener cuidado con que varios usuarios modifiquen el mismo archivo en lugares distintas, ya que al momento de mezclar los archivos, ocurrirá un error de conflicto con ese archivo.

La forma adecuada de resolver el conflicto sería hacer un reset a un commit anterior al conflicto (git reset --hard numerodecommit) y luego intentar el Merge nuevamente.

También podemos resolver el conflicto de manera manual, modificando las líneas que están provocando el conflicto.



Cuando hay un conflicto en el archivo y lo abrimos para revisarlo, aparece como en la imagen d arriba.

Aparecen las líneas que están provocando conflicto, separadas con un par de líneas paralelas.

En la parte donde aparece el HEAD, están las líneas del lugar donde estamos ubicados en la rama (donde pretendíamos recibir los cambios).

Y aparece también los cambios de la rama de donde queremos extraer los cambios.

Se debe eliminar un bloque, el que queramos para resolver el conflicto.

Luego se ejecuta **git add** y **git commit** después.

Es recomendable revisar en que rama estamos en cada momento para evitar errores.

Git remote

Un repositorio remoto es un área de trabajo mínimo que contiene únicamente el proyecto.

No nos conectamos de manera directa al servidor remoto, usamos el repositorio como intermediario.

Git es muy potente para el trabajo colaborativo

El comando git remote muestra una lista con los nombres de los repositorios remotos.

Git clone url del repositorio remoto

Clonar un repositorio remoto (origin: de donde se clonan los datos).

git remote -v url del repositorio

Muestra las url del push y del pull en el repositorio remoto

Cuando tu proyecto se encuentra en un estado que quieres compartir, tienes que enviarlo a un repositorio remoto. El comando que te permite hacer esto es sencillo: git push [nombre-remoto][nombre-rama]. Si quieres enviar tu rama maestra (master) a tu servidor origen (origin), ejecutarías esto para enviar tu trabajo al servidor:

git push origin master

Git push origin rama

Subir a la rama descrita

Git push origin :rama

Borrar la rama remota

Remote show url del repositorio

Muestra información del repositorio remoto

El protocolo git es parecido al protocolo para acceso remoto SSH, pero SSH es más seguro pues tiene autentificación y encriptación.

Git tag etiqueta

Permite ponerle etiquetas al proyecto,

Existen otros hosting para trabajo colaborativo, tanto privados como gratuitos, de los más populares son bit bucket y GitHub.

Es necesario conocer al menos uno por que en ocasiones es un requisito para algunos empleos y para enriquecer el curriculum.

Github es el más popular, es open source y lo usan empresas importantes como Google y es multiplataforma.

Github tiene una interface muy fácil de usar y tiene bastante prestigio, es buena idea tener tu carpeta de proyectos en Github.

Bitbucket es el más usado por frelancers de todo el mundo, no es tan llamativo como Git hub pero si es muy potente.

Ambos tienen clientes con interfaces gráficas fáciles de usar, en el caso de Bit bucket es recomendable usar Source tree.

Una buena recomendación es usar tanto bitbucket como github dependiendo del tipo de proyecto, para mostrarlo es recomendable Github por su reputación, y para proyectos en proceso es recomendable bitbucket.

Solo es necesario un poco de práctica para dominarlos.