En el post de hoy veremos algunos de los comandos git más utilizados, pero tranquilo, si ya eres un usuario más avanzado de git, también veremos algunos comandos menos conocidos y avanzados pero igualmente útiles.

Pero antes de todo esto, no está de más recordar qúe es git y para qué sirve por si hay alguien que se quiere iniciar en git.

Git es una herramienta que sirve para gestionar el control de versiones. Pero, ¿qué es el control de versiones? Pues es simplemente una forma de tener controlados todos los cambios que realizamos sobre cualquier tipo de archivos. Este control se puede efectuar a mano, aunque es recomendable usar una herramienta que nos facilite la vida como por ejemplo git, aunque hay muchas más.

**Instalación y configuración de Git.**

Lo primero es tener instalado git en nuestra máquina para tener acceso a todos sus comandos. Para descargar e instalar git lo puedes hacer desde su página oficial:

Si estás en Windows, puedes descargarlo directamente desde aquí:

<https://git-scm.com/download/win>

Si usas otro sistema puedas buscar el tuyo aquí:

<https://git-scm.com/downloads>

Una vez instalado git, puedes configurar tu identidad ejecutando en la terminal (si estás en windows se te habrá instalado un programa llamado Git bash):

git config --global user.name "Tu nombre aquí"

git config --global user.email ejemplo@example.com

Los comandos de git que voy a explicar a continuación se ejecutan tambien en consola.

**Comandos básicos**

Bien, pongámonos en la situación de que tenemos un proyecto que estamos dearrollando y queremos cambiar algo drástico. Lo mejor es usar git para llevar el control de estos cambios por si queremos revertirlos.

El primer paso es abrir la terminal de git, o la terminal de nuestro sistema, para dirigirnos a la carpeta donde tengamos guardado el proyecto que estemos desarrollando, y ejecutar el siguiente comando:

git init

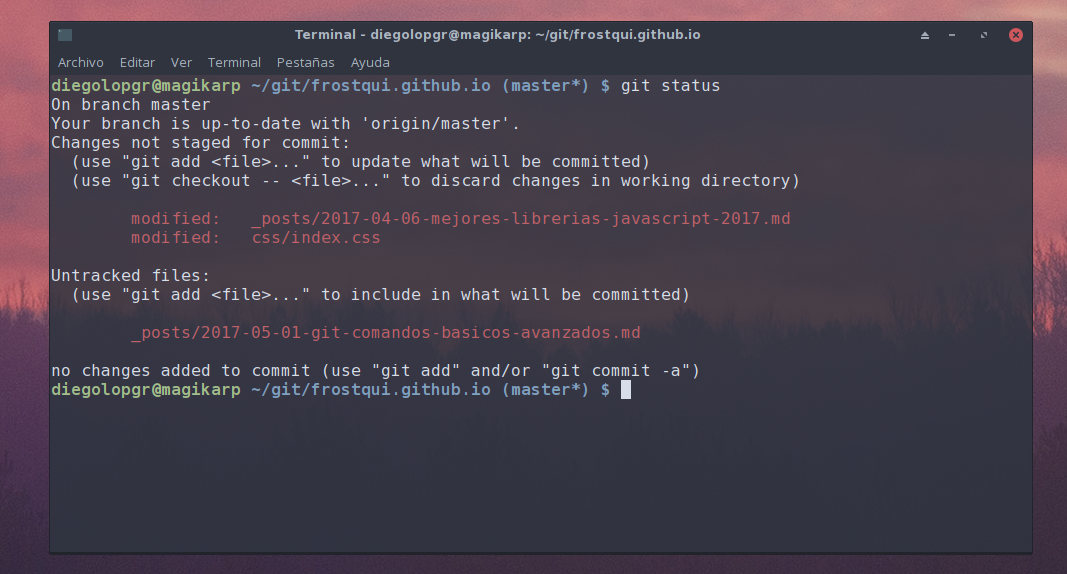
Con este comando de git lo que estamos haciendo es decirle a git que este pendiente de los cambios que se produzcan en los archivos de ese directorio.

Este comando solo lo tenemos que ejecutar una sola vez para cada proyecto que estemos realizando.

Ahora podemos continuar desarrollando nuestro proyecto.

Cuando queramos guardar los cambios con git tenemos que hacer lo siguiente:

git status



Este comando git imprimirá los archivos que van a ser guardados, el siguiente paso es ejecutar:

git add .

Con esto añadiremos todos los archivos que aparecían anteriormente para ser guardados. Si queremos añadir un archivo o carpeta en concreto lo podemos hacer mediante

git add NOMBREDELARCHIVO

Si queremos eliminar los archivos que acabamos de añadir para ser guardados lo podemos hacer con

git rm .

Con estos comandos hemos añadido o hemos quitado archivos pero aún no han sido guardados, para ello:

git commit -m "Nombre descriptivo del cambio que hemos realizado"

Acabamos de hacer nuestro primer commit, un commit es un guardado con mensaje de los cambios que hemos realizado en un momento determinado en nuestro proyecto. Normalemente el flujo de trabajo con git para el día a día consiste en hacer git status para ver la lista de cambios, git add de todos los archivos o de los que queramos, y por último hacer commit.

Para imprimir todos los commits que hemos realizado tenemos el comando de git:

git log

La cadena de números y letras que aparece al lado de la palabra “commit” es el identificador que podemos usar para revertir los cambios y volver atrás a ese punto. Para ello ejecutamos:

git reset --hard IDENTIFICADOR

Sustituimos IDENTIFICADOR por el identificador que comentaba anteriormente, eliminando así, todos los commits posteriores a ese commit.

Con *git revert* se crea un nuevo commit que revierte los cambios realizados en el último commit, pero no elimina dicho commit.

Si quires ver los cambios que has realizado desde el último commit lo puedes hacer con:

git diff

**Administrando proyectos git**

Ahora, imaginemos que queremos administrar un proyecto ubicado en otro servidor o en un almacén de repositorios como es Github

Si queremos bajarnos el proyecto para empezar a gestionarlo lo podemos hacer usando:

git clone git://servidor/ruta/a/los/archivos

Por ejemplo, para GitHub:

git clone https://github.com/proyectogithub

Si alguien hace un cambio desde otro sitio a este mismo repositorio, tenemos que bajarnos el cambio a nuestra máquina, para ello:

git pull

Pero, si tenemos cambios sin guardar, tenemos que guardarlos haciendo commit, antes de bajarnos nada.

Tras hacer commit de cambios nuevos, podemos subirlos al repositorio remoto:

git push origin master

Donde origin es la dirección del repositiorio remoto y master es la rama a la que estamos subiendo los cambios.

Para añadir la dirección de un repositorio remoto a origin, lo podemos hacer así:

git remote add origin https://github.com/user/repo.git

De esta forma, podemos usar la palabra origin para referirnos a esa dirección.

**Ramas en git**

Muy bien, pero antes has dicho que has subido los cambios a una rama, ¿qué es una rama?

Las ramas sirven para llevar un control de cambios independiente en el mismo repositorio, es decir, podemos crear una rama a partir de la rama base, o rama master, con otra serie de cambios, posteriormente podemos subir estos cambios a la rama master haciendo una combinación de ambas ramas:

git push origin master

En verde, saldrán las líneas que has añadido, y en rojo las que has eliminado desde la última vez.

Para posicionarnos en una rama escribe:

git checkout nombre\_de\_la\_rama

Y para crear una rama:

git checkout -b rama\_nueva

Para combinar ramas, es decir, git va a tratar de hacer una fusión de los cambios de ambas ramas, si llega a una línea que se ha cambiado en los dos sitios a la vez al no saber que cambio es el que el usuario quiere saltará un conflicto de git. Para arreglarlo tienes que decidir que cambio quieres conservar y volver a pushear para hacer el merge:

git merge hotfix

Este comando mergea los cambios de la rama llamada **hotfix** a la rama en la que nos encontremos. Puedes ver en qué rama te encuentras si ejecutas:

git brach

Normalmente las ramas en git se utilizan para asilar funcionalidades, es decir, creamos una rama con el nombre de una funcionalidad, hacemos todos los commits que queramos y al finalizar la funcionalidad hacemos un merge de ramas. Esto permite tener equipos trabajando independientemente a la vez en el mismo proyecto sin pisar funcionalidad, esta estrategia se conoce como **git flow**. Puedes consultar más información de esto en este enlace: <http://aprendegit.com/que-es-git-flow/>

**Comandos avanzados**

Vistos estos comandos básicos de git toca ver algo más complejo. No son comandos muy complicados pero no son recomendables si no controlas del todo git. Para ciertas situaciones estos comandos de git son muy útiles.

**Cómo crear alias en git**

Para no tener que estar escribiendo todo el rato “commit” o “checkout” podemos crear alias, los alias de git sirven para decirle a git que comando tiene que ejecutar para el alias que le hemos indicado, estos son los alias más comunes, aunque puedes crear y configurar más. Personalmente considero los alias imprescindibles.

git config --global alias.co checkout

git config --global alias.br branch

git config --global alias.ci commit

git config --global alias.st status

De esta forma, al escribir *git st*, git ejecutará el comando *git status* ahorrándonos mucho tiempo.

**Descartar temporalmente cambios**

Si estas trabajando en una rama y quieres cambiarte a otra, git no te dejará porque tienes cambios sin guardar, una forma de solucionar esto es haciendo un commit, pero si no queremos hacerlo lo que podemos hacer es descartar los cambios temporalmente, para ello:

git stash

Posteriormente si los quieres recuperar:

git stash pop

Este comando git es de los más útiles para no tener que estar guardando cosas momentáneamente en otros sitios si te quieras cambiar rápido de rama

**Pull de un solo commit**

Si por cualquier motivo, necesitas hacer un pull pero solo de un determinado commit lo que puedes hacer es usar este comando:

git cherry-pick <commitSHA>

**Git log avanzado**

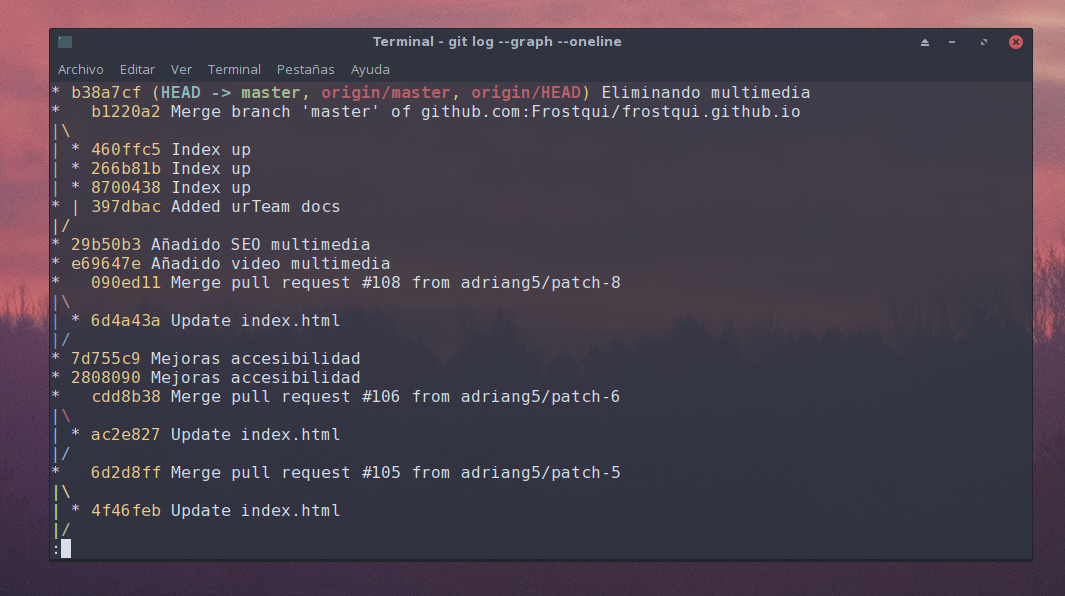
Hay veces en las que el comando git log ofrece demasiada información, pero esto se puede personalizar, por ejemplo:

git log --online

Se imprimirá en cada linea un commit, con su identificador y el texto del commit.

Otro parámetro bastante útil del log es el de git graph

git log --graph --oneline



Esto imprimira la lista de commits y mediante caracteres ASCII, representara el árbol con las ramas y los cambios entre ellas.

También podemos filtrar los commits, por ejemplo:

git log --author="John"

git log --after="2014-7-1"

git log -- foo.py bar.py

Estos comandos filtarán los commits por autor, por fecha y por los archivos que fueron modificados respectivamente.

**Conclusiones**

Git es una herramienta muy potente que ofrece muchísmos comandos para la gestios de versiónes de nuestros proyectos. Los commmits que he explicado son unos cuantos, me dejo muchos de ellos, pero te animo a que eches un vistazo a la documentación oficial de git para que descubras muchos mas comandos y configuraciones.