Comandos GIT

git help <command>

git clone <uri> namedir # clona usando como nombre de directorio namedir.

git add <dir> # añade recursivamente todos los archivos del dir.

git diff --staged #compares staged changes with last commit

git commit -v # muestra el diff en el editor

git commit -a -m ” #automatically stage tracked files. No hace falta git add

git rm --cached <file or regexp> #Git no realiza un seguimiento del archivo, pero los deja en el directorio de trabajo. Útil cuando se olvida añadir archivos al .gitignore y ya hemos agregado dichos archivos al repositorio.

git rm <file> #borrarlos con git siempre.

git rm -f <file> # si ya está modificado y en el index.

git mv <file> <renamed\_file>

gitk # tcl/tk. Herramienta gráfica para git

git commit --amend #Modificar el mensaje del último commit

git reset HEAD <file> # to unstage

git checkout -- <file> # Descartar cambios en el directorio de trabajo.

AÑADIR ARCHIVOS

git add -i #interactive staggin

git add -p #crea patch

git stash #guarda el estado en una pila y limpia el directorio para poder cambiar de rama

git stash list #muestra la pila

git stash apply # vuelve al estado original del dir. Stash{n} especifica uno concreto Y --index reaplica los cambios stagged

git stash pop # elimina el primero en la pila. O drop

git branch -d <rama> #elimina la rama

git push origin --delete <branchName> # Elimina una rama del servidor

git mergetool #Herramienta gráfica para resolver conflictos

git branch # lista ramas

git branch -v # lista ramas mostrando último commit

git branch --merged #lista ramas que han sido mezcladas con la actual. Si no tienen un \*, pueden borrarse, ya que significa que se han incorporado los cambios en la rama actual.

git branch --no-merged #lista ramas que no han sido incorporadas a la actual.

**SERVIDOR**

Siempre hay que hacer pull antes de push en caso de que alguien haya subido cambios al servidor. Ejemplo:

User1 clona el repo y hace cambios, realiza un commit

User2 clona el repo, hace cambios, hace commit y sube los cambios con push

User1 intenta hacer push, pero será rechazado con: ! [rejected] master -> master (non-fast forward). No puede subir los cambios hasta que no mezcle el trabajo que ha subido User2. Así que debe hacer lo siguiente:

git fetch origin

git merge origin/master

git push origin master

Mientras User1 hacía estas operaciones, User2 ha creado una rama issue54 y realizado 3 commits, sin haber descargado los cambios de User1. Para sincronizar el trabajo, User2 debe hacer:

git fetch origin

git log --no-merges origin/master ^issue54 #Observa qué cambios ha hecho User1

git checkout master

git merge issue54 && git merge origin/master

git push origin master

git diff --check #Antes de hacer commit, ejecutar esto para ver si hemos añadido demasiados espacios que puedan causar problemas a los demás.

Commits pequeños que se centren en resolver un problema, no commits con grandes cambios.

git add --patch #En caso de hacer varios cambios en el mismo archivo

El mensaje del commit debe tener la estructura siguiente: Una linea de no más de 50 caracteres, seguida de otra línea en blanco seguida de una descripción completa del commit.

PASOS A SEGUIR PARA CONTRIBUIR A PROYECYOS AJENOS, MEDIANTE FORK

git clone <url>

git checkout -b featureA

git commit

git remote add myFork <url>

git push myFork featureA

git request-pull origin/master myFork #enviar la salida por mail al propietario del proyecto, o hacer click en pull request.

Buena practica tener siempre una rama master que apunte a origin/master, para estar siempre actualizado con los ultimos cambios en el proyecto original.

#Separar cada trabajo realizado en topic branch, que trackeen a origin/master

git checkout -b featureB origin/master

(Hacer cambios)

git commit

git push myFork featureB

(Contactar con el propietario del proyecto)

git fetch origin

#Otro ejemplo, el propietario del proyecto quiere aceptar un pull tuyo, pero quiere que hagas algunos cambios, aprovechas la oportunidad y mueves tu trabajo para basarlo en el contenido actual de la rama origin/master, aplastas los cambios en featureB, resuelves conflictos, y haces push:

git checkout -b featureBv2 origin/master

git merge --no-commit --squash featureB

(cambiar la implementacion)

git commit

git push myFork featureBv2

#--squash coge todo el trabajo de la rama mezclada y la aplasta en un no-merge commit encima de la rama en la que estas. --no-commit no registra el commit automaticamente. Así puedes realizar todos los cambios necesarios y luego hacer el commit

**SUBMODULOS**

git submodule add <url> # crea un directorio que contiene el comtenido de otro proyecto.

Clonar un repo con submodulos

git clone url

git submodule init

git submodule update

**CONFIGURACION**

git config --global <opcion> <valor> #global para usuario, system todos y sin nada, especifico para el repo.

git config {key} # muestra el valor de key

git config --global core.editor <editor> #cambia el editor por defecto

git config --global commit.template $HOME/.gitmessage.txt #plantilla para commits

git config --global core.pager ‘more|less’ #paginador por defecto, puedes usar cualquiera

git config --global user.signingkey <gpg-key-id> # clave gpg para firmar tags

git config --global core.excludesfile <file> #como gitignore

git config --global help.autocorrect 1 # autocorrige cuando se escribe un comando incorrecto. Solo en git >= 1.6.1

git config --global color.ui true # colorea la salida de git. Valores: true|false|always

git config --global core.autocrlf input #para que usuarios linux no tengan problemas con los retornos de carro de windows

git config --global core.autocrlf true #para usuarios de windows

git config --global core.whitespace trailing-space, space-before-tab, indent-with-non-tab, cr-at-eol # respectivamente: busca espacios al final de línea, busca espacios al inicio de tabulación, busca líneas con 8 o más espacios en lugar de tabulaciones, acepta retornos de carro

git apply --whitespace=warn <patch> # advierte de errores de espacios antes de aplicar el patch. Con --whitespace=fix intenta arreglarlos