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

**AÑADIR ARCHIVOS**

* **git add -i***#interactive staggin*
* **git add -p***#crea patch*

**REMOToS**

* **git remote -v***# lista los repos remotos*
* **git remote add [shortname] [url]***# crea nuevo remote, es posible descargar el contenido de ese repo con git fetch [shortname]. Master branch en [shortcode]/master*
* **git fetch <remote>***# descarga trabajo nuevo a máquina local, no sobreescribe nada tuyo. ( git pull sí hace merge automaticamente si se esta realizando un seguimiento de esa branch)*
* **git push [remote-name] [branch-name]***# sii nadie ha hecho push antes*
* **git remote show [remote-name]***# inspecciona remote.*
* **git remote rename <old-name> <new-name>***# también renombra branches: quedaría <new-name>/master*
* **git remote rm <remote-name>***# p.e si el contribuidor ya no contribuye más*

**AÑADIR VARIOS REPOSITORIOS REMOTOS**

* **git remote add bitbucket git@bitbucket.org:algui91/grado\_informatica\_tsi\_practicas.git** *# Añadir un nuevo repositorio remoto con el nombre deseado. Por ejemplo si ya tenemos uno en github y queremos añadir otro para bitbucket*
* **git push -u bitbucket –all** *# Subir el proyecto a bitbucket. A partir de ahora se puede seleccionar a qué repo publicar con***git push nombre\_repo\_remoto**

**TAGGING**

# marcan puntos importantes en la histtoria del repo ( releases )

* **git tag***# muestra las etiquetas actuales*
* **git tag -l ‘v1.4.2.\*’***# acepta regex*
* Dos tipos de tag:
  + **Lightweight** : puntero a commit ( branch que no cambia )
  + **Annotated** : se almacenan como objetos en la db, con checksum, nombre del creador, email, fecha, mensaje, posibilidad de firmarla con[GPG](http://elbauldelprogramador.com/seguridad/como-cifrar-correos-con-gpg-con-mailvelope/). ( recomendada )
* **git tag -a <tagname> -m ‘mensaje’***# annotated tag*
* **git show <tag-name>***# muestra información asociada.*
* **git tag -s <tag-name> -m ‘message’***# la firma con gpg*
* **git tag <tag-name>***# lightweight tag*
* **git tag -v <tag-name>***# verifica tags firmadas*
* **git tag -a <tag-name> [commit-chksum]***# crea tag para commit con dicho chksum*
* Por defecto no se transfieren los tags, para subirlos al servidor:
  + **git push origin [tag-name]***# una sola*
  + **git push origin --tags***# Enviar todas*
* Para usar GPG y firmar tags, hay que subir la clave pública al repositorio:
  + **gpg --list-keys** *#Coges la id pública*
  + **gpg -a --export <id> | git hash-object -w --stdin** *#Copia el SHA-1 devuelto*
  + **git tag -a maintainer-gpg-pub <SHA-1>**
  + **git push --tags***#Comparte la clave con todos los usuarios*
  + **git show maintainer-gpg-pub | gpg --import***#Cada usuario importa la clave así*
  + **git show <tag>***#Devuelve más información sobre la etiqueta*
  + **git tag -d nombre\_tag** *# eliminar la etiqueta*
  + **git push origin :refs/tags/nombre\_tag** *# Eliminar la etiqueta del repositorio remoto.*

**UTILIDADES**

* + **git show <short-SHA-1>***#Es posible ver un commit pasando la versión abreviada del SHA-1*
  + **git rev-parse <branch>***#A qué SHA-1 apunta una rama*
  + **git show HEAD^***# Muestra commit padre*
  + **git show HEAD^2***#Muestra segundo padre*
  + **git show HEAD~2***# El primer padre del primer padre*
  + **git filter-branch --tree-filter ‘rm -f <file>’ HEAD***#elimina el archivo de todos los commits*

**DEPURACIÓN**

* + File anotation
    - **git blame -L 12,22 <archivo>***# muestra cuando y por quién se modificaron de la linea 12 a la 22*
    - **git blame -C -L 141,153 <file>***# cuando renombras un archivo o lo refactorizas en varios, muestra de donde vino originalmente.*
  + Búsqueda Binaria: Cuando hay un bug que no puedes localizar, usas bisect para dererminar en qué commit empezó a producirse el bug.
    - **git bisect start**
    - **git bisect bad***# marcas el commit actual como roto*
    - **git bisect good [commit bueno]***# último commit conocido que funcionaba*
    - Ahora irá preguntando hasta que encuentres el commit culpable. Si esta bien indicas git bisect good. De lo contrario git bisect bad. Al terminar hay que resetear.
    - **git bisect reset**

**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**

**CONFIGURACIóN**

* + **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*