**CENTRO DE DESARROLLO DE SOFTWARE E INVESTIGACIÓN**



**Documentación**

**“Comandos Básicos en GIT”**

**Encargada:**

**ISC** **Erika Jazmín Robles Gómez**

**Sombrerete, Zac., a 25 de enero de 2015**

Contenido

[Introducción 2](#_Toc413100353)

[Comandos Básicos 2](#_Toc413100354)

[Añadir Archivos 4](#_Toc413100355)

[Stash 4](#_Toc413100356)

[Logs 5](#_Toc413100357)

[REMOTES # REPOS EN INTERNET 6](#_Toc413100358)

[AÑADIR VARIOS REPOSITORIOS REMOTOS 7](#_Toc413100359)

[Etiquetas 7](#_Toc413100360)

[marcan puntos importantes en la historia del repo ( releases ) 7](#_Toc413100361)

[BRANCH 9](#_Toc413100362)

[Remote branches 10](#_Toc413100363)

[Tracking Branches 11](#_Toc413100364)

[Rebase 11](#_Toc413100365)

[# Rebase y merge 11](#_Toc413100366)

[Servidor 12](#_Toc413100367)

[Generar Un Número De Compilación (Build Number) 12](#_Toc413100368)

[Generar Un Changelog 12](#_Toc413100369)

[Recomendaciones 12](#_Toc413100370)

[Pasos A Seguir Para Contribuir A Proyectos Ajenos, Mediante Fork 13](#_Toc413100371)

[Reflog. 14](#_Toc413100372)

[Utilidades 14](#_Toc413100373)

[Depuración 14](#_Toc413100374)

[Submodulos 15](#_Toc413100375)

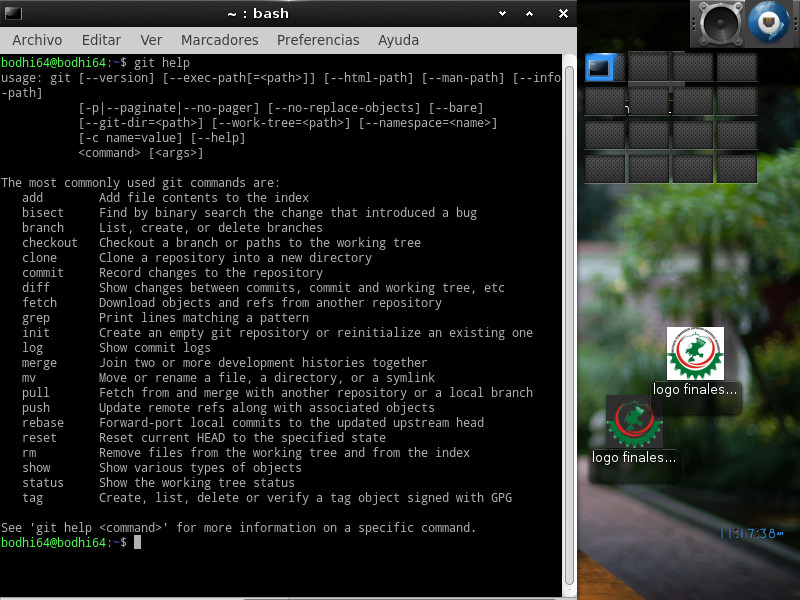
[Configuration 15](#_Toc413100376)

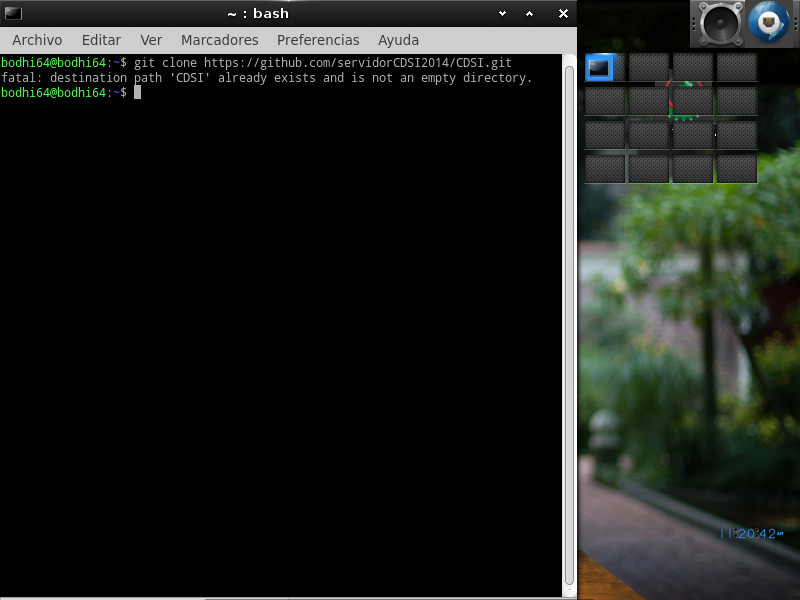
[Bibliografía 17](#_Toc413100377)

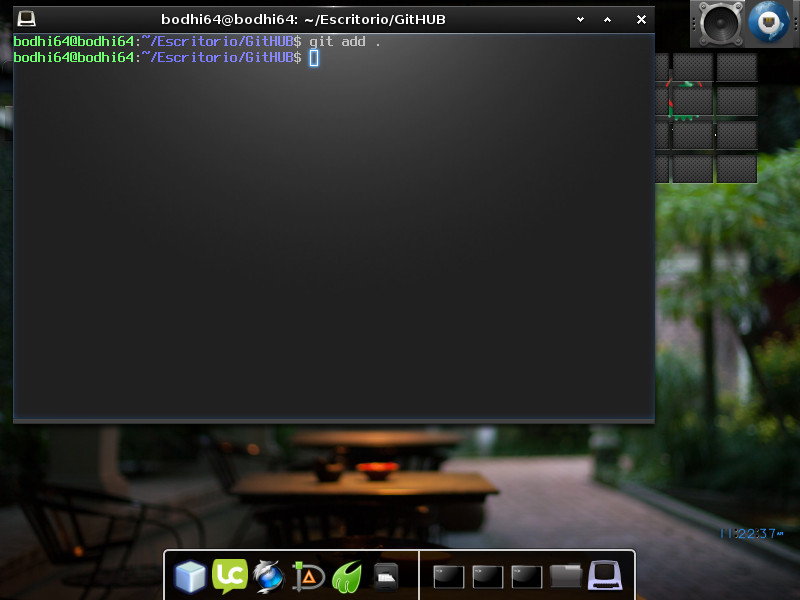
# Introducción

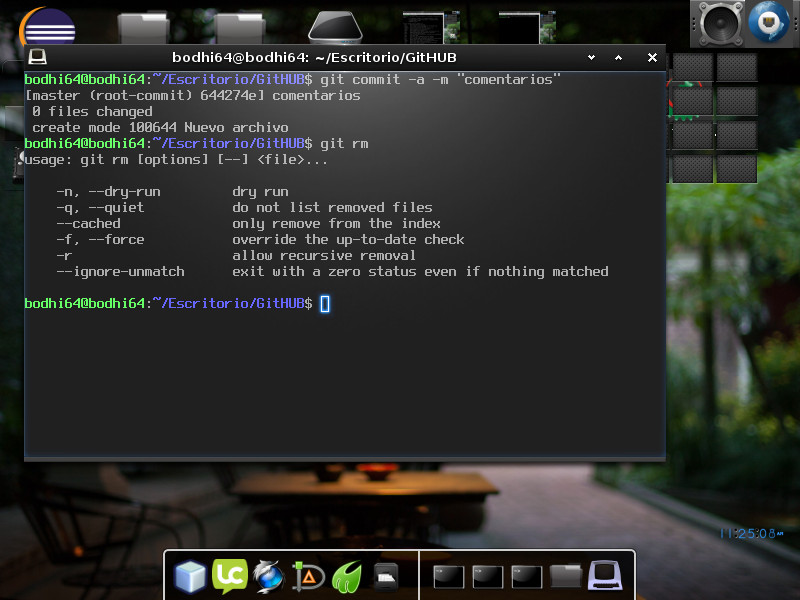
En este manual se explican algunos de los comandos básicos en Git los cuales son útiles al usarlo directamente en la consola

## Comandos Básicos

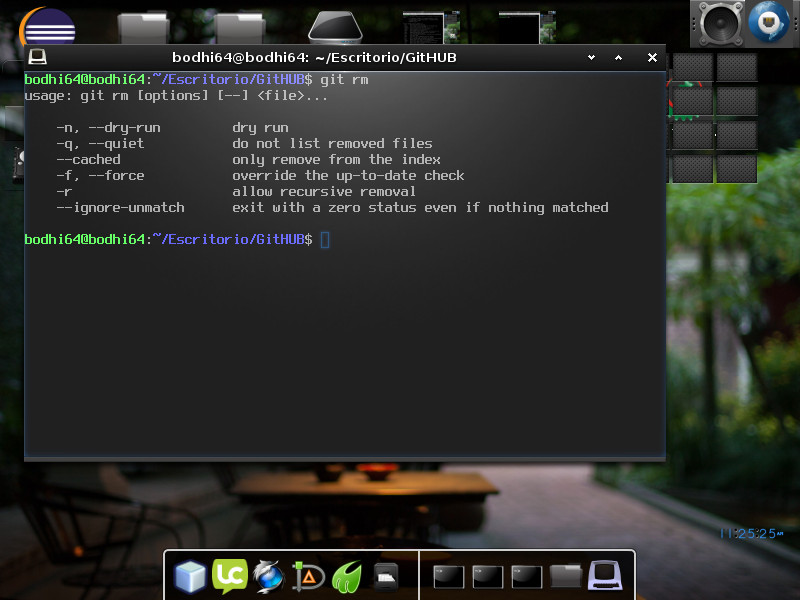
* **git help <comando>** 
  + Al teclear git help se mostrara una ventana en la que contiene comandos que se pueden utilizar en el transcurso de tu trabajo en git
* **git clone <url> namedir**
  + *#* clona usando como nombre de directorio namedir, este es usado cuando el directorio se encuentra en el mismo lugar; es decir cuando re crea el repositorio en una partición diferente.

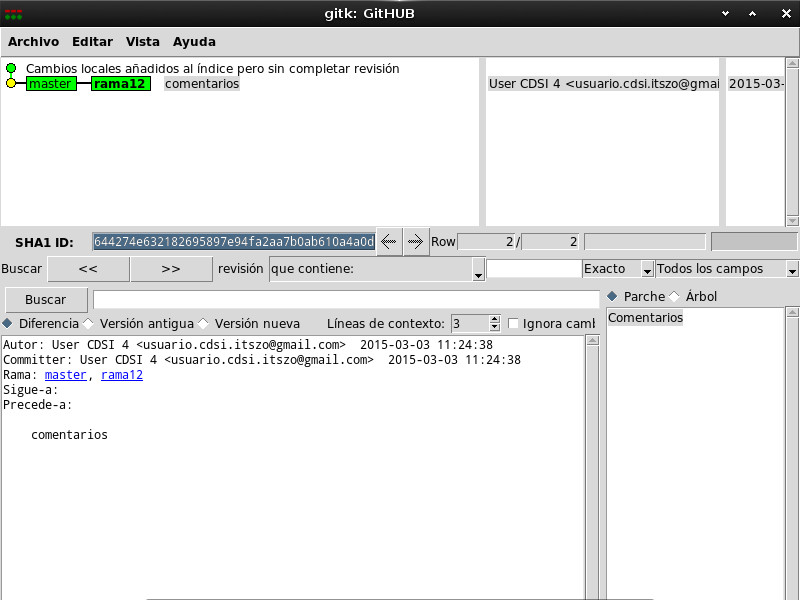


* **git add <dir>**
  + *#* añade recursivamente todos los archivos del directorio por ejemplo si se tiene el repositorio en una carpeta que contiene X número de archivos se añadirán todos los archivos existentes en el directorio.
* **git diff --staged**
  + *#(compares staged changes with last Commit)* comparara los cambios con la rama principal
* **git commit -v**
  + *#* una vez que haya realizado el comando anterior mostrara en el editor que se halla configurado las diferencias.
* **git commit -a -m ”**
  + *# (automatically stage tracked files).* No hace falta git add pero cuando se hace un Commit a un archivo se escribe git Commit –a –m seguido de ´ apostrofes y se escribe el comentario del Commit

**

* **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>**
  + Permite borrar los archivos que no deseemos.



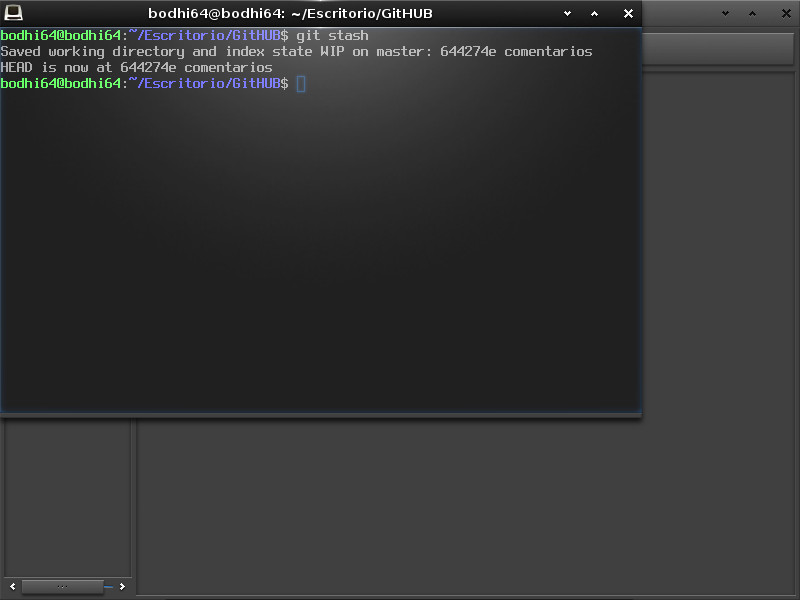
* **git rm -f <file>**
  + Si ya está modificado en el índex.
* **git mv <file> <renamed\_file>** 
  + Sirve para renombrar un archivo que se halla commiteado en el repositorio.
* **gitk**
  + tcl/tk. Herramienta gráfica para git, muestra una apariencia más amigable con el usuario, mostrando los commit que se hallan realizado, así como las ramas que contenga el repositorio.
* **git commit --amend**
  + Modificar el mensaje del último commit que se haya realizado en algún archivo del repositorio
* **git checkout -- <file>**
  + Descartar cambios en el directorio de trabajo y Mestra lo que se realizó.

## Configuration

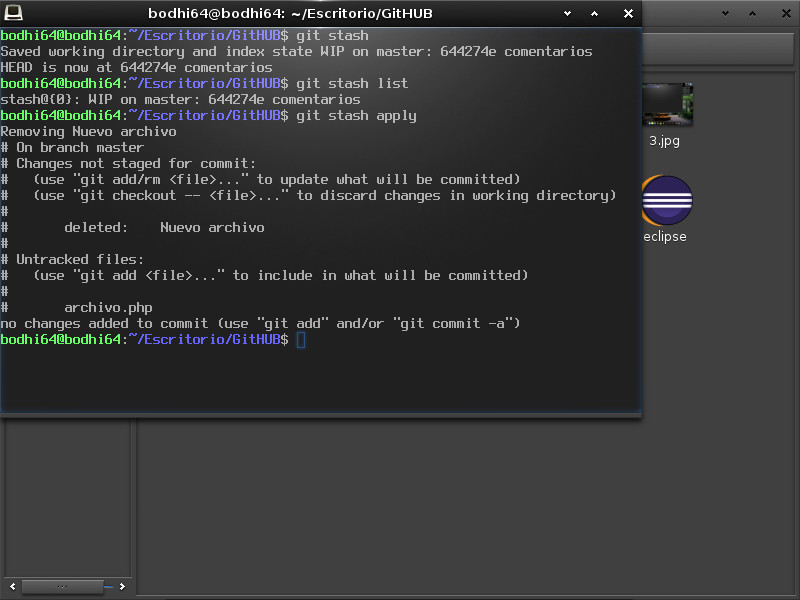
* + **git config --global <opción> <valor>**
    - Global para usuario, system todos y sin nada, específico para el repositorio.
  + **git config {key}**
    - Muestra el valor de la clave primaria
  + **git config --global core.editor <editor>**
    - Cambia el editor por defecto
  + **git config --global commit.template $HOME/.gitmessage.txt**
    - Plantilla para commits
  + **git config --global core.pager ‘more|less’**
    - Paginado por defecto, puedes usar cualquiera
  + **git config --global user.signingkey <gpg-key-id>**
    - Clave gpg para firmar tags
  + **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

## Stash (Esconder)

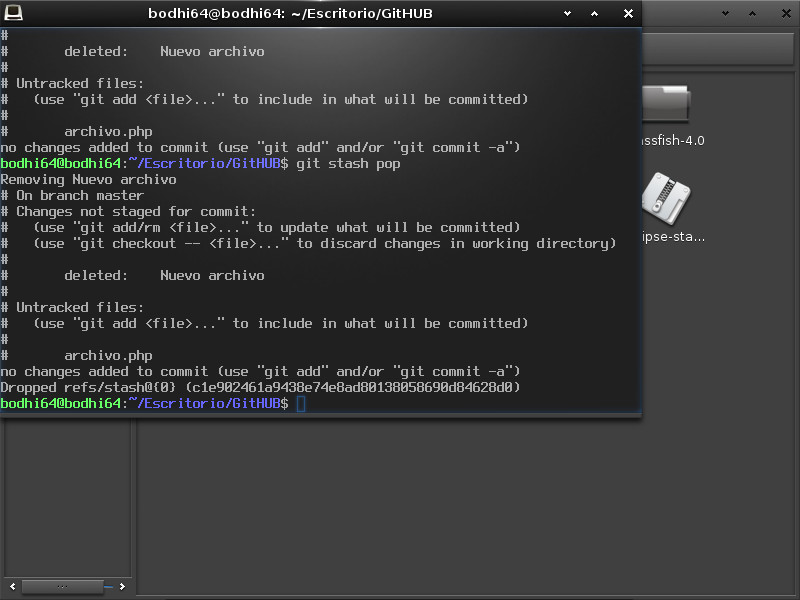
* **git stash**
  + Mediante este comando podremos guardar el estado en una pila y limpia el directorio para poder desplazarnos o cambiar de rama de la que estamos trabajando.



* **git stash list**
  + Muestra la pila, de acuerdo a los documentos que se estén desarrollando.
* **git stash apply**
  + vuelve al estado original del directorio. Stash{n} especifica uno concreto Y --índex reaplica los cambios stagged

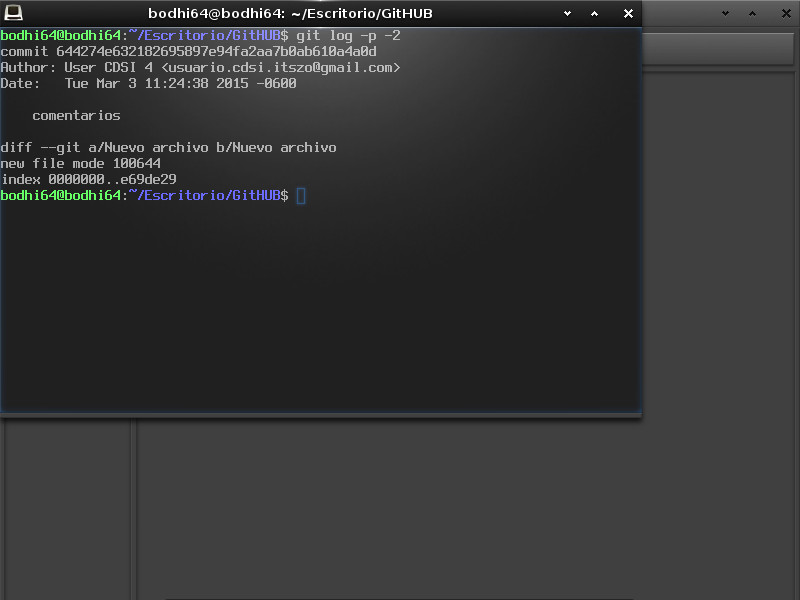


* **git stash pop**
  + Al trabajar en conjunto, mediante este comando se elimina el primero en la pila. O drop

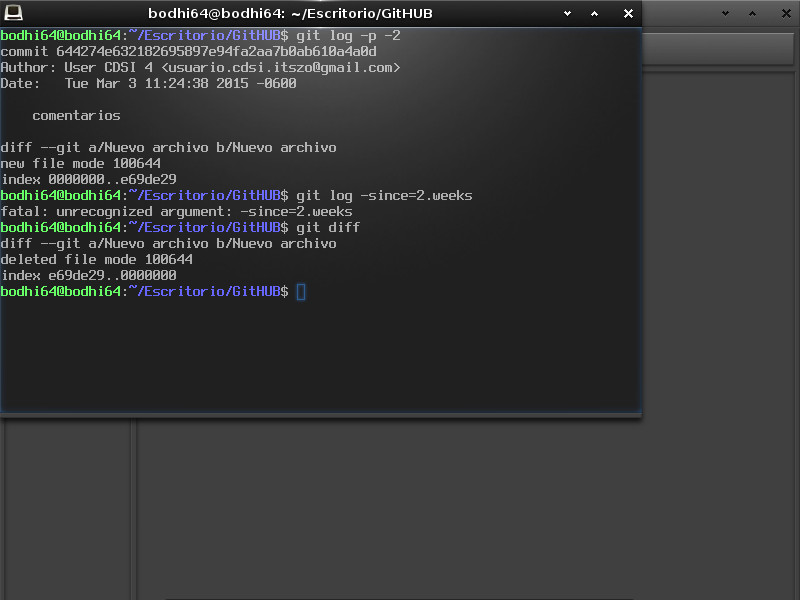


## Logs

* **git log -p -2**
  + Mediante este comando se Muestra los 2 últimos commits con diff en el editor



* **git log --since=2.weeks**
  + Muestra las diferencias o los cambios en los archivos con 2 semanas de antigüedad
* **git log <branch> --not master**
  + Muestra commit de <branch> que se han realizado, sin incluir los de master.
* **git diff master…contrib**
  + Muestra solo el trabajo que la rama contrib actual ha introducido desde su antecesor común con master



* **git log <branch1>...<branch2>**
  + Commits de branch2 que no están en branch1
* **git log origin/master...master**
  + Muestra qué commits se van a enviar al servidor

## REMOTES # REPOS EN INTERNET

* **git remote -v**
  + Muestra una lista los repos remotos a los que esta logeado
* **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 sobrescribe nada tuyo. ( *git pull sí hace merge automáticamente si se está realizando un seguimiento de esa branch*)
* **git remote show [remote-name]**
  + inspecciona el repositorio remotamente.
* **git remote rename <old-name> <new-name>**
  + Mediante este comando renombra branches: es decir si se aplica al repositorio que se está trabajando y en la rama en la que estamos colaborando quedaría así <new-name>/master

## AÑADIR VARIOS REPOSITORIOS REMOTOS

* + **git remote add** [**https://github.com/servidorCDSI2014/CDSI.git**](https://github.com/servidorCDSI2014/CDSI.git)
    - Añadir un nuevo repositorio remoto con el nombre deseado. Por ejemplo si ya tenemos uno en github y queremos añadir otro pero con otro grupo.
* **git push -u ejemplo\_de\_doc –all**
  + Subir el proyecto. A partir de ahora se puede seleccionar a qué repo publicar con **git push ejemplo\_de\_doc**

## Etiquetas

### Marcan puntos importantes en la historia del repo [(releases) Comunicados]

* **git tag**
  + Muestra las etiquetas actuales que se encuentran en los commit que se han realizado.
  + 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 show <tag-name>**
  + Muestra información asociada con la etiqueta que se le haya asociado.
* **git tag -v <tag-name>**
  + verifica tags firmadas que se hallan realizado en el repositorio
* **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 etiqueta seleccionada previamente.
  + **git push origin --tags**
    - * Enviar todas las etiquetas.
  + **git show <tag>**
    - * Devuelve más información sobre la etiqueta
  + **git tag -d nombre\_tag**
    - * Este comando se usa para eliminar la etiqueta que se ha seleccionado
  + **git push origin :refs/tags/nombre\_tag**
    - * Eliminar la etiqueta del repositorio remoto.

## BRANCH

* + **git branch <nombre-rama>**
    - Crea rama. Al Puntero al commit actual
  + **git checkout <nombre-rama>**
    - Este comando cambia la rama especificada. Para poder cambiar a otra.
  + **git checkout -b <nombre-rama>**
    - Crea y cambia de rama
  + **git merge <rama>**
    - Mezcla la rama actual con alguna otra Rama.
  + **git branch -d <rama>**
    - Elimina la rama que se ha seleccionado.
  + **git push origin --delete <branchName>**
    - Cuando se está trabajando con una rama en específico, y esa rama ya no es necesaria se utiliza este comando para elimina una rama del servidor.
  + **git mergetool**
    - Herramienta gráfica para resolver conflictos que se desarrollen ente una rama.
  + **git branch**
    - Muestra una lista de las ramas que se encuentren actualmente en el repositorio.
  + **git branch -v**
    - Mediante este comando se muestra una lista de las ramas mostrando en ellas el último commit que se ha hecho.
  + **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 en la rama actual.
  + **git branch --no-merged**
    - lista ramas que no han sido incorporadas a la actual.

## Remote branches

* **git fetch origin**
  + Descarga el contenido del servidor en la maquina donde se inicializa el repositorio.
* **git push <remote> <branch>**
  + Las ramas no se suben por defecto, has de subirlas explícitamente
* **git push <remote> <branch>:<nuevoNombre>**
  + de igual forma que la Anterior, pero en el servidor se llama a la rama con nuevoNombre en lugar de branch
  + Cuando se hace un git fetch que trae consigo nuevas ramas remotas, no se disponen de ellas localmente, solo se dispone de un puntero a la rama remota que no es editable. Para poder trabajar sobre esa rama, es necesario crearla Por ejemplo:
  + **git fetch origin**
    - Tras ejecutarlo, se mostrara la rama original
  + **git checkout -b rama\_nueva origin/rama\_nueva**
    - Crea una rama local a partir de la remota
  + **git merge origin/nueva\_rama**
    - Equivalente a la de arriba, pero sin establecer el tracking a la rama
* **git push [remotename] :[branch]**
  + Elimina una rama remota
* **git push [remotename] [localbranch]:[remotebranch]**
  + La rama en el servidor tiene distinto nombre a la local

## Seguimiento de Ramas

* **git checkout --track origin/rama**
  + Equivalente a -b rama\_nueva origin/rama\_nueva
* **git chekout -b <nuevo\_nombre> origin/<rama>**
  + Establece un nombre distinto para la rama local

## Cambios y Fusión

# Rebase y merge se diferencian en que merge mezcla dos puntos finales de dos snapshots y rebase aplica cada uno de los cambios a la rama en la que se hace el rebase. No lo uses en repos publicos con más colaboradores, porque todos los demás tendrán que hacer re-merges

* + **git rebase master**
    - Aplica todos los cambios de <una rama> a master
    - **git checkout master**
    - **git merge client**
    - fast-forward. Cliente y master en el mismo snapshot Si se quiere aplicar también los cambios de server, basta con:
  + **git rebase [basebranch] [topicbranch]**
    - Sintaxis de rebase

## Servidor

* **git instaweb**
  + Muestra una interfaz web con los commits

## Generar Un Número De Compilación (Build Number)

* + **git describe master**
    - Solo funciona para tags creadas con -s ó -a

## Generar Un Changelog

* + **git shortlog --no-merges master --not <tag>**
    - Recopila todos los commits desde <tag> y los agrupa por autor

## Recomendaciones

* + **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 Proyectos Ajenos, Mediante Fork

Este procedimiento deberá realizarse en la consola de la maquina donde se inicialice GitHub

* + **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 práctica tener siempre una rama master que apunte a origin/master, para estar siempre actualizado con los últimos 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 automáticamente. Así puedes realizar todos los cambios necesarios y luego hacer el commit

## 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 determinar 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 está 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 contenido de otro proyecto.
  + **Clonar un repo con su módulos**
  + **git clone url**
  + **git submodule init**
  + **git submodule update**

## Bibliografía

<http://elbauldelprogramador.com/mini-tutorial-y-chuleta-de-comandos-git/>