**Mini Guia. Resumen de comandos git y configuración global y local**

**Configuración de git.**  
**Listar configuración global:**  
git config --global --list

**Configurar nombre global:**  
git config --global [user.name](http://user.name) “Fernando Daniel Jaime”  
**Configurar email global:**  
git config --global user.email [tu.mail@gmail.com](mailto:tu.mail@gmail.com)  
**Configurar colores de git:**  
git config --global color.ui true

**Listar configuración local:**  
git config --local --list

**Cambiar editor de texto por default para el repositorio actual:**  
**Sublime (Escribir descripción, cerrar guardando)**  
git config --local core.editor "subl --wait"  
**Vim (guardar :wq, no guardar :q!, <esc>para escribir)</esc>**  
git config --local core.editor vim  
**Atom (Escribir descripción, cerrar guardando)**  
git config --local core.editor "atom --wait"  
**Nano (Escribir descripción, cerrar guardando)**  
git config --local core.editor nano  
**Emacs (Guardar Ctrl-x + Ctrl-s, Salir Ctrl-x + Ctrl-c)**  
git config --local core.editor emacs

**Superlog de Mike:**  
git config --global alias.superlog “log --graph --abbrev-commit --decorate --date=relative --format=format:’%C(bold blue)%h%C(reset) - %C(bold green)(%ar)%C(reset) %C(white)%s%C(reset) %C(dim white)- %an%C(reset)%C(bold yellow)%d%C(reset)’ --all”

**Comandos mas comunes:**  
git init  
git status  
git log  
**Log ultimos n commits:**  
git log -2  
**Log resumido con ramas:**  
git log --oneline --graph

**Previamente configurado globalmente:**  
git superlog

**Agregar archivo a staging:**  
touch file.txt  
git add file.txt  
**Agregar todo**  
git add .  
**Remover archivo de staging**  
git rm --cached file.txt  
**Remover archivo de staging y eliminarlo**  
git rm -f file.txt  
**Verificar si el archivo existe pero que no haga nada:**  
git add -n file.txt

**Commit de archivos (pasan al repositorio):**  
git commit -m “mensaje descriptivo del cambio”  
**Concatena nuevos cambios con commit previo:**  
git commit --amend

**Versionado de commits (version tag):**  
**Etiqueta el último commit:**  
**Light tag version:**  
git tag 1.0  
**Version anotada:**  
git tag -a 1.0 -m “Versión estable del proyecto”  
**Etiquetar commits anteriores (usando el sha1):**  
git tag 0.5 293b8305fc85e08a55fb4d96daf46e7f03d0c4ed  
**Borrar un tag:**  
git tag -d 0.5  
**Renombrar un tag (usando el sha1):**  
git tag -f -a 0.3 -m “descripcion 0.3” 293b8305fc85e08a55fb4d96daf46e7f03d0c4ed

**Ver cambios a partir desde ese commit hasta el actual:**  
git diff [SHA1]  
**Ver cambios entre 2 commits:**  
git diff [SHA1-v1] [SHA1-v2]

**Quitar todos los commits sin incluir el del SHA1 dado:**  
**Los cambios regresan a staging:**  
git reset --soft [SHA1]:  
**Los cambios además se quitan de staging:**  
git reset --mixed [SHA1]  
**Quita de staging algo puntual:**  
git reset HEAD file.txt  
**Los cambios además se eliminan del working directory:**  
**Elimina todo lo que este en staging (no toca untracked files):**  
git reset --hard  
**Elimina todos los commits sin incluir el del SHA1 dado:**  
**(Por si acaso, guardar el log haciendo: git superlog > log.txt)**  
git reset --hard [SHA1]  
**Recuperar cambios eliminados (usar log.txt para volver al commit buscado):**  
git reset --hard [SHA1]

**Listar ramas:**  
git branch -l  
**Crear rama llamada nombre1:**  
git branch nombre1  
**Eliminar rama:**  
git branch -d nombre1  
**Eliminar rama que contiene commits:**  
git branch -D nombre1  
**Renombrar una rama (nombre1 ahora se llama nombre2):**  
git branch -m nombre1 nombre2  
**Moverse entre ramas (listar ramas para ver cuales hay):**  
git checkout gh-pages  
**Moverse entre commits (usar superlog para ver los sha1):**  
git checkout sha1  
**Crear una rama y navegar esa rama al mismo tiempo:**  
git checkout -b nuevo-branch

\*\*Ejemplo: \*\*  
**Estoy en la rama funcionalidad1 y necesito hacer un fix:**  
**Me paso a la rama master:**  
git checkout master  
**Luego hago un branch para el hotfix desde master:**  
git checkout -b hotfix  
**Finalmente resuelvo el problema en la rama hotfix**

**Ejemplo Merge:**  
**Ir a la rama que quiero que obtenga los cambios:**  
git checkout master  
**Hacer el merge de las otras ramas:**

**Lo primero en mergearse es en modo fast forward dado que estoy basado de master y no hubo merge intermedios:**  
git merge hotfix

**Lo siguiente en mergearse (al no basarse del último merge) va a ser en modo recursive/auto-merge (requiere un commit adicional):**  
git merge RD

**Tenemos un merge con conflictos dado que ambas ramas modifican el mismo file index.html. Se hace un merge en modo manual:**  
git merge nueva-imagen  
**Corregir el index.html y mandarlo a staging:**  
git add index.html  
**Comitir index.html:**  
git commit  
**Aceptar o cambiar el mensaje en el editor de texto y el merge queda finalizado**

**Rebase:**  
**Al mergear recursivamente y de modo manual se agregan 2 commits adicionales. Con git rebase se evita esto, pero se afecta el orden de la historia dado que primero hace un rewind del head y aplica updateB y luego le aplica updateA.  
Ejemplo:**  
git checkout -b updateA  
touch updateA  
git add -A  
git commit -m “Agregando el update A”  
git checkout master  
git checkout -b updateB  
touch updateB  
git add -A  
git commit -m “Agregando el update B”  
git checkout master

git merge updateA  
git rebase updateB  
**En el log updateB va a figurar anterior al updateA**

**Rebase modo interactivo da opciones para el commit**  
git rebase -i updateA  
**Por ejemplo: opción reward. Guardo y cierro.  
En la consola el rebasing no esta terminado aun, muestra el mensaje rebasing(1/1) y tengo un nuevo commit abierto en mi editor de texto  
Le cambio el nombre al commit: “Agregando update B por segunda vez”  
Y queda finalizado el rebase**

Opinion personal: usar merge.

Stash  
**Si tengo cambios no agregados y necesito cambiar a otro branch puedo hacer git stash para guardar el estado del working directory y el índice en mi brach actual.**

git stash  
\*\*Si luego hago git status, ya tenemos el working directory limpio y me puedo ir a otro branch libremente.

Para ver la lista de los stash que tengo:\*\*  
git stash list  
**Para eliminar un stash de la lista:**  
git stash drop stash@{1}  
**Para aplicar el último cambio que hicimos:**  
git stash apply  
**Para aplicar un stash en particular:**  
git stash apply stash@{3}

Cherry-pick  
**Ejemplo: tengo un hotfix que hice en mi branch actual entre medio de varios otros commits y necesito mergear mi hotfix2 con master:**  
git checkout master  
git checkout -b hotfix2  
**Selecciono el sha1 de mi hotfix y lo traigo con git cherry-pick:**  
git cherry-pick sha1  
git checkout master  
git merge hotfix2

**Asociar un repositorio a uno local:**  
**Por convención el nombre del repositorio siempre lo llamamos origin:**  
git remote add origin [git@github.com](mailto:git@github.com):username/miProyecto.git  
**Para verificar que nuestro repositorio esté enlazado:**  
git remote -v  
**Para quitar la asociación al repositorio de github:**  
git remote remove origin

**Traer de mi repositorio origin la rama master:**  
git fetch origin master  
**Mergear origin/master con mi master local:**  
git merge origin/master --allow-unrelated-histories

**Trear origin y mergearlo con la rama master en un solo paso:**  
git pull origin master

**Subir mi codigo al repositorio de github en rama master (lo correcto es push a otro branch y luego mergear con master via pull request luego de aprobar el review):**  
git push origin master  
**Subir mis tags al repositorio de github en rama master:**  
git push origin master --tags  
**Subir la rama funcionalidad1 al repositorio de github:**  
git push origin funcionalidad1