Git (sistema de control de versiones)

<https://www.youtube.com/watch?v=YUCwr1Y6bFI&t=2721s>

## 3 Estados

* Comited: En base de latos local
* Modified: Modifcado, solo en el sistema de archivos de mi pc, no de git
* Staged : Guardado en el sistema de archivos de git, pero no en el repositorio local

## Comandos básicos

*Add :* guarda cambios para ser commit

*Checkout:* descarta cambios

Commit –m para introducir mensaje

Git log para historial de commits, para salir con q

Git reset HEAD –archivo- devuelve al estado del head //reversa de commit

Git diff –archivo- muestra las diferencias entre tu directorio de trabajo y el área estable.

## Comandos ramas

Git branch –b *nombre* para borrar branch.

Git checkout –b “*nombre”* para crear branch y acceder a ella.

Git branch *nombre*  crea la branch pero no se cambia.

Git push origin *nombre* sube los cambios a *nombre*(server)

Git branch –list lista de branchs

Git blame *archivo* para ver el historial de notificaciolnes

Git commit –m “-mensaje-” hacer un commit con mesaje incluido

Git remote –v ver las referencias del repositorio al que apunta

Git remote add origin *link*  para cuando no tiene referencia

Git remote set-url origin *link* para cambiar de referencia

Git push –u origin master para que el master local apuente al master remoto

Borrar branch remota git push origin :*nombre*

Cuando un pull request se merge a develop es necesario notificar a todos los desarrolladores para que jalen los cambios:

1. Cambiarme de la branch en la que estoy trabajando a develop.
2. Pull para actualizar develop local desde el servidor.
3. Cambierme al branch en el que estoy trabajando.
4. Merge o rebase desde develop local a la branch que estoy trabajando para integrar los cambios.

## Comandos bajo el agua plumbers commands

1. Nuevo directorio
2. Git init para convertir el foder en un repositorio
3. Touch hello\_world.txt
4. Modificar hello\_world.txt
5. git hash-object -w hello\_world.txt para crear el blop
6. Find .git/objects –type f
7. git cat-file -p 5c1b14949828006ed75a3e8858957f86a2f7e2eb imprime el contenido del archivo
8. modifica hello world
9. git hash-object -w hello\_world.txt para crear referencia del blop (-w para guardarlo)
10. git update-index --add hello\_world.txt
11. git ls-files –stage muesta los archivos en el stage y el hash de la última modificación
12. nano foo\_bat.txt
13. git update-index --add foo\_bar.txt lo pasa a stage
14. git write-tree crua un nuebo objeto tree que genera un snapshot del index
15. echo 'first commit' | git commit-tree f572c9a0e8f137e898d8e2245ad68a88f85e2e5b para crear el commit (hash del objeto tree) y arroja un nuevo hash que referencia el commit pero no se ha ligado a una branch
16. modificamos el archivo fooBar.txt
17. git update-index --add foo\_bar.txt
18. git write-tree para un nuevo snapshot (objeto tree)
19. echo 'second commit' | git commit-tree 3ec7609cf3d8dbb8c555a5ce2f956f28ea60d93f -p 19cc41d03310a0b21dd4c6bf2bc3c3dc6c28e341 (con –p se agrega el commit padre)
20. git log –stat ver logs
21. git update-ref refs/head/master 36340fa9eaf2b818b8f1e0f773477129d577343 hace referencia del ultimo commit a la brach master
22. creamos una nueva branch
23. cat .git/HEAD muestra la ruta donde se guarda la branch
24. git count-objects –H contar objetos
25. git gc garbage elimina la basura
26. git count-objects –H contar objetos
27. find .git/objects -type f muestra los objetos con su tipo
28. git verify-pack -v .git/objects/pack/pack-e917d279c1d266288b06315def8a00b1fe0300a5.idx para ver que tiene un paquete

blop es donde se guarda todo el contenido de los archivos de los repositorios

las branches solo son referencias al último que se hizo desde esa branch y se guardan en .git/refs/head/ (per ejemplo .git/refs/head/master)

# directed acyclic graph (DAG)

merkle tree merkle dag

De acuerdo con el flujo, los hijos siempre tienen referencia al padre, pero los padres no tienen referencia de los hijos.

.

|-- COMMIT\_EDITMSG

|-- FETCH\_HEAD

|-- HEAD

|-- ORIG\_HEAD

|-- branches

|-- config

|-- description

|-- hooks

| |-- applypatch-msg

| |-- commit-msg

| |-- post-commit

| |-- post-receive

| |-- post-update

| |-- pre-applypatch

| |-- pre-commit

| |-- pre-rebase

| |-- prepare-commit-msg

| `-- update

|-- index

|-- info

| `-- exclude

|-- logs

| |-- HEAD

| `-- refs

|-- objects

`-- refs

|-- heads

|-- remotes

|-- stash

`-- tags

Analicemos algunos de los archivos que puedes ver normalmente en el directorio .git:

COMMIT\_EDITMSG: Contiene la descripción del último commit. En realidad este archivo no es utilizado por Git, sólo existe para fines informativos.

config: Contiene las opciones de configuración de este repositorio. La configuración específica a este repositorio se define en este archivo (incluso los [aliases](http://gitready.com/intermediate/2009/02/06/helpful-command-aliases.html)!) El uso más común de este archivo es especificar la ubicación de cada repositorio remoto y definir algunas opciones claves, como ser si tu repositorio posee o no un área de trabajo.

description: Si utilizas [gitweb](http://git.or.cz/gitwiki/Gitweb) o ejecutas git instaweb, esta descripción se mostrará cuando estés viendo tu repositorio o la lista de repositorios.

FETCH\_HEAD: Los SHAs de todas las ramas que fueron actualizadas por el último git fetch.

HEAD: El ref actual. En la mayoría de los casos éste es probablemente refs/heads/master.

index: El área de [staging](http://gitready.com/beginner/2009/01/18/the-staging-area.html) conteniendo información acerca de los archivos que serán incluídos en el próximo commit.

packed-refs: refs inactivos. Esta no es la lista definitiva de refs en tu repositorio (¡el directorio refs tiene la lista real!) Mira el comentario de gitster para mayor información sobre este tema.

ORIG\_HEAD: Durante una integración, este es el SHA de la rama en la que se está realizando la operación.

MERGE\_HEAD: Durante una integración, este es el SHA de la rama desde la cual se están tomando los cambios a integrar.

MERGE\_MODE: Se utiliza para pasar opciones de un git merge al siguiente git commit cuando ocurre un conflicto durante la integración y se requiere un git commit para finalizar la operación. Actualmente --no-ff es la única opción que se pasa de esta forma.

MERGE\_MSG: Enumera los conflictos que se produjeron en el último git merge.

RENAMED-REF: Aún estoy intentando analizar este archivo. Aparentemente, está relacionado con errores ocurridos al grabar refs.

También hay unos cuantos directorios:

hooks: Un directorio que rapidamente se convertirá en tu mejor amigo: contiene distintos scripts que son ejecutados en determinados momentos al trabajar con Git, como por ejemplo después de cada commit o antes de cada rebase. Una serie completa de artículos acerca de los hooks está en camino.

info: Relativamente sin importancia, a excepción del archivo exclude que se encuentra dentro de este directorio. Ya hemos analizado este archivo en el artículo [ignorando archivos](http://gitready.com/beginner/2009/01/19/ignoring-files.html), pero para refrescar la memoria: puedes usar este archivo para ignorar archivos en este proyecto. Eso sí, ¡ten cuidado! Este archivo no está versionado como sí lo estaría un archivo.gitignore.

logs: Contiene la historia de las distintas ramas. Aparentemente se lo utiliza principalmente con el comando [reflog](http://gitready.com/intermediate/2009/02/09/reflog-your-safety-net.html).

objects: El depósito interno de objetos, indexados por SHAs.

rebase-apply: La base para el comando [rebase](http://gitready.com/intermediate/2009/01/31/intro-to-rebase.html) y también para git am. Si te atreves, es posible ajustar el archivo patch dentro de este directorio.

refs: La copia maestra de todos los refs que existen en tu repositorio, ya sean stashes, tags, ramas locales o remotas.

# Git bajo las sabanas:

porcelan commands

Plumber commands

Al hacer un git add se cra un nuevo blob.

Al hacer un commit se crea un tree