GIT MANUAL BÁSICO

Control de versiones

Nos permite:

- Gestionar la información de múltiples versiones
    - Código / tests
    - Archivos de configuración
    - Esquemas de BD
    - Documentación
    - Etc.
- De una forma estructurada y estandarizada
    - Commit
    - Checkout
    - Update
    - Trunk
    - Branch
    - Tag
- ¿Por qué necesitamos control de versiones?
    - Permite colaboración entre desarrolladores.
    - Versión principal (trunk)
    - Versiones paralelas (branching)
    - Facilita la gestión de releases (tags)
    - Nos permite retroceder versiones
    - El commit es una operación atómica
    - Integración con otras herramientas
    - Muchos desarrolladores creando cientos/miles de líneas de código.
    - El código es 'comiteado' (commit) a un repositorio central.
    - Los conflictos disparan alertas.
    - Gestión de acceso a través de credenciales.
    - Se puede definir usuarios y grupos.
    - Diferentes versiones pueden coexistir.

Trunk:

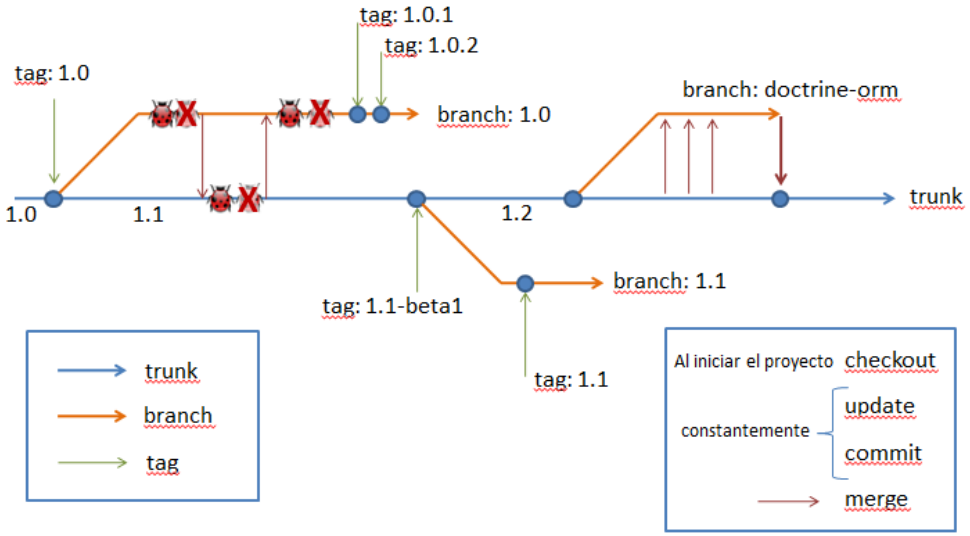
- ✓ Área principal de desarrollo.
- ✓ Acá es donde tu siguiente versión mayor reside.
- ✓ Generalmente es donde están las últimas funcionalidades.

Branches:

- ✓ Cada vez que lances una versión mayor, se le crea un nuevo branch.
- ✓ Esto te permite hacer bug fixes y hacer un nuevo release sin tener que sacar una versión con posibles features sin terminar.
- ✓ También existen branches que se hacen por algún cambio importante que se hace, con la finalidad de que no interfiera con el resto del desarrollo. Se llaman branches de aislamiento.

Tag:

- ✓ Cada vez que tu sacas una nueva versión (final release, release candidates (RC), y betas) debes hacerle un tag.
- ✓ Esto nos da estado del código en un punto en el tiempo.
- ✓ Y nos permite volver a una versión anterior y poder reproducir cualquier bug si fuera necesario.



Principales comandos en GIT

git clone <repo>
Clona un repositorio.

Ejemplos:
git clone /home/eanaya/repos/demo
git clone git://dev.myapp.com/git/myapp.git
git clone git+shh://dev.myapp.com/git/myapp.git
git clone https://dev.myapp.com/git/myapp.git
Cuando hacemos un *clone*, git crea un remote llamado *origin*

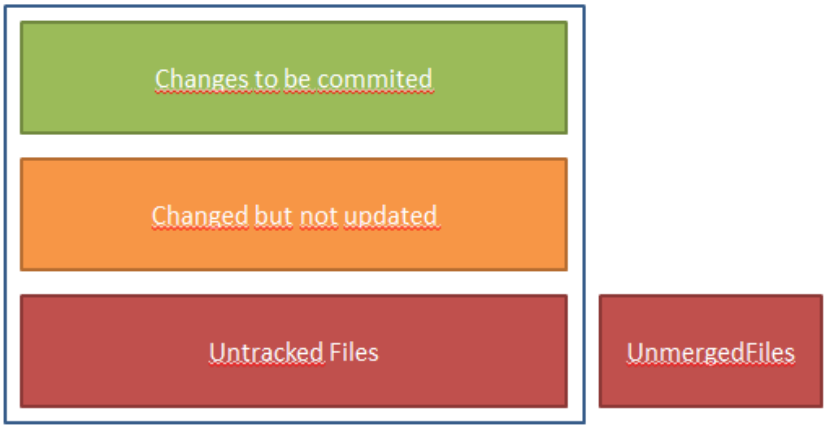
git status

Obtienes el estado de tu repo:

- Archivos para ser 'comiteados' (Changes to be committed)
- Archivos que no van a ser 'comiteados' (Changed but not updated)
- Archivos nuevos (Untracked files)

Ejemplos:
git status

GIT AREAS



git add <file>

Agrega uno o varios archivos al staging-area, es decir, los marca para ser commiteados.

Ejemplos:
git add config.php
git add *.php
git add src/library/*
git add --all

git commit –m "Mi commit message"

Realiza un commit en el repositorio. (localmente)

Ejemplos:
git commit –m "Mi commit message"

git pull <remote> <branch>

Pull ("jalar") realiza 2 operaciones simultaneas: git fetch + git merge.
Fetch obtiene los cambios remotos.
Merge une estos cambios con nuestros cambios.

Ejemplos:
git pull (por defecto toma el remote origin y obtiene todos los branches)
git pull origin devel

git push<remote> <branch>

Push (“empujar”) envia nuestros commit al repo remoto.

Ejemplos:
git push (por defecto toma el remote origin y obtiene todos los branches)
git push origin devel

git checkout <branch>

Cambiamos de una rama a otra.

Ejemplos:
git checkout devel (Cambiamos a la rama previamente ecistente: devel)
git checkout -b devel (Creamos la rama devel y nos cambiamos a esa)

git checkout <branch>

Cambiamos de una rama a otra.

Ejemplos:
git checkout devel (Cambiamos a la rama previamente ecistente: devel)
git checkout -b devel (Creamos la rama devel y nos cambiamos a esa)

git merge <branch>

Unimos los commits hechos en la rama <branch> con los hechos en la rama actual.

Ejemplos:
git merge devel
(Asumiendo que estoy en la rama release, con este commando estaré uniendo los commits nuevos hechos en la rama devel con los hechos en la rama release)

simple daily git workflow

git pull

pull all the changes from the remote repository

git checkout -b branch-name-here

create a new branch for your bug/feature/issue

DO YOUR WORK HERE

keep it in small chunks, the smaller your commits the better, in case things go wrong

git add .

and any new files you've created

git status and/or git diff

see the changes you're going to commit

git commit -m "Detailed message here"

make the commit with a nice detailed message

commit loop

git checkout master

switch back to the master branch when the feature is done, your tests pass right?

git merge branch-name-here

update the master branch to update the master with all your changes

feature loop

git push

send your changes up to the remote repository