git

Control de Versiones

NÉSTOR ANAYA CHAVEZ

¿Qué es GIT? ¿Y Github?

GIT es un software diseñado por Linus Torvalds que permite la gestión y el control de versiones de una aplicación Software. Esta se encuentra disponible en dos versiones:

- Versión CLIENTE.
- Versión SERVIDOR.

¿Y Github?

GITHUB es un servicio en la nube cuyo fin es alojar en sus servidores (Servidores GIT) repositorios GIT de manera tal que un proyecto pueda accederse en cualquier momento y desde cualquier lugar. El servicio de GITHUB es gratuito para todo repositorio de acceso público. Repositorios privados requieren el pago de un arancel.


github
SOCIAL CODING

Caracteristicas GITHUB

- Issue Tracking (Seguimiento de Incidencias).
- Soporte para Milestones (Hitos) y Labels (Etiquetas).
- Soporte para palabras claves en Commits (closes,fixes).
- Soporte para discusiones detalladas acerca de todos y cada uno de los commits realizados (por línea y por commit en su totalidad). ● Ver Ejemplo

¿Existen Alternativas a GITHUB?


Blit


GITORIOUS


Atlassian
Bitbucket


GitLab

Conceptos Básicos

Conceptos

Repositorio: Se denomina al sitio donde se almacenan los archivos del proyecto en forma centralizada.

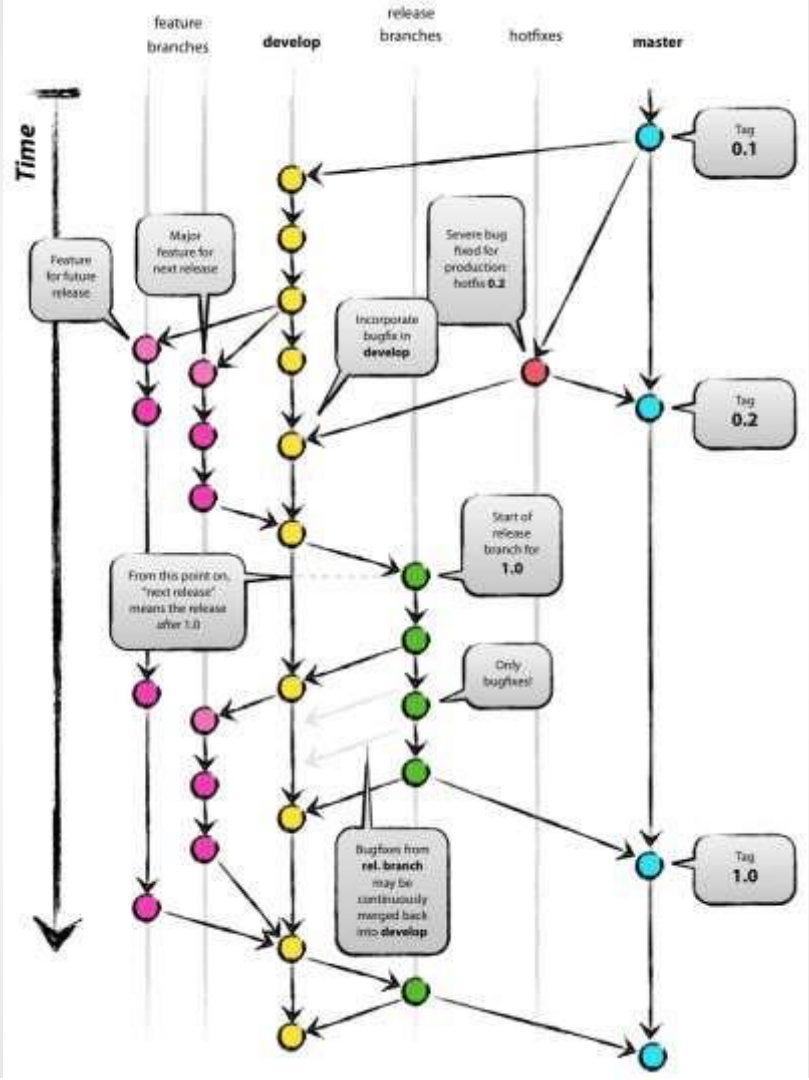
Commit: Consignación de un conjunto de cambios. Un commit genera una nueva versión. La misma tiene asociado un conjunto de cambios.

Conceptos

Branch: Es una ramificación de un proyecto. De forma ideal (los proyectos open-source de hecho lo implementan) todo proyecto debería tener cuatro bifurcaciones:

- Master
- Development
- Features

- Hotfixes


feature branches
develop
release branches
hotfixes
master
Time
Tag 0.1
Feature for future release
Major feature for next release
Severe bug fixed for production: hotfix 0.2
Incorporate bugfix in develop
Tag 0.2
Start of release branch for 1.0
From this point on, "next release" means the release after 1.0
Only bugfixes!
Bugfixes from rel. branch may be continuously merged back into develop
Tag 1.0

Conceptos

Master: Es la rama principal. Contiene el repositorio que contiene la versión de la aplicación que se encuentra en producción, por lo que debe estar siempre en un estado “estable”.

Conceptos

Development: Es un ramificación de master. Es la rama de integración de todas las nuevas funcionalidades. Luego que se realice la integración y se corrijan los errores (en caso de haber alguno), es decir que la rama se encuentre en un estado “estable”, se puede hacer una fusión entre las ramas de development y la rama master.

Conceptos

Features: Cada nueva funcionalidad se debe realizar en una rama nueva, específica para esa funcionalidad. Estas se deben sacar de development. Una vez que la funcionalidad se encuentre “cocinada”, se hace un fusión de la rama especifica de dicha funcionalidad sobre la rama development, donde se integrará con las demás funcionalidades.

Conceptos

Hotfix: Esta rama se utiliza para solucionar bugs que surgen en la aplicación que se encuentra en producción, por lo que se deben arreglar y publicar de forma urgente. Es por ello, que son ramas que bifurcan de master. Una vez corregido el error, se debe fusionar la misma sobre master. Al final, para que no quede desactualizada, se debe realizar el merge de master sobre development.

Conceptos

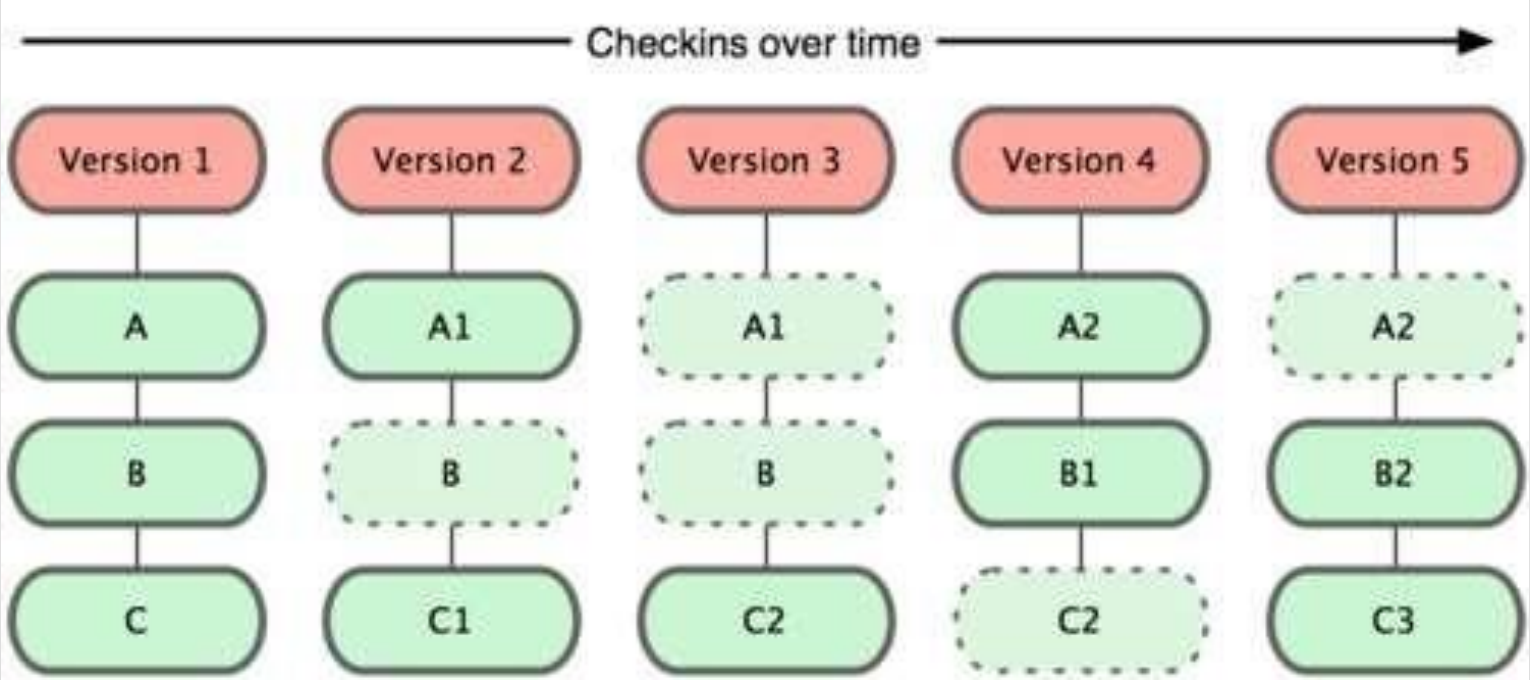
Merge: Es la fusión o mezcla de dos ramas del proyecto.

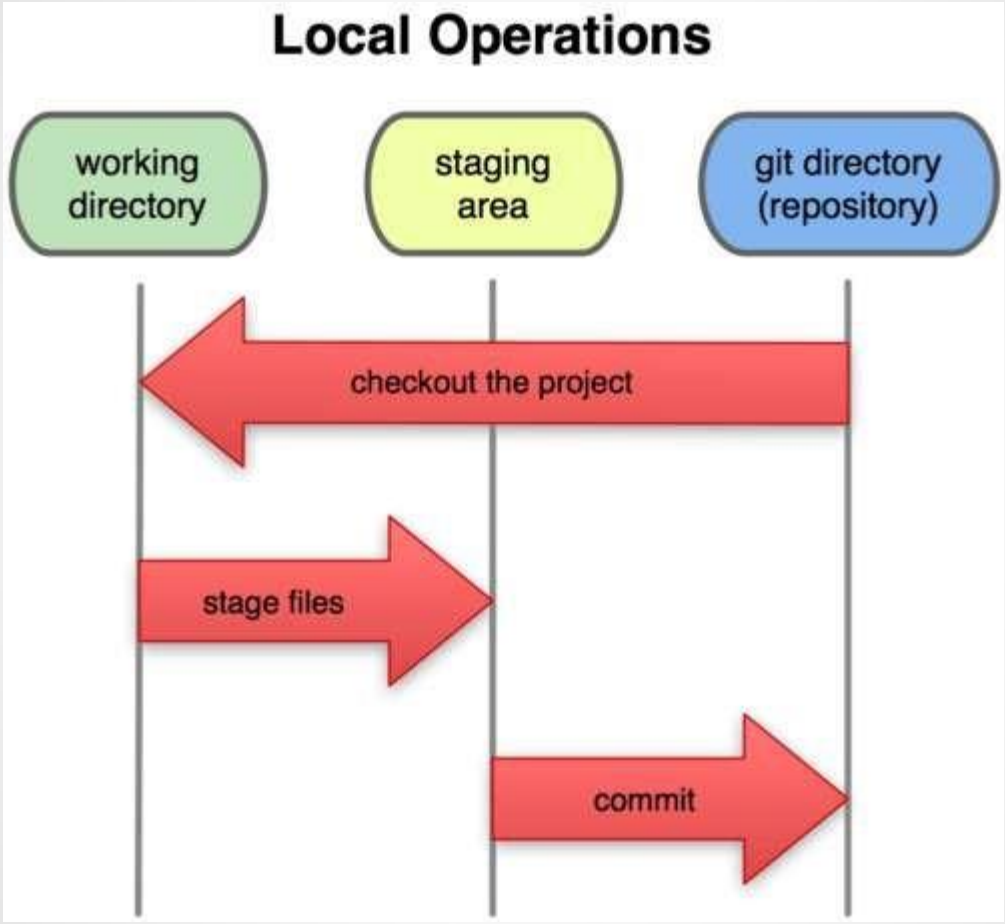
Staging Area: Área de preparado de cambios.

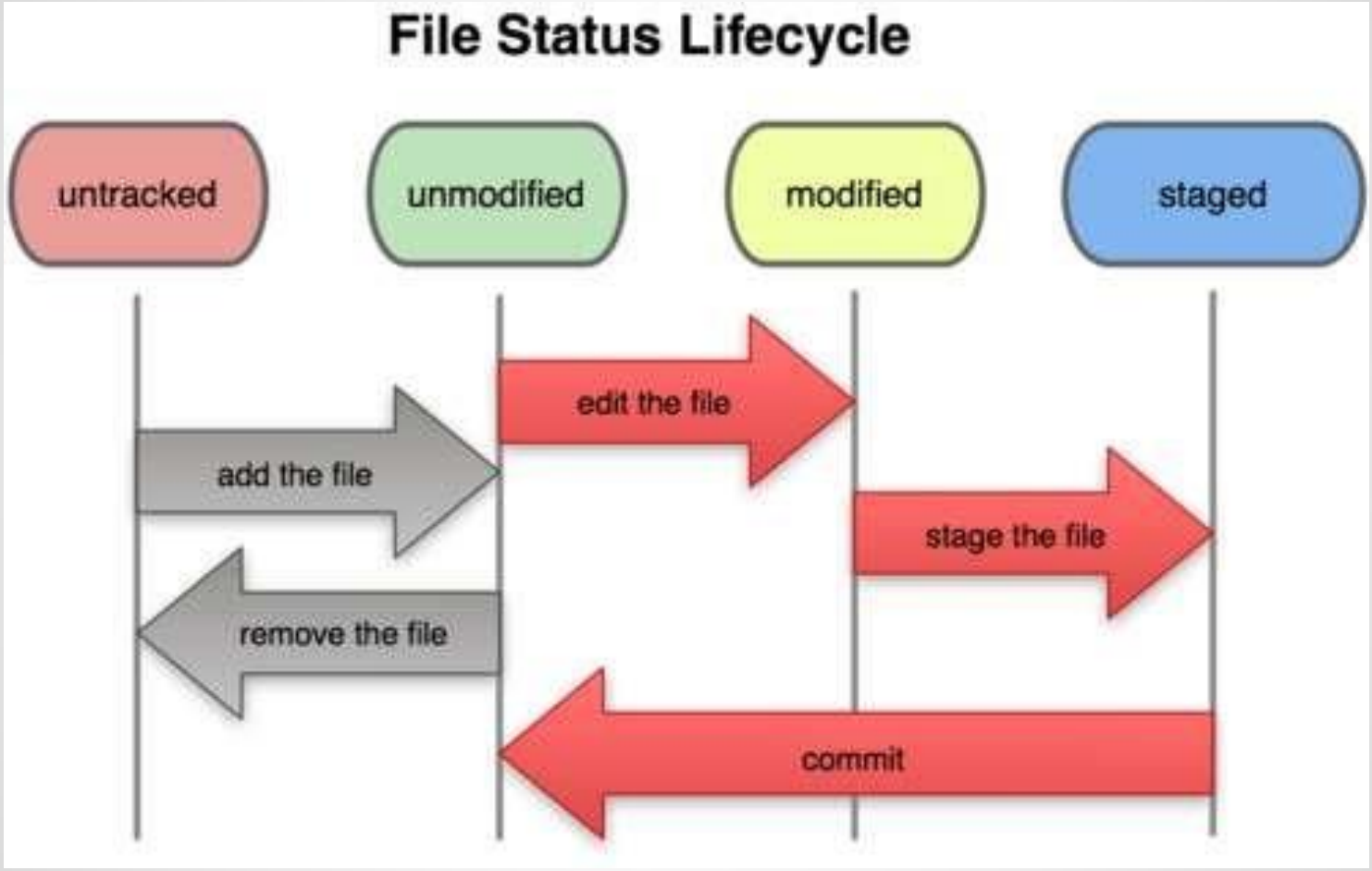
Snapshot: Imagen/Fotocopia del estado actual de un proyecto.

HEAD: Cabecera que apunta al último snapshot (commit) realizado.

Funcionamiento de GIT


Checkins over time
Version 1
Version 2
Version 3
Version 4
Version 5
A
A1
A1
A2
A2
B
B
B
B1
B2
C
C1
C2
C2
C3


Local Operations
working directory
staging area
git directory (repository)
checkout the project
stage files
commit


File Status Lifecycle
untracked
unmodified
modified
staged
edit the file
add the file
stage the file
remove the file
commit

Acciones Básicas

Acciones Básicas

clone: Clonar un repositorio.
init: Crear un repositorio en forma local.
add: Agregar un documento a un área de preparación de cambios (Staging area).
commit: Consignar un conjunto de cambios.
push: Subir al servidor el/los cambios realizados (Una nueva versión).

Acciones Básicas

pull: Descargar y actualizar los cambios realizados en el repositorio remoto.

checkout: Crear una nueva rama/bifurcación del proyecto (repositorio) actual.

touch: Crear un archivo. **reset:** Cambiar el estado del repositorio a un estado anterior.

Acciones Básicas

diff: Ver los cambios realizados entre dos versiones (Línea por línea).
log: Ver un log de los cambios realizados.
merge: Unificar, mezclar cambios realizados en dos ramas y/o bifurcaciones del proyecto.
branch: Lista, crear o eliminar ramas y/o bifurcaciones del proyecto.

Primeros Comandos

Primeros Comandos

Configurar datos del usuario:

~$ git config --global user.email “fperez@ejemplo.com”
~$ git config --global user.name “Fulanito Perez”

Iniciar un repositorio local:

~$ cd D:
~$ cd ruta/a/mi/proyecto
~$ git init

Primeros Comandos

Sincronizar repositorio remoto en el directorio actual:

~$ git remote add origin http://url/del/repositorio/git.git

En este punto estamos listos para agregar nuestros archivos y editarlos hasta que estemos seguros de que deseamos sincronizar el repositorio local con el remoto.

Primeros Comandos

El archivo .gitignore se utiliza para definir la lista de recursos que no se deben incluir en la lista de recursos “listos para agregarlos a la staging area”.

Crear el archivo .gitignore:

~$ touch .gitignore

Realizar un Commit

Realizar un commit

Listar archivos ready to stage:

~$ git status

Agregar archivos de la lista a la staging area.

~$ git add mi_archivo.txt

Enviar commit con un comentario.

~$ git commit -m “Este es el primer commit.”

~$ git commit -am “Este es el primer commit.” // No necesita del comando “git add”.

Realizar un commit

Listar la lista de commits realizados.

~$ git log | ~$ git log -n <numero_max_commits>

Resultado (Lista de N commits):

commit 6cdae7e324d73331ab7668a667a1cf3a8a6fa369
Author: Fulanito Perez <fperez@ejemplo.com>
Date: Wed Apr 30 10:17:54 2014 -0300
Agrego modificaciones sobre...

Push commits

Push Commits - Subir al repo.

Listar la lista de commits realizados.

~$ git push -u origin <<nombre_branch>>

Por defecto siempre se trabaja y se sincroniza en el branch “master”.

~$ git push -u origin master

Push Commits - Subir al repo.

Listar la lista de commits realizados.

~$ git push -u origin <<nombre_branch>>

Por defecto siempre se trabaja y se sincroniza en el branch “master”.

~$ git push -u origin master

Gestión Branches

Gestión Branches

Crear un branch (Ramificación):

~$ git branch <<nombre_branch>>

~$ git branch development //Ejemplo

Crear un branch y cambiar al mismo:

~$ git checkout -b <<nombre_branch>>

~$ git checkout -b development //Ejemplo

Gestión Branches

Listar todas las ramas locales:

~$ git branch //Locales.

~$ git branch -a //Locales y remotas.

Eliminar una rama:

~$ git branch -d <<nombre_rama>>

~$ git branch -d feature_ABMCliente //Ejemplo

Gestión Branches

Fusionar las ramas master y hotfix.

~$ git checkout master //cambiar a la rama master.

~$ git merge hotfix //Fusionar hotfix con master.

Eliminar una rama:

~$ git branch -d <<nombre_rama>>

~$ git branch -d feature_ABMCliente //Ejemplo

Deshacer Commits

Deshacer Commits

Deshacer un commit:

~$ git reset --soft HEAD^ // Mantiene estado archivos.

~$ git reset --hard HEAD^ // No mantiene estado archivos.

~$ git reset --soft HEAD~N// Deshace los últimos N commits anteriores. Mantiene estado.

Crear Alias de comandos:

```
~$ git config --global alias.undo-commit 'reset --soft HEAD^'
```

Deshacer Commits

E => Estado Archivos - Master => Cabecera HEAD actual.
A, B, C => Commits realizados.

~$ git reset --hard HEAD^ || ~$ git reset --soft HEAD^

(E)

(E)

A-B-C

A-B-C

↑

master

↑

master

Deshacer Commits

Crear un alias para deshacer el último commit:

```
~$ git config --global alias.undo-commit 'reset --soft HEAD^'
```

Links de Utilidad

Links de Utilidad

- http://try.github.com/
- http://nvie.com/posts/a-successful-git-branching-model/
- http://weblog.masukomi.org/2008/07/12/handling-and-avoiding-conflicts-in-git