REPÚBLICA BOLIVARIANA DE VENEZUELA.

MINISTERIO DEL PODER POPULAR PARA LA EDUCACIÓN UNIVERSITARIA.

UNIVERSIDAD VALLE DEL MOMBOY.

FACULTAD DE INGENIERÍA – SEDE ESTOVACUY.

**GIT.**

**INTEGRANTES:**

* Barreto Erikson
* Briceño Rosibel
* González David.
* González Josias.

**Prof.:** Roberto Di Michelle.

­

Carvajal, Julio del 2018.

ÍNDICE.

[INTRODUCCIÓN. 3](#_Toc518411851)

[¿QUÉ ES GIT? 4](#_Toc518411852)

[¿PARA QUÉ SE USA GIT? 4](#_Toc518411853)

[¿QUÉ PROBLEMAS RESUELVE? 5](#_Toc518411854)

[Mantiene tu código a salvo de peligro. 5](#_Toc518411855)

[Elimina la dificultad de trabajar en un mismo proyecto varias personas. 5](#_Toc518411856)

[Permite conocer los cambios concretos que se han realizado. 6](#_Toc518411857)

[USO DE LAS RAMAS EN GIT. 6](#_Toc518411858)

[ÓRDENES O COMANDOS BÁSICOS. 8](#_Toc518411859)

[CONCLUSIONES 10](#_Toc518411860)

# INTRODUCCIÓN.

GIT es una herramienta indispensable para cualquier persona que trata con cualquier tipo de versionado, es un sistema que administra y controla versiones de archivos, su nacimiento comenzó con el creador de Linux “Linux Torvalds” quien después de haber trabajado con una compañía que le ayudaba a administrar las versiones del núcleo de Linux llamada “BitKeeper” se vino abajo, Linux Torvalds y la comunidad de desarrolladores de Linux comenzó a desarrollar su propia herramienta con la experiencia que había tenido con esa empresa, algunos objetivos del sistema eran:

* Velocidad
* Diseño sencillo
* Fuerte apoyo al desarrollo no lineal (miles de ramas paralelas)
* Completamente distribuido
* Capaz de manejar grandes proyectos (como el núcleo de Linux) de manera eficiente (velocidad y tamaño de los datos)

Después de su creación en el 2005, GIT ha evolucionado y se ha convertido en una herramienta más sencilla de utilizar, conservando los objetivos iniciales de desarrollo y proporcionando grandes beneficios.

# ¿QUÉ ES GIT?

Git (pronunciado "guit"​) es un software de control de versiones diseñado por Linus Torvalds, pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando éstas tienen un gran número de archivos de código fuente. Su propósito es llevar registro de los cambios en archivos de computadora y coordinar el trabajo que varias personas realizan sobre archivos compartidos.

Al principio, Git se pensó como un motor de bajo nivel sobre el cual otros pudieran escribir la interfaz de usuario o front end como Cogito o StGIT.​ Sin embargo, Git se ha convertido desde entonces en un sistema de control de versiones con funcionalidad plena.​ Hay algunos proyectos de mucha relevancia que ya usan Git, en particular, el grupo de programación del núcleo Linux.

El mantenimiento del software Git está actualmente (2009) supervisado por Junio Hamano, quien recibe contribuciones al código de alrededor de 280 programadores. En cuanto a derechos de autor Git es un software libre distribuible bajo los términos de la versión 2 de la Licencia Pública General de GNU.

El flujo de trabajo básico en Git es algo así:

* Modificas una serie de archivos en tu directorio de trabajo.
* Preparas los archivos, añadiendolos a tu área de preparación.
* Confirmas los cambios, lo que toma los archivos tal y como están en el área de preparación, y almacena esas instantáneas de manera permanente en tu directorio de Git.

Si una versión concreta de un archivo está en el directorio de Git, se considera confirmada (committed). Si ha sufrido cambios desde que se obtuvo del repositorio, pero ha sido añadida al área de preparación, está preparada (staged). Y si ha sufrido cambios desde que se obtuvo del repositorio, pero no se ha preparado, está modificada (modified).

# ¿PARA QUÉ SE USA GIT?

Con el control de versiones de GIT, podrás:

* Llevar registro de cada cambio (commit) realizados a tus archivos, puedes volver atrás, hacer distintas variaciones de ellos para probar y todo sin cambiarte de carpeta ni archivos duplicados.
* Puedes mantener el orden con los archivos que están en correcciones, desarrollo de nuevas funcionalidades, en producción u otro propósito (ramas); y puedes cambiar a cada uno de ellos en un instante y volver.
* Puedes desplegar rápidamente a un servidor remoto de producción subiendo solamente los cambios y no archivos completos.
* Como es descentralizado varias personas pueden trabajar en un mismo archivo y luego Git se encarga de incorporar los cambios (a veces no, pero son las menos y muestra gráficamente los conflictos).
* Puedes llegar a saber exactamente cuando se produjo un error y qué autor lo produjo.
* Puedes alojar archivos fácilmente en distintos servicios (github, gitlab, bitbucket) o en tu propio servidor remoto. Pudiendo acceder fácilmente, etc.
* Y no sólo sirve para código informático, puede servir de control al escribir libros, wikis, incluso para llevar el control de cambios al retocar imágenes fotográficas (aunque no se recomienda por el peso).

# ¿QUÉ PROBLEMAS RESUELVE?

## Mantiene tu código a salvo de peligro.

* Si nuestra máquina se estropea y deja de funcionar, corrompiendo el disco duro, no hay problema, ya que sólo se perdió la información de nuestro proyecto que teníamos en local, si contamos con servicios como GitHub, Bitbucket u otros similares, tenemos nuestros repositorios almacenados en la nube, y podemos volver a descargar el proyecto y seguir trabajando.
* Permite tener control completo de todo tu código y permite el acceso y la posibilidad de estar conectado con tu equipo, sin importar si se encuentran en la otra punta del planeta.
* Antes de hacer cualquier cambio en el código y subirlo al repositorio Git hace uso de sistemas de árbol SHA1, lo que asegurará que hasta que no se realice la comprobación del cifrado o firma NO se escribirán los cambios.
* Se puede hacer una compartición selectiva ya que tú decides a quien le compartes qué de tus archivos dentro de la red. Puedes crear .gitignore y estableces los nombres de los archivos o carpetas que no desees subir al repositorio, de este modo, sólo quedaran guardados en tu computadora.

## Elimina la dificultad de trabajar en un mismo proyecto varias personas.

* Sin una herramienta como ésta, se hace muy difícil trabajar varias personas en un mismo proyecto, ya que por ejemplo, dos personas no pueden trabajar en un mismo archivo a la vez, y se crean muchos conflictos a la hora de averiguar qué cosas cambio “x” persona para yo no cambiarlo, o cuál fue el último cambio hecho en el código que produjo un error “x”. Git, nos resalta con colores qué partes hemos modificado nosotros y qué otras las ha hecho nuestro compañero de trabajo, para que así podamos decidir qué código implementar.
* Al trabajar en Github, Bitbucket, etc., si tenemos que crear una web con unos estilos concretos y una estructura que ya hemos definido, simplemente tenemos que crear algo parecido a una plantilla de la que todos los integrantes del grupo de trabajo podrán partir, que se guardará en la rama master. De este modo todos pueden partir del mismo punto, pero cada persona se puede concentrar en trabajar en alguna característica en concreto
* Cada Integrante puede trabajar desde cualquier lugar que se encuentre, simplemente conectándose al repositorio remoto donde está almacenado el repositorio.

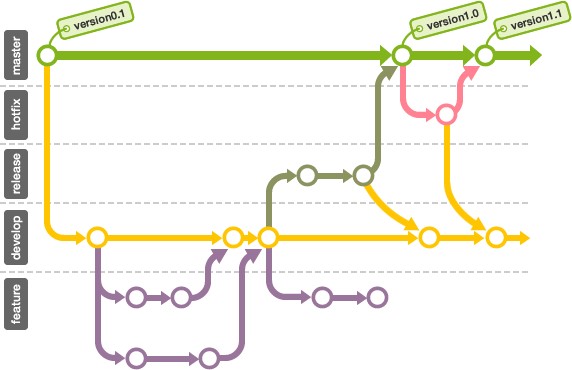
## Permite conocer los cambios concretos que se han realizado.

* ¿Cuántos no han escuchado a estudiantes hablar de cuando hacían su tesis tenían que escribir varios nombres para saber la “versión más actual o la que sería la definitiva”? Ejemplo: Tesis\_final, Tesis\_final\_esta\_sí, Tesis\_ya\_casi\_lista… Con Git esto no ocurriría, porque cada cambio que vamos realizando, lo podemos ir subiendo al repositorio, al hacerlo vamos escribiendo un pequeño mensaje de qué hicimos en él, por lo que vamos teniendo una certeza de cuál es nuestro último código, o en qué cambio concreto tuvimos un error, esto lo verificamos revisando los commits.
* Podemos hacer varias versiones y realizar pruebas en nuestro código, para poder recular si nos hemos equivocado y nuestra aplicación ya no funciona, o para trabajar en funcionalidades nuevas sin necesidad de modificar la versión funcional y así no romper el proyecto.

# USO DE LAS RAMAS EN GIT.

Cada desarrollador o equipo de desarrollo puede hacer uso de Git de la forma que le parezca conveniente. Sin embargo una buena práctica es la siguiente: Se deben utilizar 4 tipos de ramas: Master, Development, Features, y Hotfix.

* Master: Es la rama principal. Contiene el repositorio que se encuentra publicado en producción, por lo que debe estar siempre estable.
* Development: Es una rama sacada de master. Es la rama de integración, todas las nuevas funcionalidades se deben integrar en esta rama. Luego que se realice la integración y se corrijan los errores (en caso de haber alguno), es decir que la rama se encuentre estable, se puede hacer un merge de development sobre la rama master.
* 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 esté desarrollada, se hace un merge de la rama sobre development, donde se integrará con las demás funcionalidades.
* Hotfix: Son bugs que surgen en producción, por lo que se deben arreglar y publicar de forma urgente. Es por ello, que son ramas sacadas de master. Una vez corregido el error, se debe hacer un merge de la rama sobre master. Al final, para que no quede desactualizada, se debe realizar el merge de master sobre development.
* Release o Stage branch: Es aquella rama en la cual los usuarios o el cliente realizan sus test de aceptación y validan aquellas funcionalidades que fueron desarrolladas o la resolución de algún bug.



# ÓRDENES O COMANDOS BÁSICOS.

* git fetch:

Descarga los cambios realizados en el repositorio remoto.

* git merge <nombre\_rama>:

Impacta en la rama en la que te encuentras parado, los cambios realizados en la rama “nombre\_rama”.

* git pull:

Unifica los comandos fetch y merge en un único comando.

* git commit -am "<mensaje>":

Confirma los cambios realizados. El “mensaje” generalmente se usa para asociar al commit una breve descripción de los cambios realizados.

* git push origin <nombre\_rama>:

Sube la rama “nombre\_rama” al servidor remoto.

* git status:

Muestra el estado actual de la rama, como los cambios que hay sin commitear.

* git add <nombre\_archivo>:

Comienza a trackear el archivo “nombre\_archivo”.

* git checkout -b <nombre\_rama\_nueva>:

Crea una rama a partir de la que te encuentres parado con el nombre “nombre\_rama\_nueva”, y luego salta sobre la rama nueva, por lo que quedas parado en esta última.

* git checkout -t origin/<nombre\_rama>:

Si existe una rama remota de nombre “nombre\_rama”, al ejecutar este comando se crea una rama local con el nombre “nombre\_rama” para hacer un seguimiento de la rama remota con el mismo nombre.

* git branch:

Lista todas las ramas locales.

* git branch -a:

Lista todas las ramas locales y remotas.

* git branch -d <nombre\_rama>:

Elimina la rama local con el nombre “nombre\_rama”.

* git push origin <nombre\_rama>:

Commitea los cambios desde el branch local origin al branch “nombre\_rama”.

* git remote prune origin:

Actualiza tu repositorio remoto en caso que algún otro desarrollador haya eliminado alguna rama remota.

* git reset --hard HEAD:

Elimina los cambios realizados que aún no se hayan hecho commit.

* git revert <hash\_commit>:

Revierte el commit realizado, identificado por el “hash\_commit”.

CONCLUSIONES**.**

Un controlador de versiones es un sistema que nos permite guardar un registro de las modificaciones que realizamos sobre un fichero a lo largo del tiempo. Habitualmente se utiliza en entornos de desarrollo de software, pero puede resultar de gran utilidad para cualquier persona que necesite un control robusto sobre la tarea que está realizando. La mayoría de la gente utilizamos algún sistema de control de versiones sin ser conscientes de ello. El ejemplo más claro es cuando editamos cualquier tipo de fichero en servicios de almacenamiento en “la nube” como Google Drive o Dropbox.

Gracias a los sistemas de control de versiones más potentes –como Git–, conseguimos muchas ventajas como el acceso al registro de modificaciones sobre un fichero. Esto nos permite además ver los comentarios asociados a cada modificación y la persona que ha realizado dicha modificación, posibilidad de regresar al estado anterior de un fichero o conjunto de ficheros, creación de ramas para gestionar cambios que finalmente se mezclarán con la rama principal, entre otras. Podemos encontrar en ellos muchas otras utilidades. Tienen mecanismos que nos permiten buscar los cambios que han introducido un fallo en el código, lo que acorta mucho la depuración de los programas. Además, debido a su carácter distribuido resulta poco probable que se produzca cualquier pérdida de código.