**GIT**

**¿Qué es GIT?**

Es un **sistema de** **control de versione**s, es gratis y open source. Es el sistema de control de versiones más utilizado en desarrollo en la actualidad.

**¿Qué es control de versiones?**

La gestión de cambios a documentos, programas de computadora, sitios web y otra colección de información.

Con el rastreamos nuestros cambios de código.

Guardamos una versión inicial de nuestro código en git y después cuando actualizamos el código volvemos a guardarlo en git, una y otra vez.

A medida que nuestro código continúa cambiando podemos visualizar todos los cambios que hicimos a través del tiempo, esto nos ayuda a ver y entender lo que hicimos, así como rastrear errores o volver a una versión anterior del código en caso de ser necesario.

Es un **sistema de control de versiones**, es el mas popular, cualquier desarrollador debería de usarlo.

Es un sistema que controla o administra las distintas versiones de un programa. Un programador lo utiliza para controlar los cambios su proyecto. Tener listado de los cambios y contribuir con otros desarrollados, o solamente en un proyecto personal, pero de manera mas profesional.

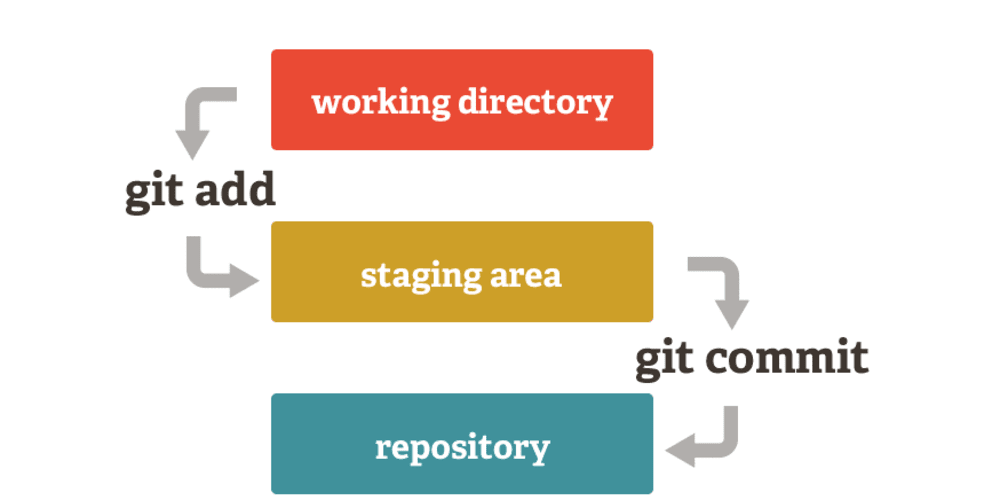
* Control de versiones distribuido.
* Coordina el trabajo entre múltiples desarrolladores.
* Ayuda a conocer quien hizo que cambio y cuando.
* Revertir en cualquier momento.
* Repositorios locales y remotos.

Siempre existe una copia local del repositorio.

Puedes regresar a versiones anteriores (commits). **Se puede navegar entre cualquier punto de la versión del programa.**

Al empezar a trabajar con GIT te encuentras con una gran cantidad de comandos, con una gran cantidad de operaciones que puedes realizar.

Toma snapshots de tus archivos, tu decides en que momento realizar un snapshot al hacer un commit, puedes visualizarlos en cualquier momento, puedes preparar tu código antes de hacer commit.



Git, interactúa directamente con el sistema operativo, Normalmente lo manejas a través de la **consola**, git instala un bash que es como una consola, pero con mejores funcionalidades, trae conceptos de unix.

En Git existen 3 estados:

* **Working directory:**

Donde se esta trabajando con los archivos.

* **Staging area:**

Agregar todos los archivos que vas a ir preparando para el guardado.

* R**epository:**

Donde se encuentra el proyecto(huésped)

**Comandos básicos**

* **git init:** se le dice al sistema operativo que vamos a utilizar en el proyecto git.
* **git add <file>:** Pasar archivos del working directory al staging área.
* **git status:** Ver en que estado están tus archivos: working directory o staging área.
* **git commit:** Pasar archivos del staging área al repositorio. Realizar un snapshot.
* **git push:** Subirlo a un repositorio remoto, cuando se tiene código ya trabajado se sube a un servidor para poder ser accedido por los desarrolladores.
* **git pull:** Traer cambios de otros desarrolladores.
* **git clone:** Hacer una copia desde el servidor central donde esta el código hasta tu computador.

**Instalar GIT.**

-Descargarlo desde la página oficial.

- Instalar bash con unix

**ls:** Listar directorios y archivos, en donde estamos

**pwd:** ver en que directorio estamos, ver en que ruta de nuestro sistema esta posicionado la terminal.

**git config – global user.email “your mail”:** para configurar el email de este usuario.

**git config – global user.name “your name”:** para configurar el nombre de este usuario.

**git log:** Ver todos los commit que hemos creado.

**git checkout – “nombre archivo”:** Revertir los cambios en el archivo que he hecho.

**git diff “nombre archivo”:** Ver las diferencias hechas en los archivos.

**GitHub** – Es el sitio web donde hospedas(almacenas) todos tus repositorios de Git, todos sus repositorios de git están en línea, lo que facilita el trabajo en grupo con otras personas y organiza sus proyectos en una cartera para mostrar a los empleados potenciales.

GitHub Workflow

**write code**

**Commit changes**

**Make a pull request**

Local Git Workflow

**write code**

**Stage changes**

**Commit changes**

**Push changes**

**Make a Pull Request**

Iniciar de nuevo un repositorio:

1. Git init
2. Agregar un nuevo archivo
3. Git status
4. Git add
5. Git commit
6. Hacer un git push origin master <- dice que origin no existe
7. Crear nuevo repositorio
8. Git remote add origin (pegar lint que copie)
9. Git push origin master

Agregar archivo de .gitignore

HACER RAMA NUEVA

HACER COMMIT EN RAMA NUEVA

Regresar a la rama de master.

Visualizar github, crear repositorio, subirlo, clonarlo.

**Git branching – Master Branch (Main or Default branch)**

Se llama branching porque es más como un arbol cuando tienes multiples ramas.

**Master –** Es la main o default rama en el repositorio, si solo utilizas una rama, todo lo que hagas será esa rama.

**Feature –** facción - rama característica

Master se queda como esta, si trabajas en la rama feature todos los cambios que hagas se quedaran solamente en esa rama. Cada rama indivual no saben que commit o cambios se hicieron en otras diferentes ramas.

Esto es muy útil porque podemos hacer realizar nuevas caracteríasticas a nuestro código que podría break el codigo o que no hemos terminado aún, cuando el codigo este correcto se hace un merge a la rama principal.

**Hotfix Branch:** Se hace un cambio en un commit, pruebas que este funcionando correctamente y haces merge en tu rama principal.

**Proyecto utilizando ramas**

**git checkout -b “feature-descriptivo”**

**git branch** – ver en que rama estoy

**git checkout master** – checkout me cambia de ramas

**git checkout “feature-descriptivo”**

**\*realizar un cambio en algún archivo**

**git status** – saber que fue lo que modifique

**git add “archivo que modifique”**

**git commit -m “titulo”**

**git checkout master** – no se ven los cambios de mi otra rama

**git diff “feature-descriptivo”** – ver diferencias entre ramas

**git checkout “feature-descriptivo”**

**git push origin -u “feature-descriptivo”** - “-u” es para setterar el hilo

\*\*Explica lo del pull request, hacer merge mediante pull request

Cambiar a master para ver que los cambios persisten

git merge “feature-descriptivo”

**git pull origin master** – Traer los nuevos cambios hechos con el merge

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**git branch -b “feature-descriptivo”** – Se supone que ya no debes de usar ramas que ya hiciste merge.

**git branch** – Ya no existe la rama.

Git checkout -b “nueva-rama”

\*modificar un arhivo

**git status** y **git diff**

**git commit -am “mensaje de commit”**

**-am :** solo funciona para archivos modificados

**git checkout master**

\*en master modificar lo mismo pero con otro texto.

**git checkout “nueva-rama”** – deberia de marcar error

**git status**

**git commit -am “nuevo mensaje”**

**git checkout “nueva-rama”** – si debe de dejar

**git diff master** – muestra los cambios

**git merge master** – debe de marcar conflictos

\***Ir al editor de código – para correguir los errores**

**git status** – se debe de hacer otro commit

**git commit -am “resolver conflictos”**

**UNDOING WITH GIT**

**\*Añadir un cambio en otro archive**

**git add .**

**git status**

**git reset o git reset “nombrearchivo”** – uso git reset solo

git status – el archivo no esta en stage

**git add “nombre archive”**

**git commit -m “se añadio nuevo cambio”**

**git status – dice que ya no hay nada a que hacer commit**

**git reset HEAD~1 –** head apunta al ultimo commit – 1 es que avance 1 commit

mac: option + ñ windows: **Alt gr + 4 + espacio -** ALT + 126

**git status**

**git diff** – muestra los cambios

**git log** – para ver los commit que tenemos

\*Copiar un hash de commit

**git reset “hash”**

**\*Aun se pueden ver los cambios pero ya no estan en stage en Git**

**git log**

**git reset - -hard “Hash”**

**\*Visualizar que ya no tengo los cambios**

**FORK** -crea una copia complete del repositorio

Full access, full control, hacer actualizaciones

**Pull requests** permiten informar a otros sobre los cambios que ha enviado a una rama en un repositorio en GitHub. Una vez que se abre una solicitud de extracción, puede discutir y revisar los cambios potenciales con los colaboradores y agregar confirmaciones de seguimiento antes de que sus cambios se fusionen en la rama base.

**SSH KEY**

Si aún no tiene una clave SSH, debe generar una nueva clave SSH. Si no está seguro de si ya tiene una clave SSH, verifique las claves existentes.

Si no desea volver a ingresar su frase de contraseña cada vez que usa su clave SSH, puede agregar su clave al agente SSH, **que administra sus claves SSH y recuerda su frase de contraseña.**

**Ssh-keygen -t rsa -b 4096 -C “**[**email**](mailto:yeratzyen@gmail.com)**”**

**ls | grep yeratzye**

**Carpeta:**

$ cd ~/.ssh

$ cat ~/.ssh/id\_rsa.pub

https://docs.github.com/en/github/authenticating-to-github/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent

1. Abrir terminal
2. Pegue el texto a continuación, sustituyéndolo por su dirección de correo electrónico de GitHub.

$ **ssh-keygen -t rsa -b 4096 -C "your\_email@example.com"**

Esto crea una nueva clave ssh, utilizando el correo electrónico proporcionado como etiqueta.

**> Generando par de claves rsa pública / privada.**

1. Cuando se le solicite "Ingresar un archivo en el que guardar la clave", presione Entrar. Esto acepta la ubicación de archivo predeterminada.

> Ingrese un archivo en el cual guardar la clave (/Users/you/.ssh/id\_rsa): [Presione enter]

1. Cuando se le indique, escriba una frase de contraseña segura. Para obtener más información, consulte "Trabajar con frases de contraseña de clave SSH".

**> Ingrese la frase de contraseña (vacío si no hay frase de contraseña): [Escriba una frase de contraseña]**

**> Ingrese la misma contraseña nuevamente: [Escriba nuevamente la contraseña]**