GIT HUB

¿QUÉ ES GIT?

- Git es una herramienta que realiza una función de control de versiones distribuido, , lo que significa que un clon local del proyecto de un repositorio de control de versiones completo, es decir, estos repositorios locales permiten trabajar sin conexión y de forma remota con facilidad.

  Los desarrolladores y demás usuarios revisan su trabajo localmente y a continuación sincronizar la copia del repositorio con la copia del servidor, este proceso es distinto al que usan las herramientas de control de versiones centralizado, donde los clientes sincronizar el código con un servidor antes de crear nuevas versiones, generando inconvenientes si perdemos la conexión a internet

- El control de versiones se define como los diverso cambios que se posee un elemento oproducto, en el campo del software se entiende como cuando se sincroniza y actualiza un codigo, o cuando se añaden, editan o eliminan instrucciones que afectan el programa o el pryecto en el que se este trabajando. Este control sobre las versiones de los proyectos permite llevar un historial de lo que se ha estado trabajando.

- Git se creó pensando en la eficiencia y mantenimiento de versiones de aplicaciones que poseen un gran número de archivos o cuando muchas personas requieren trabajar en ellas a la vez.

  Algunas de las herramientas de Git son:

  - GitHub
  - GitLab
  - Atlassian Bitbucket
  - Git Kraken

CARACTERÍSTICAS DE GIT:

- Rendimiento
- Seguridad
- Flexibilidad

ESTRUCTURA Y DEFINICIONES:

FLUJO DE TRABAJO: Nuestro repositorio local esta compuesta por tres “arboles” administrados por Git.

- Working Directory: Contiene los archivos en la maquina local

- Staging Area:Que actua como una zona intermedia
- Repository:Que se relaciona con el ultimo commit realizado

RAMAS:Son utilizadas para desarrollar funcionalidades aisladas. Al crear un repositorio, la rama “master” es la rama que se crea por defecto, se crean nuevas ramas durante el desarrollo y se fusionan con la rama principal al terminar.

COMANDOS BÁSICOS: Para trabajar con Git, existen una serie de comandos que nos permite interactuar con el repositorio y sus ramas, cuando manejamos un etnorno grafico de verisonamiento, el entorno se encarga de realizar esta ejecucion. A continuacion una lista de comandos y sus respectivas funcionaes.

LISTA DE COMANDOS BÁSICOS.pdf - Google Drive

README Y GITIGNORE

README:El archivo readme, es un archivo plano el cual lo vamos a utilizar para comentarle a otras personas por qué nuestro proyecto es útil, qué pueden hacer y cómo lo pueden usar,

funciona como una forma de exponer tu objetivo y la instrucciones o uso del proyecto.

Los archivos README habitualmente incluyen información sobre:

- Qué hace el proyecto
- Por qué el proyecto es útil
- Cómo pueden comenzar los usuarios con el proyecto
- Dónde pueden recibir ayuda los usuarios con tu proyecto
- Quién mantiene y contribuye con el proyecto

GITIGNOTE: Este recurso lo utilizamos para especificar los archivos que intencionalmente Git debería ignorar. Los archivos ya seguidos por Git no se ven afectados. Cada línea que posee nuestro gitignore especifica un patrón. Al decidir si ignoramos una ruta, nuestro gitignore verifica los patrones de múltiples fuentes.

.gitignore

```
# Compiled source #
###################
*.com
*.class
*.dll
*.exe
*.o
*.so

# Packages #
############
# it's better to unpack these files and commit the raw source
# git has its own built in compression methods
*.7z
*.dmg
*.gz
*.iso
*.jar
*.rar
*.tar
*.zip

```

SOURCETREE:

Es una herramienta que nos permite trabajar con los sistemas de versionado de forma gráfica desde la cual podemos:

- Crear y clonar repositorios de cualquier sitio, tanto Git como Mercurial. Además de integrarse perfectamente con Bitbucket o Github.
- Commit, push, pull y merge de nuestros archivos
- Detectar y resolver conflictos
- Consultar el historial de cambios de nuestros repositorios.

CONFIGURACION INICIAL:

$ git config –global user.name Sara (Comando para asociar un nombre)
$ git config –global user.email saracastanom19@gmail.com (Comando para asociar un correo)

CLONANDO REPOSITORIO REMOTO:

Utilizamos bitbucket y creamos una cuenta.
Le damos a nuevo espacio de trabajo, creamos un repository en Bitbucker
Ahora, queremos clonar el repositorio con la carpeta que creamos

git clone https://Amarilla1909@bitbucket.org/gitamarilla/pruebagit.git comando y ruta https:// para clonar el repositorio

Ultima linea muestra que ya se conecto

```
Sara Castaño@LAPTOP-DMI487UF MINGW64 ~/Desktop/curso-git
$ git clone https://Amarilla1909@bitbucket.org/gitamarilla/pruebagit.git
Cloning into 'pruebagit'...
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (4/4), 965 bytes | 87.00 KiB/s, done.
```

Debemos de fijarnos en la ruta de la carpeta, cuando clonamos el repositorio en la carpeta curso-git se evidencia que ahora existe otra carpeta llamada pruebagit, accedemos a ella y le damos click derecho Git Bash Here

git checkout -b prueba lo que hace es que crea una nueva rama y nos posiciona en ella, es decir, ya no estamos en la rama (master) sino en la rama (prueba)

```
Sara Castaño@LAPTOP-DMI487UF MINGW64 ~/Desktop/curso-git/pruebagit (master)
$ git checkout -b prueba
Switched to a new branch 'prueba'

Sara Castaño@LAPTOP-DMI487UF MINGW64 ~/Desktop/curso-git/pruebagit (prueba)
$
```

SUBIR CAMBIO PROYECTO LOCAL A REMOTO:

Con git status podemos observar que archivos nos hace falta por añadir a nuestro repositorio

```
Sara Castaño@LAPTOP-DMI487UF MINGW64 ~/Desktop/curso-git/pruebagit (prueba)
$ git status
On branch prueba
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        prueba.txt

nothing added to commit but untracked files present (use "git add" to track)
```

git add . el punto agrega todos los archivos que no han sido cargados

```
Sara Castaño@LAPTOP-DMI487UF MINGW64 ~/Desktop/curso-git/pruebagit (prueba)
$ git add .

Sara Castaño@LAPTOP-DMI487UF MINGW64 ~/Desktop/curso-git/pruebagit (prueba)
$ git commit -m "se agrega archivo prueba.txt"
[prueba 9dbfc30] se agrega archivo prueba.txt
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 prueba.txt
```

Se agrega un commit con el mensaje correspondiente al proceso que se esta haciendo, todo esto para entender mejor cual fue el cambio que se hizo o que actualizacion se realizo

```
Sara Castaño@LAPTOP-DMI487UF MINGW64 ~/Desktop/curso-git/pruebagit (prueba)
$ git commit -m "se agrega archivo prueba.txt"
[prueba 9dbfc30] se agrega archivo prueba.txt
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 prueba.txt
```

git push es el comando para que se envie la informacion al repositorio, pero esto nos lanza un error, debido a que esta rama no esta creada dentro del repositorio

```
Sara Castaño@LAPTOP-DMI487UF MINGW64 ~/Desktop/curso-git/pruebagit (prueba)
$ git push
fatal: The current branch prueba has no upstream branch.
To push the current branch and set the remote as upstream, use

    git push --set-upstream origin prueba

To have this happen automatically for branches without a tracking
upstream, see 'push.autoSetupRemote' in 'git help config'.
```

Copiamos y pegamos el siguiente comando:

git push --set-upstream origin prueba

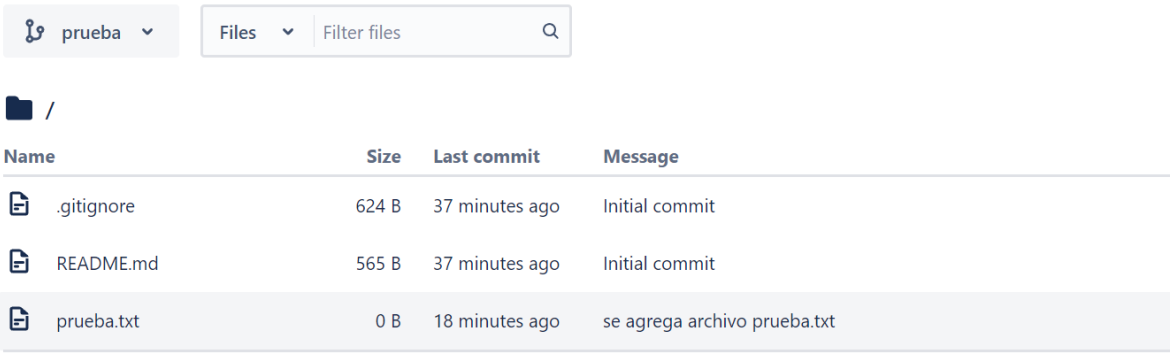
Una vez hecho esto, nuestra rama ya habra sido creada en el repositorio

```
Sara Castaño@LAPTOP-DMI487UF MINGW64 ~/Desktop/curso-git/pruebagit (prueba)
$ git push --set-upstream origin prueba
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 16 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 324 bytes | 324.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create pull request for prueba:
remote:   https://bitbucket.org/gitamarilla/pruebagit/pull-requests/new?source=p
rueba&t=1
remote:
To https://bitbucket.org/gitamarilla/pruebagit.git
 * [new branch]      prueba -> prueba
branch 'prueba' set up to track 'origin/prueba'.
```

y podemos ejecutar el comando git push nuevamente sin problema alguno, quedando asi todos los archivos actualizados en nuestro repositorio

```
Sara Castaño@LAPTOP-DMI487UF MINGW64 ~/Desktop/curso-git/pruebagit (prueba)
$ git push
Everything up-to-date
```

Comprobamos en nuestro repositorio remoto y observamos que estando en la rama prueba aprece todo el proceso que hicimos



COLABORADORES:Un colaborador es aquel que presta sus servicios y que son retribuidos por otra persona, ya sea un particular, una empresa o una institución.En git se puede añadir uno o mas colaboradores, quienes tendran acceso a los archivos y podran editar los archivos que se encuentren en el , siempre y cuando lo clonen en un repositorio local.

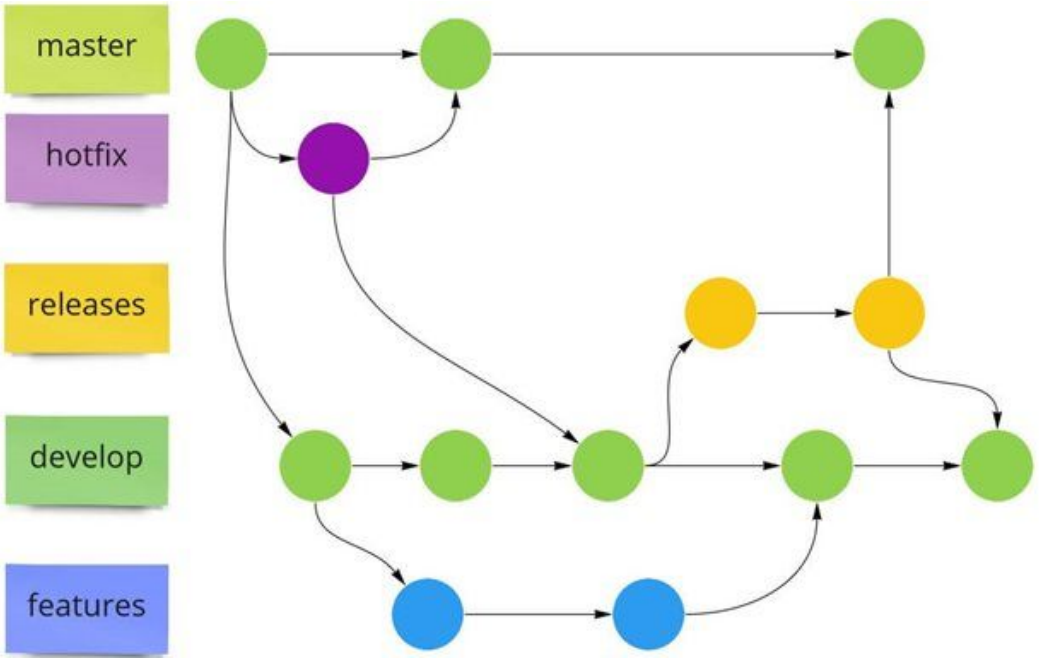
GITFLOW: Es un flujo de trabajo basado en Git que brinda un mayor control y organización en el proceso de integración continua.
En comparación con el desarrollo basado en troncos, Gitflow tiene diversas ramas de más duración y mayores confirmaciones.

¿POR QUÉ SE DEBE IMPLEMENTAR?

- Aumentamos la velocidad de entrega del código que desarrollamos como equipo
- Podemos disminuir los errores humanos a la hora de mezclar nuestras ramas
- Elimina la dependencia de funcionalidades al momento de entregar código para ser puesto en producción

¿CUÁNDO DEBEMOS IMPLEMENTARLO?
Existen unos escenarios que se nos presentan a lo largo de un desarrollo, por lo que es recomendable tener estas políticas y buenas prácticas para un correcto manejo de nuestro proyecto.

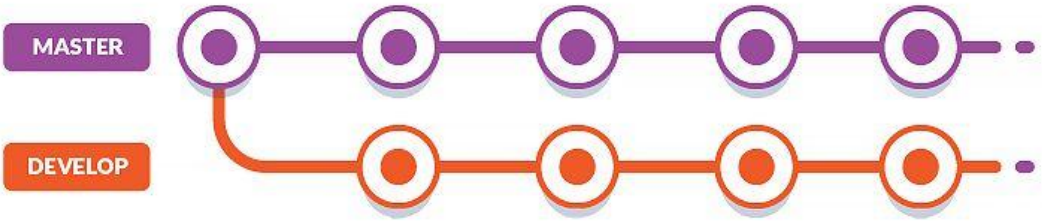
FLUJOS DE TRABAJO
El secreto para cumplir un buen flujo de trabajo, se encuentra en el correcto manejo y administración de nuestras ramas, las cuales cumplen un objetivo fundamental en el versionado.



FLUJOS DE TRABAJO

El trabajo se organiza en dos ramas principales:

- Master: Esta rama se considera como la productiva, cualquier commit que tengamos sobre esta debe estar preparada para subirse a producción
- Develop: Esta rama es en la que tenemos el código que conforma la siguiente versión planificada de nuestro proyecto



FLUJOS DE TRABAJO

Adicionalmente, se propone tres ramas auxiliares:

- feature: Estas ramas se utilizan para desarrollar nuevas características de la aplicación que, una vez terminadas, se incorporan a la rama develop.
    - Debemos crearlas a partir de la rama develop

    - Siempre debemos fusionarlas con la rama develop
- release: Estas ramas se utilizan para preparar el siguiente código en producción. En estas ramas realizamos los últimos ajustes y corregimos los últimos bugs antes de pasar a la rama master.
    - Las creamos a partir de la rama develop
    - Las fusionamos a la rama master y develop
- hotfix: Estas ramas se emplean para verificar errores y bugs en el código en producción.
    - Las creamos a partir de la rama master
    - Las fusionamos a la rama master y develop

PULL REQUEST:Es la acción que se realiza para validar el código que se va a mergear de una rama a otra. En este proceso de validación pueden presentarse:

- Builds (validaciones automáticas)
- asignación de código a tareas
- validaciones manuales por parte del equipo
- despliegues