JAVA

Versionado

¿Qué es un sistema de control de versionado?

Un sistema de control de versionado (SCV o CVS en inglés) registra el historial de cambios a medida que las personas y equipos colaboran.

A medida que el proyecto evoluciona, los equipos pueden ejecutar pruebas, corregir errores y aportar código nuevo con la confianza de que cualquier versión se puede recuperar en cualquier momento. Los desarrolladores pueden revisar el historial del proyecto para averiguar:

- ¿Qué cambios fueron realizados?
- ¿Quién realizó los cambios?
- ¿Cuándo los cambios fueron realizados?
- ¿Por qué los cambios fueron necesarios?

¿Qué es un sistema de control de versionado distribuido?

Git es un ejemplo de un **sistema de control de versiones distribuido** (SCVD o en inglés DVCS) comúnmente utilizado para el desarrollo de software comercial y de código abierto.

Los DVCS permiten:

- el acceso completo a cada archivo, rama e iteración de un proyecto,
- y permiten que cada usuario acceda a un historial completo y autónomo de todos los cambios.

A diferencia de los sistemas de control de versiones centralizados que alguna vez fueron populares, **los DVCS como Git no necesitan una conexión constante a un repositorio central**. Los desarrolladores pueden trabajar en cualquier lugar y colaborar de forma asincrónica desde cualquier zona horaria.

ACLARACIÓN:
Github (u otros como Bitbucket o Gitlab) no es Git. Github es una página web que aloja y permite gestionar proyectos que usan GIT

INFORMATORIO

Relacionemos Conceptos y Comandos

ESCENARIO PRÁCTICO

Deseo almacenar varios ejercicios prácticos de Java en un solo directorio, donde pueda actualizarlos, eliminarlos y agregar nuevos ejercicios a medida que voy avanzando en el curso de Java del Informatorio.

Importante:

1. Suponemos que tenemos una cuenta creada en Github y Git instalado. Sino es así, podes ver en la presentación de “Introducción - (Herramientas de Desarrollo)” los pasos para crear la cuenta de Github y cómo instalar Git.
2. La mayoría de los comandos de Git los realizaremos por consola. Es posible usar plugins/extensiones que provee Visual Studio Code que abrevian en pocos clicks las operaciones que realizaremos. Pero no se recomienda, porque al fallar o tener conflictos se debe recurrir a los comandos por consola.

REPOSITORIO

Un repositorio es un proyecto GIT, abarca la colección completa de archivos y carpetas asociados con un proyecto, junto con el historial de revisión de cada archivo.

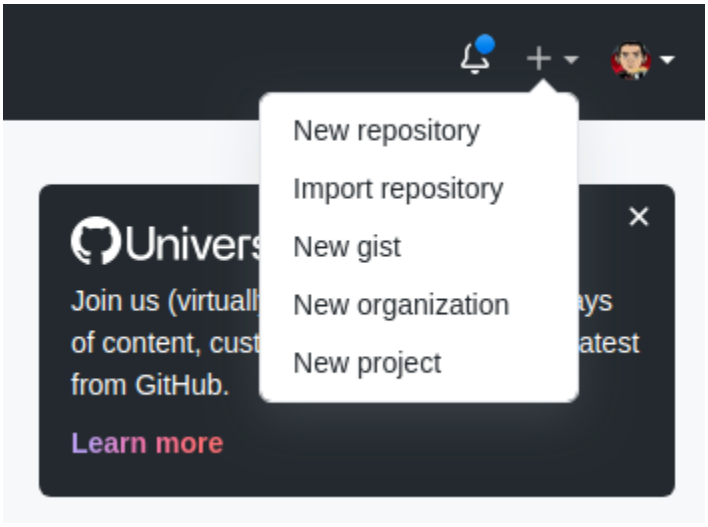
Repositorio Local: Es la copia del repo/proyecto en nuestra PC/Laptop

Repositorio Remoto: Es el lugar donde está alojado nuestro proyecto (ej: Github) y donde sincronizamos nuestros cambios periódicamente (sea desarrollador independiente o parte de un equipo).

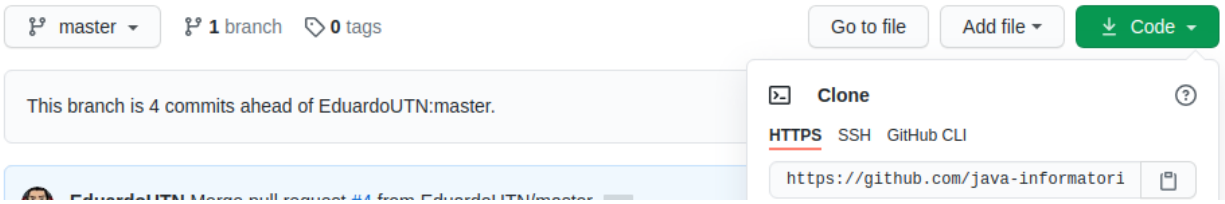
REPOSITORIO - PRÁCTICA

Para poder arrancar la práctica de nuestro escenario debemos:

1. Abrir la consola:
    a. **Windows** (cmd - Solo simbolo del Sistema o como sugerimos Git Bash)
    b. **Linux (Ubuntu)**: `Ctrl + Alt + T` o ejecutar el acceso directo a la aplicación “Terminal”.
2. Elegir nuestro espacio de trabajo. Ejemplos (no obligatorios):
    a. **Windows**:
        i. `C:\Proyectos\Informatorio-Java2020`
    b. **Linux**:
        i. `~/Proyectos/Informatorio-Java2020( ~ es igual a decir /home/mi-nombre-usuario`. Que varía según cada uno)
3. Desde la consola podemos posicionarnos en ese directorio (Como? - buscamos en Google como navegar entre directorios por la consola. TIP: comandos `pwd`, `cd`, `mkdir`, `ls` y mas).
4. Luego iniciamos sesión en Github, y creamos un nuevo repo:
    a. ¿Dónde? Hay varios atajos o links para crear un nuevo repositorio, pero el más común es hacer click en el símbolo + (a la izquierda de tu avatar) y luego elegir New Repository.



b. Luego nos solicitará un nombre para nuestro repo. Ejemplo: java-info-2020
c. Continuamos los pasos normalmente (si nos sugiere agregar un README.md, acepto).
d. Luego, nos llevará a la página de inicio de nuestro Nuevo Repo. Seleccionamos el botón verde de “Code” (y copiamos la URL)



e. Volvemos a la consola, asegurarnos de que estamos en el directorio que creamos para nuestros repos. Y ejecutamos (reemplacen el valor url-del-repo por el que copiaron antes):
    i. git clone **url-del-repo**
    ii. cd java-info-2020 (reemplacen por el nombre del repo que crearon en el paso 4.b)

iii. Excelente, ya tenemos nuestro repo.

REALIZAR CAMBIOS - AGREGAR, MODIFICAR Y ELIMINAR

Dentro de nuestro repo local (nuestra PC/Laptop) podemos realizar cualquiera de los cambios siguientes:

- Agregar carpetas (directorios) y/o archivos
- Renombrar carpetas (directorios) y/o archivos
- Mover carpetas (directorios) y/o archivos
- Eliminar carpetas (directorios) y/o archivos

ESPACIO DE TRABAJO

También llamado Área de trabajo o Workspace. Contiene todos los cambios que mencionamos antes (agregar, renombrar, mover y eliminar)

Git a partir de que clonamos comienza a detectar todos los cambios en la carpeta de proyecto/repo local (también llamada raíz del proyecto/repo).

¿Cómo podríamos saber cuales son los archivos y que tipo de cambios realizamos?

Para eso ejecutamos: **`git status`**

Que nos detalla todos los archivos y que operación se hizo (new, modified, deleted y otros. TIP: Usen Google translate si no reconocen la palabra para entender el significado)

Buenísimo. Ya está en Github?
NO. Git solo sabe que hiciste cambios en tu PC/Laptop, todavía no sabe qué cosas subir (repo remoto)

Como le decimos a GIT qué cambios quiero agregar para subir a nuestra cuenta en Github?

Primero debemos tener en cuenta que si realizamos muchos cambios (por ej: 20 archivos). Tengo la opción de:

- Agregar todos los cambios (debo estar muy seguro), los 20 archivos
  - Ejecutamos: **`git add .`**

    Observación: El símbolo "." (punto) representa todo lo que está en este directorio y todo lo que se encuentra en los directorios hijos (anidados)

- O solo algunos de los archivos (ej: 2 o 3, el resto todavía quiero seguir trabajando un poco más)
  - Al ejecutar el comando **`git status`** nos retorno los cambios que registró git, copiamos los archivos (si están dentro de carpetas deben copiar toda la ruta que les devuelve el comando)
  - Ejecutamos:
    - **`git add archivo1`** -- para solo un archivo
    - **`git add archivo1 ejemplos1/archivo2`** -- para multiples archivos (vean que archivo 2 esta dentro del directorio ejemplos1)
    - **`git add ejemplos1/`** --- para incluir un directorio, en este caso si dentro hay 5 archivos los agregara también.

Buenísimo. Ya está en Github?
NO. Git solo agregó estos archivos al Staging Area (Área de Preparación o Listos para Commit), sabe que estamos preparando cambios para enviar a nuestro repositorio remoto.

ÁREA DE ENSAYO (STAGING AREA)

Luego de ejecutar el comando git add. Los archivos y/o directorios pasan del Workspace al Staging Area. Resulta útil pensar en él como un intermediario entre el directorio de trabajo y el historial del proyecto.

¿Cómo le confirmó a GIT qué cambios que están en el STAGING AREA (o Listo para Commit) están listos para subir a nuestra cuenta en Github?

- Primero confirmamos que están en el staging area con: **`git status`**
- Debajo del Mensaje **Changes to be committed:** estarán numerados los archivos/directorios en el Staging Area.
- Ejecutamos:
    - `git commit -m "Primeros ejercicios de Informatorio Java 2020"`
    - Observación: `-m` es una opción del comando commit de git, al colocarlo quiere decir que el string siguiente será el mensaje de commit.

OOPS NO FUNCIONÓ:
Si es la primera vez que estás trabajando con git o estás en otra PC/Laptop, debes configurar el Nombre y email de la persona que está realizando el cambio.
Hagámoslo, ejecutemos (recordá reemplazar lo que está dentro de comillas por tus valores):

- **`git config --global user.name "FIRST_NAME LAST_NAME"`**
- **`git config --global user.email "MY_NAME@example.com"`**
- Es por única vez
- Verifiquemos que quedo seteado, ejecuta: `git config --global -l`
- Busca las variables `user.name` y `user.email` y comproba que tenga los valores que agregaste
- Ahora vuelve a ejecutar el comando ***git commit*** ...

Buenísimo. Ya está en Github?
NO. Los cambios están registrados (tienen fecha y autor) pero GIT todavía no recibió orden de enviar a Github (repo remoto).

HISTORIAL (HISTORY):

Luego de ejecutar el comando git commit ..., los archivos y/o directorios que se encontraban en el Staging Area pasar a registrarse al Historial de cambios.

¿Ahora si podemos enviar a Github?

SI. con los cambios registrados (commit) podemos indicar a GIT que actualice nuestro repositorio remoto con los cambios que tengo en mi local.

1. Verifiquemos si commit se encuentra registrado (por si acaso)
    a. Ejecutar: **`git log`**
    b. Observación: Nos retorna un listado de cambios, donde veremos las fechas, autores y si los mensajes son relevantes hasta nos podrían decir el motivo del commit.
2. Ahora, ejecutamos:
    a. `git push -u origin master`
    b. Observación: Como recién estamos arrancando y solo nosotros realizaremos cambios en nuestro repo. Usaremos el branch master (que dejaremos estos conceptos intermedios para más adelante cuando realicemos trabajos grupales)

CONCLUSIÓN

Git es un ejemplo de un sistema de control de versiones distribuido (DVCS) y Github es una página web que aloja y permite gestionar proyectos que usan GIT.

Siendo un desarrollador único o parte de un equipo, sincronizo mis cambios de mi repositorio local (PC, Laptop o VM) con los del repositorio remoto (Github, Bitbucket u otro). Y viceversa.

El flujo de trabajo para añadir cambios desde mi local al remoto debe atravesar varias etapas:

Workspace (Área de Trabajo) - Contiene los cambios que realizó para testear, prototipar, etc.

Staging Area (Area de Preparacion/Ensayo) - En esta etapa se agregan los cambios que fueron parte del comando git add

History (Historial de cambios) - Una vez confirmados los cambios del staging área (por el comando git commit) estos son agregados al historial y se registra el fecha y hora, mensaje de commit y autor.

Por último, para sincronizar nuestro historial de nuestro repo local con el remoto, debemos ejecutar un git push.

Finalmente podremos refrescar nuestro repo en Github con los cambios realizados. Puede llegar a suceder que el comando git push falle (conflictos, que lo veremos más adelante). Que es muy común al trabajar en equipos.