Nombre: Richard Andrés Montero Ogando

Matricula: 2022-2146

Asignatura: Programación lll

1. ¿Qué es Git?

Git es un sistema de control de versiones distribuido, lo que significa que un clon local del proyecto es un repositorio de control de versiones completo. Estos repositorios locales plenamente funcionales permiten trabajar sin conexión o de forma remota con facilidad. Los desarrolladores confirman su trabajo localmente y, a continuación, sincronizan la copia del repositorio con la del servidor. Este paradigma es distinto del control de versiones centralizado, donde los clientes deben sincronizar el código con un servidor antes de crear nuevas versiones.

La **flexibilidad** y **popularidad** de Git lo convierten en una excelente opción para cualquier equipo. Muchos desarrolladores y graduados universitarios ya saben cómo usar Git. La comunidad de usuarios de Git ha creado recursos para entrenar a los desarrolladores y la popularidad de Git facilita recibir ayuda cuando se necesita. Casi todos los entornos de desarrollo tienen compatibilidad con Git y las herramientas de línea de comandos de Git implementadas en todos los sistemas operativos principales.

2. ¿Cuál es el propósito del comando git init en Git?

El comando **git init** crea un nuevo repositorio de Git. Puede utilizarse para convertir un proyecto existente y sin versión en un repositorio de Git, o para inicializar un nuevo repositorio vacío. La mayoría de los demás comandos de Git no se encuentran disponibles fuera de un repositorio inicializado, por lo que este suele ser el primer comando que se ejecuta en un proyecto nuevo.

Al ejecutar **git init**, se crea un subdirectorio de **.git** en el directorio de trabajo actual, que contiene todos los metadatos de Git necesarios para el nuevo repositorio. Estos metadatos incluyen subdirectorios de objetos, referencias y archivos de plantilla. También se genera un archivo **HEAD** que apunta a la confirmación actualmente extraída.

Aparte del directorio de **.git**, en el directorio raíz del proyecto, se conserva un proyecto existente sin modificar (a diferencia de SVN, Git no requiere un subdirectorio de **.git** en cada subdirectorio).

De manera predeterminada, **git init** inicializará la configuración de Git en la ruta del subdirectorio de **.git**. Puedes cambiar y personalizar dicha ruta si quieres que se encuentre en otro sitio. Asimismo, puedes establecer la variable de entorno **$GIT_DIR** en una ruta personalizada para que **git init** inicialice ahí los archivos de configuración de Git. También puedes utilizar el argumento **--separate-git-dir** para conseguir el mismo resultado. Lo que se suele hacer con un subdirectorio de **.git** independiente es mantener los archivos ocultos de la configuración del sistema (**.bashrc**, **.vimrc**, etc.) en el directorio principal, mientras que la carpeta de **.git** se conserva en otro lugar.

3. ¿Qué representa una rama en Git y cómo se utiliza?

Las ramas en git son una división del estado del código, esto permite crear nuevos caminos a favor de la evolución del código y como crear nueva rama en git. Normalmente la creación de ramas en otros sistemas de control de versiones puede tomar mucho tiempo y ocupar demasiado espacio en el almacenamiento. Por el contrario, las ramas en Git son parte diaria del desarrollo, son una guía instantánea para los cambios realizados.

Ahora que hemos respondido qué son las ramas, hablaremos de su funcionamiento. Las ramas en Git figuran una línea independiente de desarrollo. Estas sirven como una abstracción en los procesos de edición, preparación y confirmación. Las puedes entender como una forma de solicitar un nuevo directorio de trabajo, un nuevo ambiente de trabajo o un nuevo historial en el proyecto. Todas esas nuevas confirmaciones son registradas en el historial de la rama principal, creando una bifurcación en el historial del proyecto.

El comando git Branch permite crear, enumerar y eliminar ramas, así como editar su nombre. eso sí, no permite cambiar entre ramas o volver a unir un historial bifurcado. Es por este motivo que git branch está estrechamente integrado con los comandos git checkout y git merge.

Nuevas ramas

El comando **git checkout** se usa habitualmente junto con **git branch**. El comando **git branch** permite crear una rama nueva. Si quieres empezar a trabajar en una nueva función, puedes crear una rama nueva a partir de la rama **main** con **git branch new_branch**. Una vez creada, puedes usar **git checkout new_branch** para cambiar a esa rama. Además, el comando **git checkout** acepta el argumento **-b**, que actúa como un práctico método que creará la nueva rama y cambiará a ella al instante. Puedes trabajar en varias funciones en un solo repositorio alternando de una a otra con **git checkout**.

```
git checkout -b <new-branch>
```

Cambio de rama

El comando **git checkout** te permite desplazarte entre las ramas creadas por **git branch**. Al extraer una rama, se actualizan los archivos en el directorio de trabajo para reflejar la versión almacenada en esa rama y se indica a Git que registre todas las confirmaciones nuevas en dicha rama. Puedes contemplar todo esto como una forma de seleccionar la línea de desarrollo en la que trabajas.

```
$> git branch
main
another_branch
feature_inprogress_branch
```

```
$> git checkout feature_inprogress_branch
```

4. ¿Cómo puedo determinar en qué rama estoy actualmente en Git?

Para determinar en qué rama estás actualmente en Git, puedes utilizar el siguiente comando en la terminal dentro del directorio de tu repositorio:

```
$> git branch
```

5. ¿Quién es la persona responsable de la creación de Git y cuándo fue desarrollado?

Git fue concebido por **Linus Torvalds** como una respuesta a las limitaciones del sistema de control de versiones existente en ese momento, conocido como **BitKeeper**, que era utilizado para el desarrollo del **kernel** de **Linux**. En 2005, debido a problemas relacionados con la licencia de BitKeeper, la comunidad del kernel de Linux perdió el acceso gratuito a este sistema de control de versiones, lo que llevó a Torvalds a desarrollar su propio sistema.

Linus Torvalds comenzó el desarrollo de Git en abril de 2005, y en tan solo unos meses, en julio de 2005, Git ya se utilizaba ampliamente dentro de la comunidad del kernel de Linux. El desarrollo inicial de Git estuvo motivado por la necesidad de un sistema de control de versiones distribuido que fuera rápido, eficiente y capaz de manejar el tamaño y la complejidad del desarrollo del kernel de Linux, que involucra a miles de desarrolladores distribuidos en todo el mundo.

Git se diseñó con ciertos principios en mente, como la velocidad, la distribución, la capacidad de manejar grandes proyectos y la capacidad de manejar ramas y fusiones de manera eficiente. Estos principios permitieron que Git se convirtiera rápidamente en una herramienta fundamental para el desarrollo de software en equipos distribuidos y colaborativos.

El desarrollo de Git continuó después de su lanzamiento inicial, con contribuciones significativas de la comunidad de código abierto. Hoy en día, Git es utilizado no solo para el desarrollo del kernel de Linux, sino también en una amplia variedad de proyectos de software en todo el mundo. Su flexibilidad, velocidad y robustez lo han convertido en uno de los sistemas de control de versiones más populares y ampliamente utilizados en la industria del desarrollo de software.

6. ¿Cuáles son algunos de los comandos esenciales de Git y para qué se utilizan?

git init creará un nuevo repositorio local GIT. El siguiente comando de Git creará un repositorio en el directorio actual:

```
git init
```

Como alternativa, puedes crear un repositorio dentro de un nuevo directorio especificando el nombre del proyecto:

```
git init [nombre del proyecto]
```

git clone se usa para copiar un repositorio. Si el repositorio está en un servidor remoto, usa:

```
git clone nombredeusuario@host:/path/to/repository
```

A la inversa, ejecuta el siguiente comando básico para copiar un repositorio local:

```
git clone /path/to/repository
```

git add se usa para agregar archivos al área de preparación. Por ejemplo, el siguiente comando de Git básico indexará el archivo temp.txt:

```
git add <temp.txt>
```

git commit creará una instantánea de los cambios y la guardará en el directorio git.

```
git commit –m “El mensaje que acompaña al commit va aquí”
```

git config puede ser usado para establecer una configuración específica de usuario, como el email, nombre de usuario y tipo de formato, etc. Por ejemplo, el siguiente comando se usa para establecer un email:

```
git config --global user.email tuemail@ejemplo.com
```

La opción -global le dice a GIT que vas a usar ese correo electrónico para todos los repositorios locales. Si quieres utilizar diferentes correos electrónicos para diferentes repositorios, usa el siguiente comando:

```
git config --local user.email tuemail@ejemplo.com
```

git status muestra la lista de los archivos que se han cambiado junto con los archivos que están por ser preparados o confirmados.

```
git status
```

git push se usa para enviar confirmaciones locales a la rama maestra del repositorio remoto. Aquí está la estructura básica del código:

```
git push  origin <master>
```

git checkout crea ramas y te ayuda a navegar entre ellas. Por ejemplo, el siguiente comando crea una nueva y automáticamente se cambia a ella:

```
command git checkout -b <branch-name>
```

Para cambiar de una rama a otra, sólo usa:

```
git checkout <branch-name>
```

git remote te permite ver todos los repositorios remotos. El siguiente comando listará todas las conexiones junto con sus URLs:

```
git remote -v
```

Para conectar el repositorio local a un servidor remoto, usa este comando:

```
git remote add origin <host-or-remoteURL>
```

Por otro lado, el siguiente comando borrará una conexión a un repositorio remoto especificado:

```
git remote <nombre-del-repositorio>
```

git branch se usa para listar, crear o borrar ramas. Por ejemplo, si quieres listar todas las ramas presentes en el repositorio, el comando debería verse así:

```
git branch
```

Si quieres borrar una rama, usa:

```
git branch -d <branch-name>
```

git pull fusiona todos los cambios que se han hecho en el repositorio remoto con el directorio de trabajo local.

```
git pull
```

git merge se usa para fusionar una rama con otra rama activa:

```
git merge <branch-name>
```

git diff se usa para hacer una lista de conflictos. Para poder ver conflictos con respecto al archivo base, usa:

```
git diff --base <file-name>
```

El siguiente comando se usa para ver los conflictos que hay entre ramas antes de fusionarlas:

```
git diff <source-branch> <target-branch>
```

Para ver una lista de todos los conflictos presentes usa:

```
git diff
```

git tag marca commits específicos. Los desarrolladores lo usan para marcar puntos de lanzamiento como v1.0 y v2.0.

```
git tag 1.1.0 <instert-commitID-here>
```

git log se usa para ver el historial del repositorio listando ciertos detalles de la confirmación. Al ejecutar el comando se obtiene una salida como ésta:

```
commit 15f4b6c44b3c8344caasdac9e4be13246e21sadw

Author: Alex Hunter <alexh@gmail.com>

Date:   Mon Oct 1 12:56:29 2016 -0600
```

git reset sirve para resetear el index y el directorio de trabajo al último estado de confirmación.

```
git reset - -hard HEAD
```

git rm se puede usar para remover archivos del index y del directorio de trabajo.

```
git rm filename.txt
```

git stash guardará momentáneamente los cambios que no están listos para ser confirmados. De esta manera, pudes volver al proyecto más tarde.

```
git stash
```

git show se usa para mostrar información sobre cualquier objeto git.

```
git show
```

git fetch le permite al usuario buscar todos los objetos de un repositorio remoto que actualmente no se encuentran en el directorio de trabajo local.

```
git fetch origin
```

git ls-tree te permite ver un objeto de árbol junto con el nombre y modo de cada ítem, y el valor blob de SHA-1. Si quieres ver el HEAD, usa:

```
git ls-tree HEAD
```

git cat-file se usa para ver la información de tipo y tamaño de un objeto del repositorio. Usa la opción -p junto con el valor SHA-1 del objeto para ver la información de un objeto específico, por ejemplo:

```
git cat-file –p d670460b4b4aece5915caf5c68d12f560a9fe3e4
```

git grep le permite al usuario buscar frases y palabras específicas en los árboles de confirmación, el directorio de trabajo y en el área de preparación. Para buscar por www.hostinger.com en todos los archivos, usa:

```
git grep www.hostinger.com
```

gitk muestra la interfaz gráfica para un repositorio local. Simplemente ejecuta:

```
gitk
```

git instaweb te permite explorar tu repositorio local en la interfaz GitWeb. Por ejemplo:

```
git instaweb –http=webrick
```

git gc limpiará archivos innecesarios y optimizará el repositorio local.

```
git gc
```

git archive le permite al usuario crear archivos zip o tar que contengan los constituyentes de un solo árbol de repositorio. Por ejemplo:

```
git archive - -format=tar master
```

git prune elimina los objetos que no tengan ningún apuntador entrante.

```
git prune
```

git fsck realiza una comprobación de integridad del sistema de archivos git e identifica cualquier objeto corrupto

```
git fsck
```

git rebase se usa para aplicar ciertos cambios de una rama en otra. Por ejemplo:

```
git rebase master
```

7. ¿Puedes mencionar algunos de los repositorios de Git más reconocidos y utilizados en la actualidad?

- Linux Kernel
- Git
- Jquery
- React
- Bootstrap
- Docker