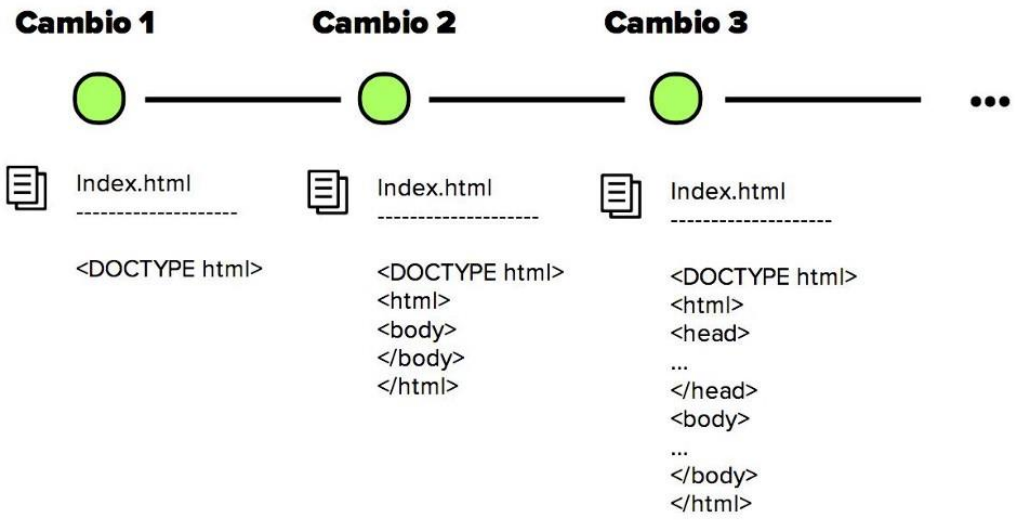
Proyectos de Integración con Git

INTRODUCCIÓN

Los sistemas de control de versiones son una categoría de herramientas de software que ayudan a un equipo de software a gestionar los cambios en el código fuente a lo largo del tiempo. El software de control de versiones realiza un seguimiento de todas las modificaciones en el código en un tipo especial de base de datos. Si se comete un error, los desarrolladores pueden ir atrás en el tiempo y comparar las versiones anteriores del código para ayudar a resolver el error al tiempo que se minimizan las interrupciones para todos los miembros del equipo.

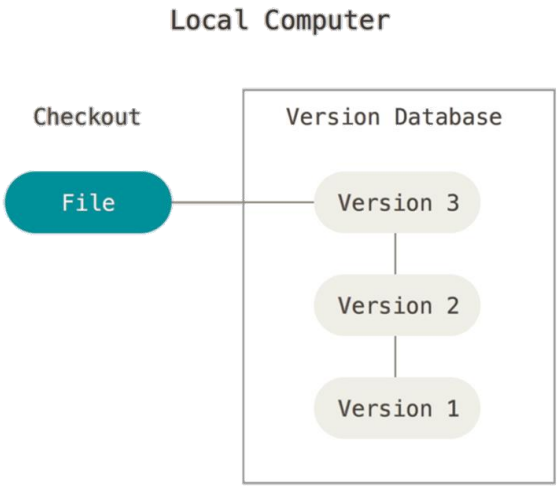


Los desarrolladores de software que trabajan en equipos están escribiendo continuamente nuevo código fuente y cambiando el que ya existe. El código de un proyecto, una aplicación o un componente de software normalmente se organiza en una estructura de carpetas o "árbol de archivos". Un desarrollador del equipo podría estar trabajando en una nueva función mientras otro desarrollador soluciona un error no relacionado cambiando código. Cada desarrollador podría hacer sus cambios en varias partes del árbol de archivos.

Los sistemas de control de versiones han ido evolucionando a lo largo del tiempo y podemos clasificarlos en tres tipos: Sistemas de Control de Versiones Locales, Centralizados y Distribuidos.

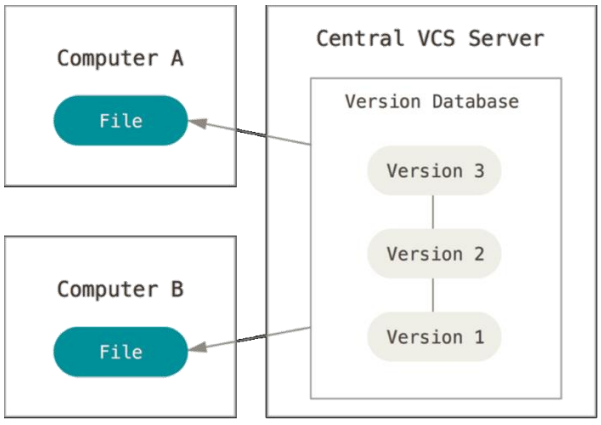
1. **Sistemas de Control de Versiones Locales**

   Los sistemas de control de versiones locales en vez de mantener las versiones como archivos independientes, los almacenaban en una base de datos. Cuando era necesario revisar una versión anterior del proyecto se usaba el sistema de control de versiones en vez de acceder directamente al archivo, de esta manera en cualquier momento solo se tenía una copia del proyecto, eliminando la posibilidad de confundir o eliminar versiones.



2. Sistema de Control de Versiones Centralizados

   Para facilitar la colaboración de múltiples desarrolladores en un solo proyecto los sistemas de control de versiones evolucionaron: en vez de almacenar los cambios y versiones en el disco duro de los desarrolladores, estos se almacenaban en un servidor.

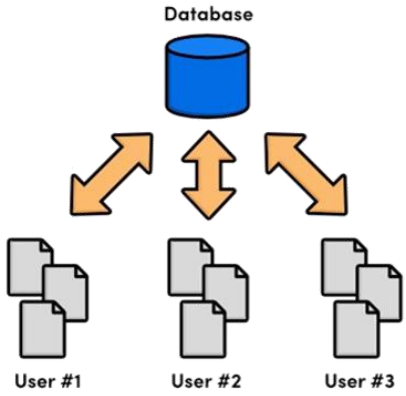


Sin embargo, aunque el avance frente a los sistemas de control de versiones locales fue enorme, los sistemas centralizados trajeron consigo nuevos retos:

- ¿Cómo trabajaban múltiples usuarios en un mismo archivo al mismo tiempo?

  Si dos personas editaban el mismo archivo y se presentaba un conflicto alguien debía solucionar este problema de manera manual y el desarrollo no podía continuar hasta que todos los conflictos fueran resueltos y puestos a disposición del resto del equipo.
- ¿Qué pasa si falla el servidor centralizado?

  Si ese servidor se cae durante una hora, entonces durante esa hora nadie podrá colaborar o guardar cambios en archivos en los que hayan estado trabajando. Si el disco duro en el que se encuentra la base de datos central se corrompe, y no se han realizado copias de seguridad adecuadamente, se perderá toda la información del proyecto, con excepción de las copias instantáneas que las personas tengan en sus máquinas locales.

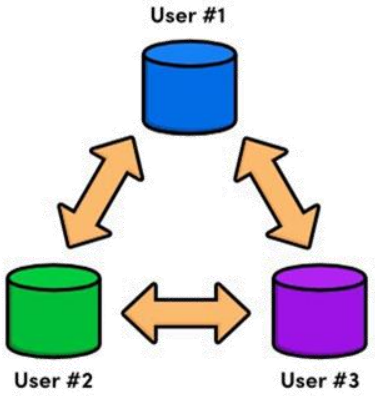


Estos sistemas, como CVS, Subversion y Perforce, tienen un único servidor que contiene todos los archivos versionados y varios clientes que descargan los archivos desde ese lugar central. Este ha sido el estándar para el control de versiones por muchos años.

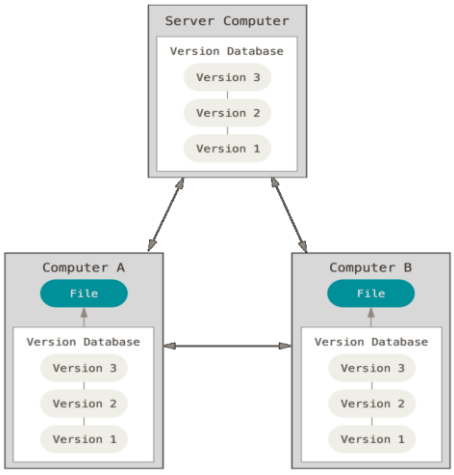
3. Sistemas de Control de Versiones Distribuidos.

La siguiente generación de sistemas de control de versiones se alejó de la idea de un solo repositorio centralizado y optó por darle a cada desarrollador una copia local de todo el proyecto, de esta manera se construyó una red distribuida de repositorios, en la que cada desarrollador podía trabajar de manera aislada, pero teniendo un mecanismo de resolución de conflictos mucho más elegante que en su versión anterior.

Al no existir un repositorio central, cada desarrollador puede trabajar a su propio ritmo, almacenar los cambios a nivel local y mezclar los conflictos que se presenten solo cuando se requiera. Cómo cada usuario tiene una copia completa del proyecto el riesgo por una caída del servidor, un repositorio dañado o cualquier otro tipo de perdida de datos es mucho menor que en cualquiera de sus predecesores.



En un Sistema de Control de Versiones Distribuidos (Git, Mercurial, Bazaar o Darcs), los clientes no solo descargan la última copia instantánea de los archivos, sino que se replica completamente el repositorio. De esta manera, si un servidor deja de funcionar y estos sistemas estaban colaborando a través de él, cualquiera de los repositorios disponibles en los clientes puede ser copiado al servidor con el fin de restaurarlo. Cada clon es realmente una copia completa de todos los datos.



Algunos de estos sistemas se encargan de manejar numerosos repositorios remotos o en cloud con los cuales pueden trabajar, de tal forma que puedes colaborar simultáneamente con diferentes grupos de personas en distintas maneras dentro del mismo proyecto. Esto permite establecer varios flujos de trabajo que no son posibles en sistemas centralizados (Integración Continua).

Actividad 01

+ Responda de manera breve cada una de las preguntas propuestas:

1. ¿Qué es la Integración continua (CI)?
2. ¿Qué es la Entrega continua (CD)?
3. ¿Por qué el control de versiones es importante en los proyectos de CI/CD?
4. ¿Qué es DevOps?
5. ¿Porque seria importante implementar DevOps en proyectos de IoT?

+ Link de interés

Además de las fuentes que usted puede conseguir, puede hacer lectura de la informacion que está contenida en estos links:

https://aws.amazon.com/es/devops/continuous-integration/

https://codigofacilito.com/articulos/integracion-continua

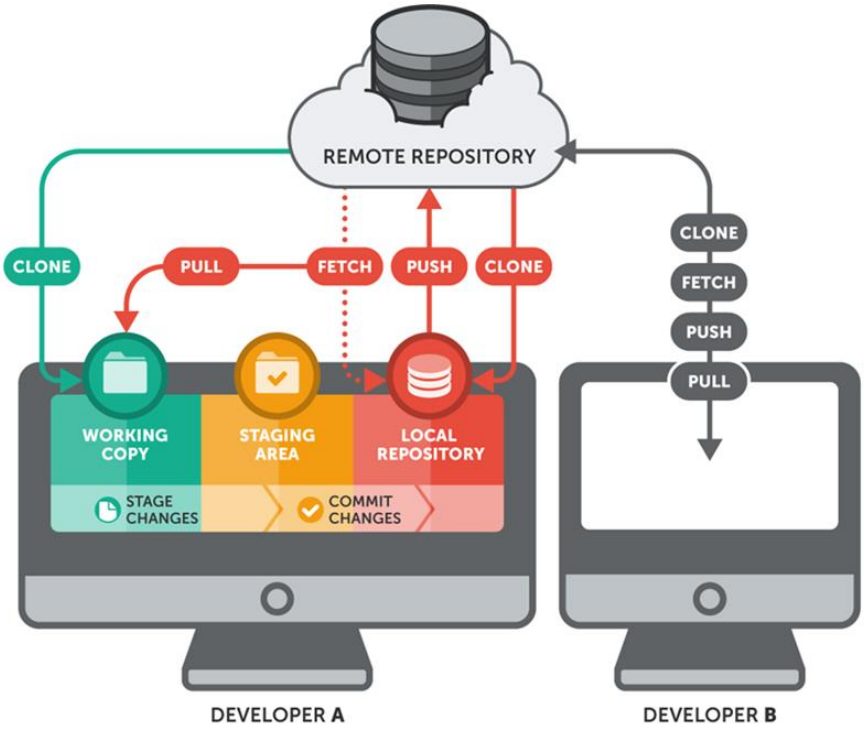
+ Nota

Documentar las respuestas y subir al foro como: A01-<<NombreApellido>.pdf

¿Qué es GIT?

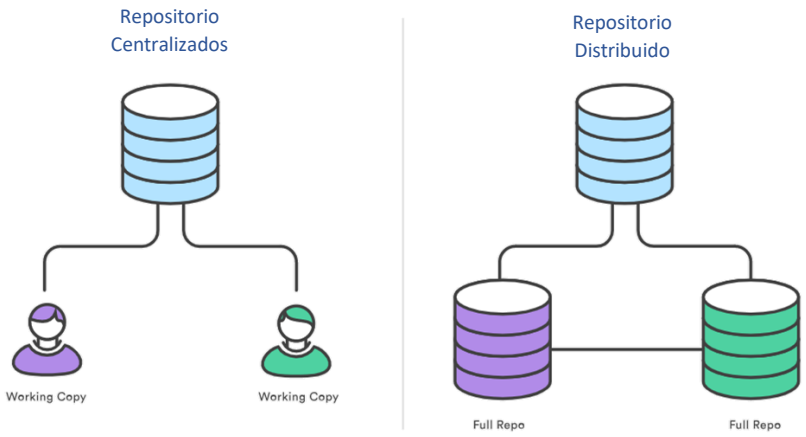
Actualmente **Git es el sistema de control de versiones** más utilizado a nivel mundial. Git es un proyecto de código abierto maduro y con un mantenimiento activo que desarrolló originalmente Linus Torvalds, el famoso creador del kernel del sistema operativo Linux, en 2005. Un asombroso número de proyectos de software dependen de Git para el control de versiones, incluidos proyectos comerciales y de código abierto.

Git, que presenta una **arquitectura distribuida**, es un ejemplo de DVCS (sistema de control de versiones distribuido, por sus siglas en inglés). En lugar de tener un único espacio para todo el historial de versiones del software, como sucede de manera habitual en los sistemas de control de versiones más antiguos, en Git, la copia de trabajo del código de cada desarrollador es también un repositorio que puede albergar el historial completo de todos los cambios.



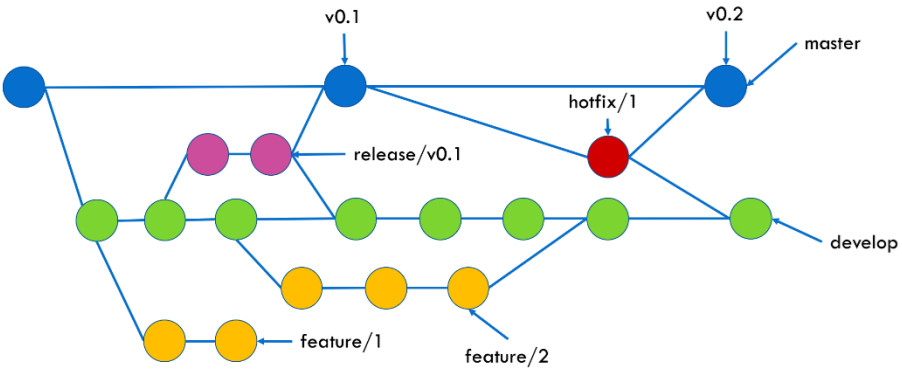
DESARROLLO DISTRIBUIDO

En los sistemas de control de versiones centralizados, cada desarrollador obtiene una copia de trabajo que apunta a un único repositorio central. Sin embargo, Git es un sistema de control de versiones distribuido. En lugar de una copia de trabajo, cada desarrollador obtiene su propio repositorio local, completo con un historial completo de confirmaciones.



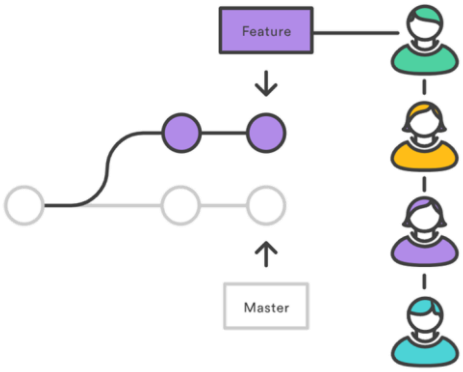
Tener un historial local completo hace que Git sea rápido, ya que significa que no necesita una conexión de red para crear confirmaciones, inspeccionar versiones anteriores de un archivo o realizar diferencias entre confirmaciones.

El desarrollo distribuido también facilita la ampliación de un equipo de proyectos. Si alguien rompe la rama de producción en los sistemas de versiones centralizados, otros desarrolladores no pueden verificar sus cambios hasta que se solucione. Con Git, este tipo de bloqueo no existe. Todos los integrantes del equipo pueden continuar con sus tareas haciendo uso de sus propios repositorios locales.



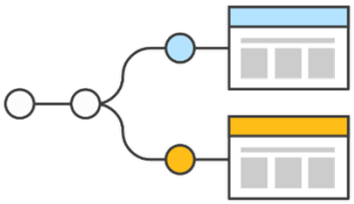
Solicitudes de extracción

Muchas herramientas de administración de código fuente, mejoran la funcionalidad principal de Git con solicitudes de extracción. Una solicitud de extracción es una forma de pedirle a otro desarrollador que combine una de sus ramas en su repositorio. Esto no solo hace que sea más fácil para los líderes del proyecto realizar un seguimiento de los cambios, sino que también permite a los desarrolladores iniciar discusiones sobre su trabajo antes de integrarlo con el resto del código base.



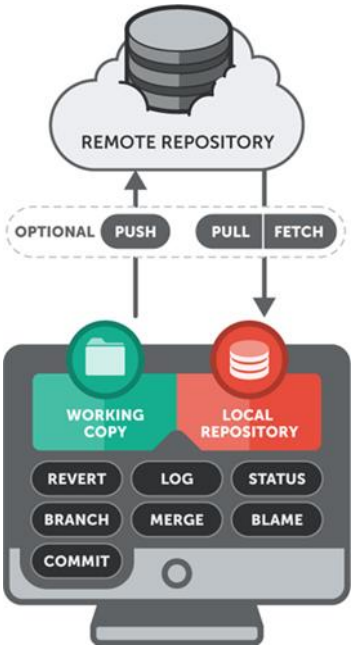
Ciclo de liberación más rápido

Git funciona muy bien con entornos de integración y entrega continuas. Los servicios de Git le permiten ejecutar scripts cuando ocurren ciertos eventos dentro de un repositorio, lo que le permite automatizar la implementación de contenido en su producto. Incluso puede crear o implementar código desde ramas específicas a diferentes servidores.



GIT: Conceptos generales

Algo que como desarrolladores deben conocer muy bien y saber manejar es el uso de repositorios de control de código fuente, garantizando que los proyectos realizados están debidamente versionados, teniendo una trazabilidad de los cambios y del estado de la aplicación en los distintos ambientes. Hoy en día una gran parte de los repositorios de control de código fuente está usado Git, y cada vez son más los que están migrando a dicha plataforma.



Componentes del repositorio Git

1. **Repositorio local**

   Este repositorio es el que está en nuestra máquina local, y es sobre el cual haremos la mayor cantidad de operaciones. Además de que es el que siempre deberá existir de forma obligatoria, ya que se puede tener un repositorio Git solamente en una PC sin uno remoto (por lo cual nunca se podrá compartir el código asociado).

2. **Repositorio remoto**

   Este repositorio es el que está en el servidor, permitiendo sincronizar los cambios de los distintos repositorios locales asociados al mismo.

> *Este esquema implica que todos los cambios y modificaciones que hagamos en el repositorio local no tendrán impacto en el servidor remoto hasta que ejecutemos los comandos de sincronización.*

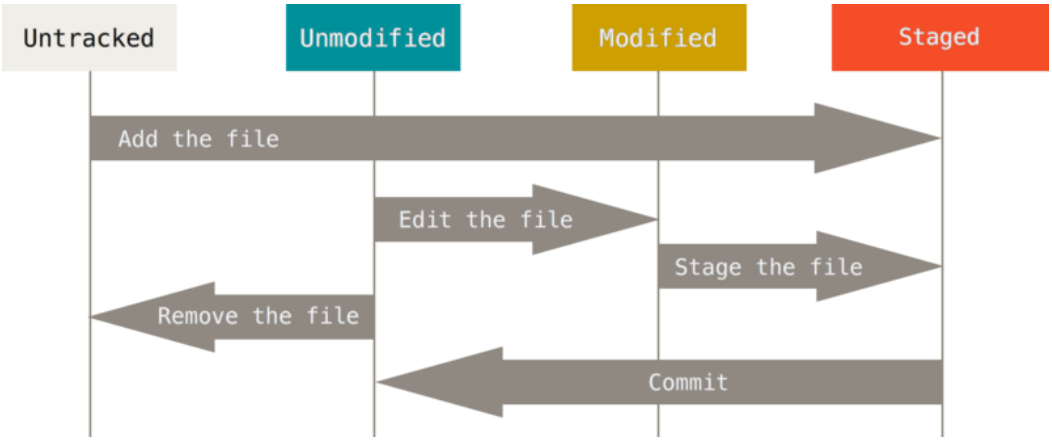
3. **Clonación de repositorios**

   Al instalar Git se debe inicializar un repositorio, para lo cual tendremos dos alternativas:

   - Clonar un repositorio remoto ya creado: Si integramos Git a repositorio que ya se venía usando, o bien la creación inicial se hace de los repositorios que tenemos en sitios como GitHub o BitBucket.
   - Crear un repositorio local: Se crea directamente el repositorio localmente (mediante el comando git init) sobre la carpeta donde se va a trabajar el proyecto.

4. **Estados de los archivos en Git**

   - **Untracked**: Son los archivos nuevos sobre los que Git no está haciendo ningún seguimiento (Rastrear). Esto puede ser tanto porque es un archivo que agregamos o bien porque son archivos excluidos de nuestro repositorio (binarios, dependencias, etc.)
   - **Unmodified**: En este estado estarán todos los archivos cuyo estado no ha cambiado con respecto a lo que está confirmado en el repositorio local.
   - **Modified**: Se da por la modificación del contenido de archivos rastreados por Git
   - **Staged**: En este estado podremos colocar los archivos locales que estén listos para confirmar. Lo interesante de este estado es que una vez que colocamos un archivo en el mismo, lo podremos seguir modificando posteriormente, pero dichos cambios no estarán en **staged** sino en **modified**. Esto implica que la copia en el estado **staged** no es necesariamente el estado de dicho archivo en disco, sino el contenido del mismo que nosotros llevamos a este estado



Estas acciones se realizan sobre una de las unidades claves en Git: las ramas o branches.

5. **Rama**

   Es un puntero a un determinado commit, indicando cuál es el estado de esta. En Git cuando cambiamos de rama, se actualiza el estado de la carpeta donde tenemos nuestro repositorio local. Esto implicará también que en Git no podamos tener dos ramas distintas al mismo tiempo, sin embargo, se pueden crear ramas para organizar el trabajo del proyecto, ya sean para las distintas partes de este y los distintos ambientes de nuestra aplicación.

   Es sobre las ramas donde se realizan los distintos commits, es decir proteger nuestros cambios locales. dicha operación se realizará con los cambios que tengan el estado staged.

Actividad 02

+ Responda de manera breve cada una de las preguntas propuestas:

1. ¿Qué es Git?
2. ¿Qué es Git Hub?
3. ¿Cuál es la diferencia entre Git y Git Hub?

+ Link de interés

Además de las fuentes que usted puede conseguir, puede hacer lectura de la informacion que está contenida en estos links:

https://www.youtube.com/watch?v=YlBy34ECh2s

https://kinsta.com/es/base-de-conocimiento/git-vs-github/

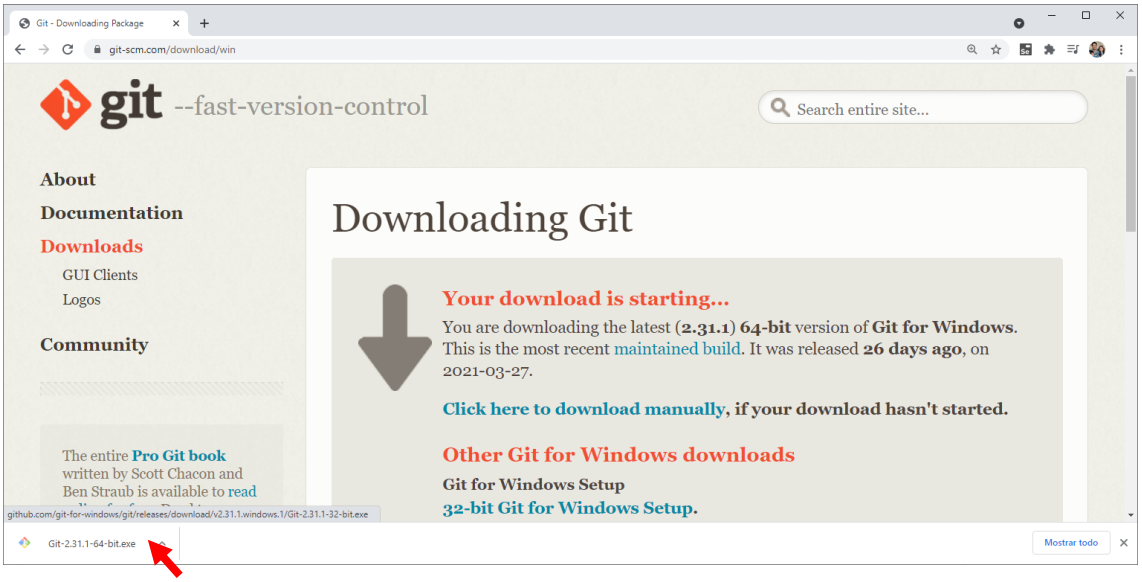
+ Nota

Documentar las respuestas y subir al foro como: A02-<<NombreApellido>.pdf

INSTALAR GIT

1. Ingresar al link
   https://git-scm.com/downloads

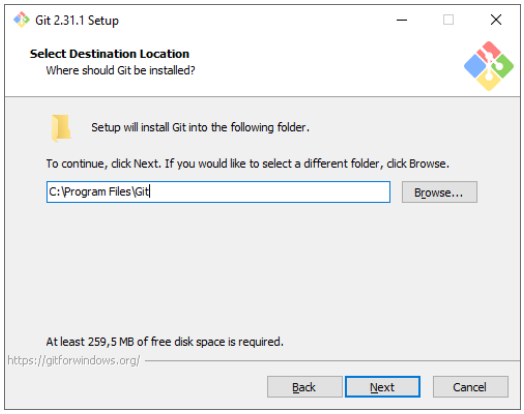
2. Dar clic al botón Download y esperar la descarga



3. Ejecutar el instalador y dar clic en Install

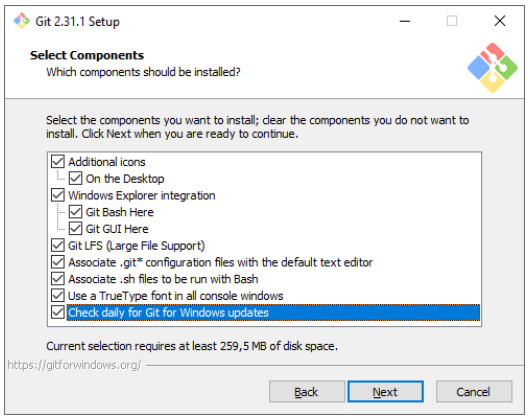
4. Configurar la ruta de instalación

   Aparecerá una ventana donde se debe indicar la ruta donde se desea guardar los archivos de configuración de Git. Seleccione la ruta de su preferencia y de clic en Siguiente.

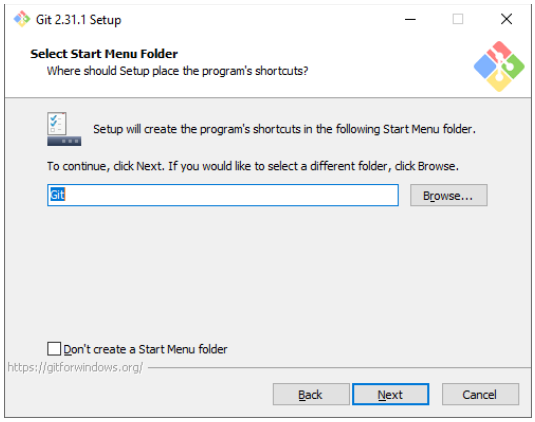


5. Configurando instalación de Git

   En la siguiente pantalla nos muestra algunas configuraciones que podemos cambiar. Seleccione todas las opciones y clic en Siguiente.

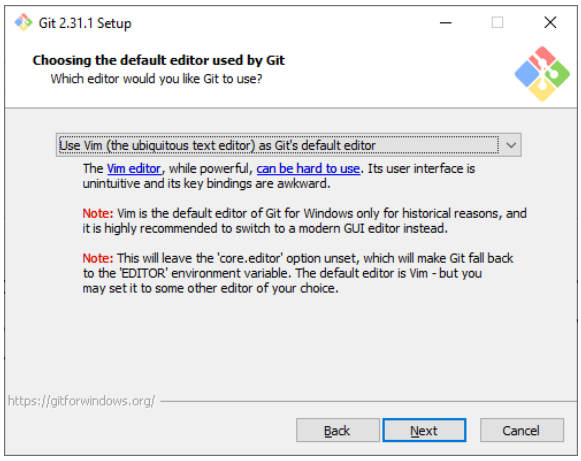


   En la siguiente ventana se muestra el nombre de la carpeta de instalación. No modifique y de clic en Siguiente.

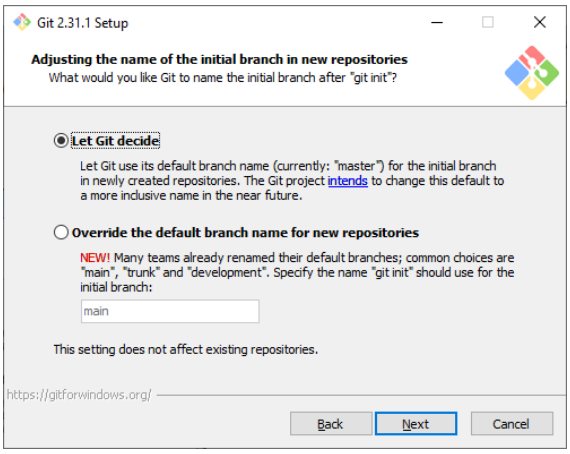


6. En esta ventana nos indica que por default utilizara el editor vim para trabajar en consola, al hacer alguna modificación a un archivo. No modificar.

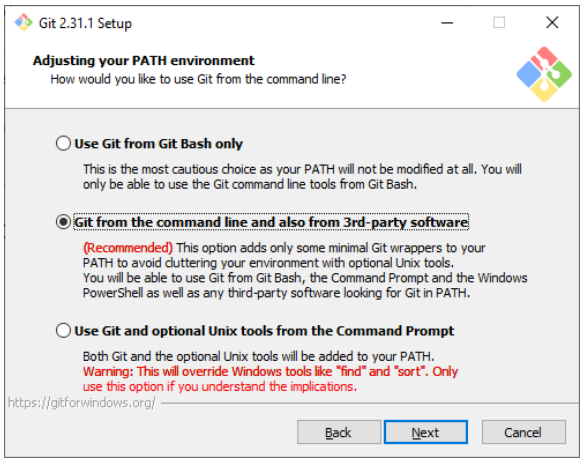
   Dar clic en Siguiente.



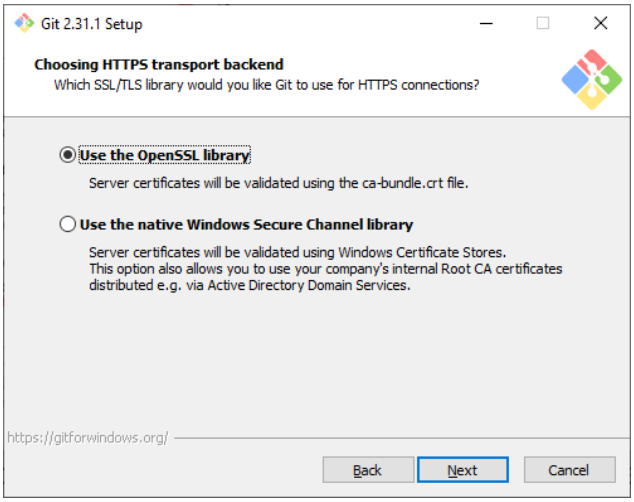
7. En esta ventana, tiene seleccionada por default usar Git como línea de comandos y poder usar git bash para lanzar comandos del propio Git. No modificar. Dar clic en Siguiente.



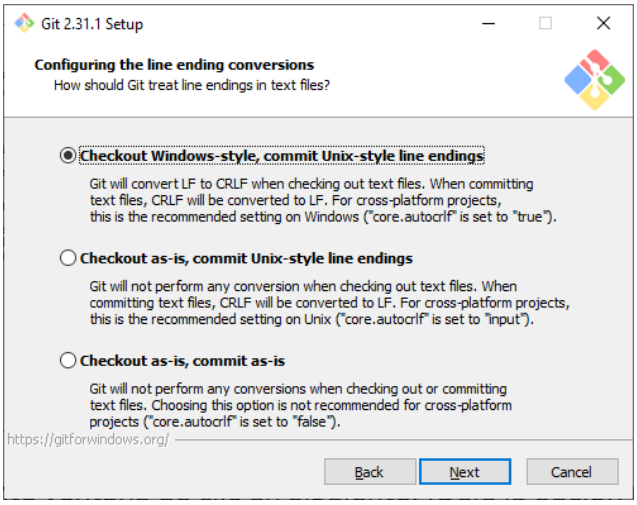
8. En esta ventana selecciones la opción recomendada y dar clic en Siguiente



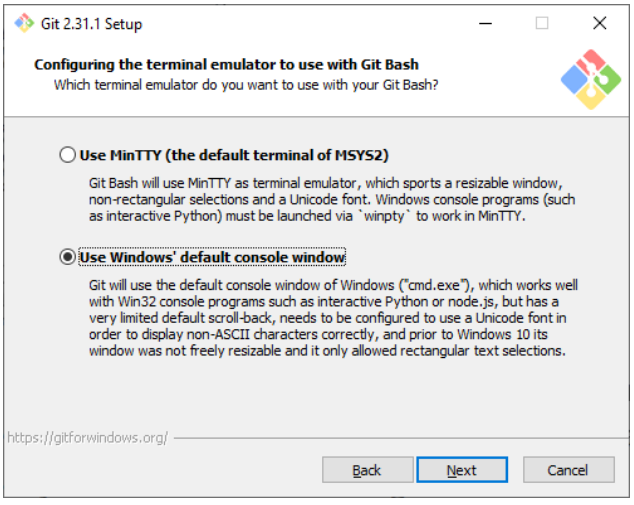
9. En esta ventana, nos indica que el puente de seguridad que va a utilizar Git es a través del protocolo SSL. Clic en Siguiente.



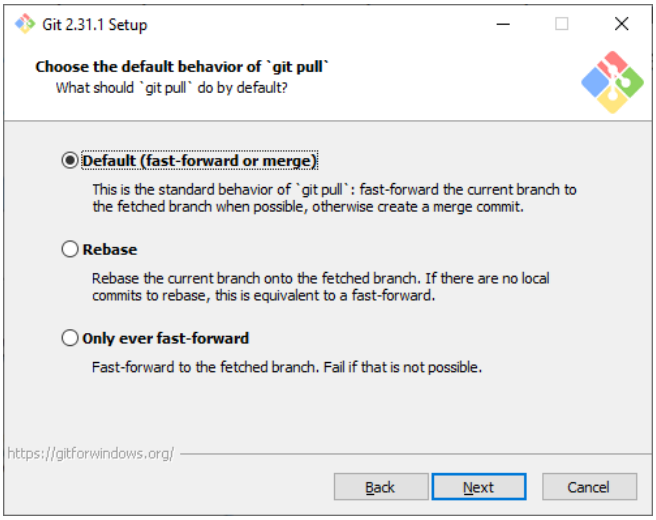
10. En la siguiente ventana de clic en Siguiente. (Deje la opción por defecto)



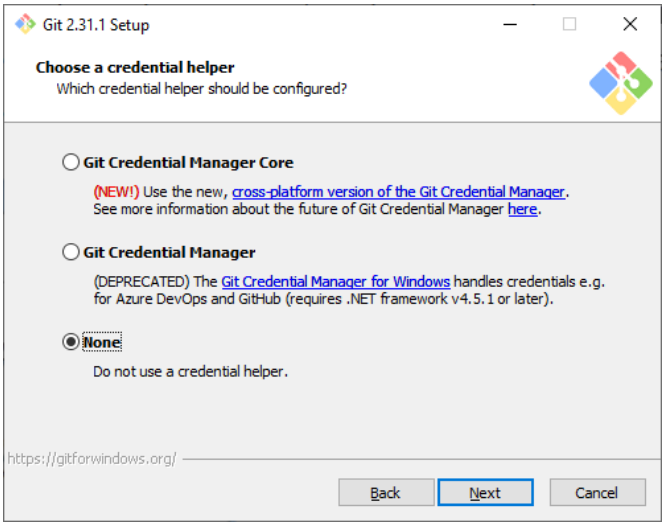
11. En esta ventana seleccionar la opción Use Windows y de clic en siguiente



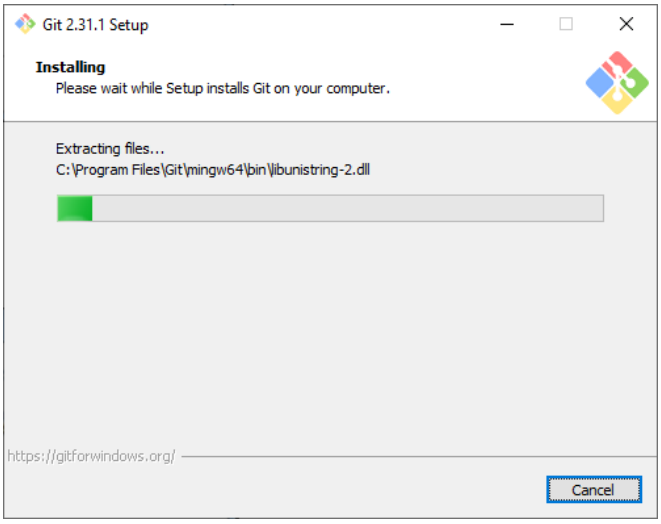
12. En la venta que se muestra dar clic en Siguiente.



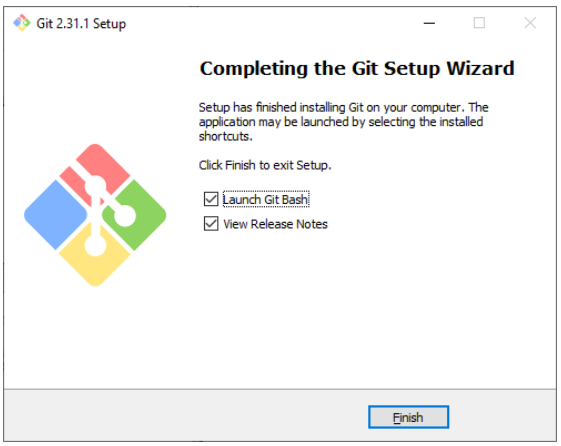
13. En la siguiente ventana seleccionar la opción None y de clic en Siguiente:



14. Clic en las demás ventanas conservando las opciones por defecto y luego Instalar.



15. De clic en Finish



16. Se mostrará la siguiente ventana.

Muestra una lista con los comandos más utilizados en GIT

git help

Clona un proyecto

git clone

Agrega al repositorio TODOS los archivos y carpetas que estén en nuestro proyecto, los cuales GIT no está siguiendo

git add.

Crear localmente un repositorio

git init

Crear una aplicación en Node-Red para leer la temperatura ambiente y convertirla a grados Celsius, así como cuando sobrepase los 30°C nos envíe un mensaje de Hace Calor y encender el Aire acondicionado y que cuando esté por debajo de 22°C nos envíe un mensaje de Hace Frio y encender la calefacción.