**3.2.1 Realice una investigación y responda a las siguientes preguntas, generando una conclusión sobre cada uno de ellos con sus propias palabras:**

**✓ ¿Qué es un Sistema de Control de Versiones?**

(SCV) es un sistema que registra los cambios realizados sobre un archivo o conjunto de archivos a lo largo del tiempo, de modo que se pueda recuperar versiones específicas más adelante. Cualquier tipo de archivo que se encuentre en un ordenador puede ponerse bajo control de versiones.

**✓ ¿En sus palabras porque es necesario el control de versiones?**

un sistema de control de versiones es una elección obligatoria, Ya que permite revertir archivos a un estado anterior, revertir el proyecto entero a un estado anterior, comparar cambios a lo largo del tiempo, ver quién modificó por última vez algo que puede estar causando un problema, quién introdujo un error y cuándo, y mucho más. Usar un VCS también significa generalmente que, si se dañan o se pierden archivos, se recuperar fácilmente

**✓ ¿Sistema de Control de Versiones Locales, Centralizadas y Distribuidas?**

**Sistemas de control de versiones locales**

Un método de control de versiones usado por mucha gente es copiar los archivos a otro directorio (quizás indicando la fecha y hora en que lo hicieron, si son avispados). Este enfoque es muy común porque es muy simple, pero también tremendamente propenso a errores. Es fácil olvidar en qué directorio te encuentras, y guardar accidentalmente en el archivo equivocado o sobrescribir archivos que no querías.



Una de las herramientas de control de versiones más popular fue un sistema llamado rcs, que todavía podemos encontrar en muchos de los ordenadores actuales. Hasta el famoso sistema operativo Mac OS X incluye el comando rcs cuando instalas las herramientas de desarrollo. Esta herramienta funciona básicamente guardando conjuntos de parches (es decir, las diferencias entre archivos) de una versión a otra en un formato especial en disco; puede entonces recrear cómo era un archivo en cualquier momento sumando los distintos parches.

**Sistemas de control de versiones centralizados**

El siguiente gran problema que se encuentra la gente es que necesitan colaborar con desarrolladores en otros sistemas. Para solventar este problema, se desarrollaron los sistemas de control de versiones centralizados (Centralized Version Control Systems o CVCSs en inglés). 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. Durante muchos años éste ha sido el estándar para el control de versiones



Esta configuración ofrece muchas ventajas, especialmente frente a VCSs locales. Por ejemplo, todo el mundo puede saber (hasta cierto punto) en qué están trabajando los otros colaboradores del proyecto. Los administradores tienen control detallado de qué puede hacer cada uno; y es mucho más fácil administrar un CVCS que tener que lidiar con bases de datos locales en cada cliente.

Sin embargo, esta configuración también tiene serias desventajas. La más obvia es el punto único de fallo que representa el servidor centralizado. Si ese servidor se cae durante una hora, entonces durante esa hora nadie puede colaborar o guardar cambios versionados de aquello en que están trabajando. Si el disco duro en el que se encuentra la base de datos central se corrompe, y no se han llevado copias de seguridad adecuadamente, pierdes absolutamente todo —toda la historia del proyecto salvo aquellas instantáneas que la gente pueda tener en sus máquinas locales. Los VCSs locales sufren de este mismo problema— cuando tienes toda la historia del proyecto en un único lugar, te arriesgas a perderlo todo.

**Sistemas de control de versiones distribuidos**

Es aquí donde entran los sistemas de control de versiones distribuidos (Distributed Version Control Systems o DVCSs en inglés). En un DVCS (como Git, Mercurial, Bazaar o Darcs), los clientes no sólo descargan la última instantánea de los archivos: replican completamente el repositorio. Así, si un servidor muere, y estos sistemas estaban colaborando a través de él, cualquiera de los repositorios de los clientes puede copiarse en el servidor para restaurarlo. Cada vez que se descarga una instantánea, en realidad se hace una copia de seguridad completa de todos los datos



Es más, muchos de estos sistemas se las arreglan bastante bien teniendo varios repositorios con los que trabajar, por lo que puedes colaborar con distintos grupos de gente simultáneamente dentro del mismo proyecto. Esto te permite establecer varios flujos de trabajo que no son posibles en sistemas centralizados, como pueden ser los modelos jerárquicos.

**✓ ¿Qué es GIT y su relación con Linus Torvals, Por qué y cómo nació?**

Como muchas de las grandes cosas en esta vida, Git comenzó con un poco de destrucción creativa y encendida polémica. El núcleo de Linux es un proyecto de software de código abierto con un alcance bastante grande. Durante la mayor parte del mantenimiento del núcleo de Linux (1991-2002), los cambios en el software se pasaron en forma de parches y archivos. En 2002, el proyecto del núcleo de Linux empezó a usar un DVCS propietario llamado BitKeeper.

En 2005, la relación entre la comunidad que desarrollaba el núcleo de Linux y la compañía que desarrollaba BitKeeper se vino abajo, y la herramienta dejó de ser ofrecida gratuitamente. Esto impulsó a la comunidad de desarrollo de Linux (y en particular a Linus Torvalds, el creador de Linux) a desarrollar su propia herramienta basada en algunas de las lecciones que aprendieron durante el uso de BitKeeper. Algunos de los objetivos del nuevo sistema fueron los siguientes:

* Velocidad
* Diseño sencillo
* Fuerte apoyo al desarrollo no lineal (miles de ramas paralelas)
* Completamente distribuido
* Capaz de manejar grandes proyectos (como el núcleo de Linux) de manera eficiente (velocidad y tamaño de los datos)

Desde su nacimiento en 2005, Git ha evolucionado y madurado para ser fácil de usar y aún conservar estas cualidades iniciales. Es tremendamente rápido, muy eficiente con grandes proyectos, y tiene un increíble sistema de ramificación (branching) para desarrollo no lineal

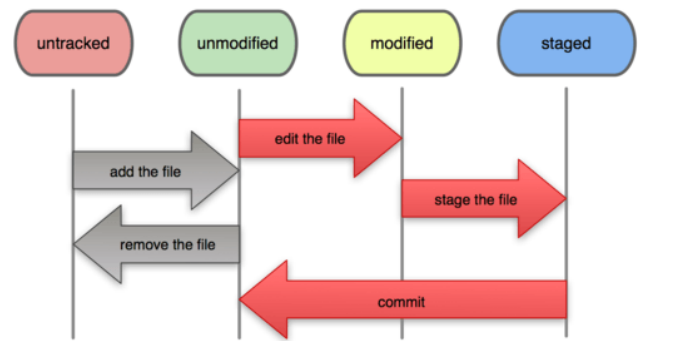
**✓ ¿Por qué usar Git, Características del GIT Frente a otros SCV?.**

Git fue creado pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando éstas tienen un gran número de archivos de código fuente, es decir Git nos proporciona las herramientas para desarrollar un trabajo en equipo de manera inteligente y rápida y por trabajo nos referimos a algún software o página que implique código el cual necesitemos hacerlo con un grupo de personas.

Algunas de las características más importantes de Git son:

* Rapidez en la gestión de ramas, debido a que Git nos dice que un cambio será fusionado mucho más frecuentemente de lo que se escribe originalmente.
* Gestión distribuida; Los cambios se importan como ramas adicionales y pueden ser fusionados de la misma manera como se hace en la rama local.
* Gestión eficiente de proyectos grandes.
* Realmacenamiento periódico en paquetes.

**3.2.3 Según las siguientes imágenes explique de forma de detallada y completa cuál es el esquema de trabajo de GIT, explicando cada uno de los espacios de trabajo, los comandos para pasar de un espacio a otro tanto de adelante como de hacia atrás (revocando cambios).**



**Fase 1: “Working Directory”.**

Aquí es donde podemos hacer cualquier cambio y no afectar nuestro repositorio en lo absoluto.En cuanto modificamos algo de nuestro código, éste tiene status de **modificado**.

Si ejecutamos el comando **git status,**nos mostrará qué archivos han sido modificados (o creados).

“Untracked file” es para archivos recién creados y “modified” para los modificados

Una vez que hemos hecho los cambios necesarios, pasamos nuestros archivos al “**staging area**” con el comando:

**git add nombreDelArchivoModificado.js**

Nota que no es necesario escribir:

**git add nombreDelArchivoModificado.js**

si escribimos:

**git add .**

Agregamos todos los archivos modificados dentro de **working directory** a **staging area**.

Cuando pasas el código de Working Directory a Stagin Area, cambias el estado del código de **modificado** a **preparado**.

**Fase 2: “Staging Area”**

Aquí es donde le podemos dar nombre a nuestra nueva versión. Y crear una “copia” de cómo quedaría nuestro repositorio en producción.

Para pasar nuestro código de staging area al Git Repository (aún no se publica el código en Github), escribimos el siguiente comando:

git commit -m "Nombre de la nueva versión"

Nota que cuando haces el commit el código pasa de estado **preparado** a **confirmado**.

**Fase 3. “Git repository”**

Una vez que el código esta confirmado ya esta listo para sincronizarse con el servidor de Git (github, bitbucket,etc). Para hacer esto, escribimos:

git push -u origin master

Como truquito adicional, te cuento que **-u**sirve para **recordar** “**origin**” que es la dirección de tu repositorio de git y “**master**” que es la branch del repositorio que se esta usando.

Como estamos “recordando” origin y master, para el próximo pusheo, ú**nicamente** podríamos hacer:

git push

Y git ya sabe que significa:

git push origin master

Si estas trabajando en un repositorio con varias personas en algún momento vas a querer descargar las cosas que tus compañeros agreguen o modifiquen.

Para eso ejecutamos:

git pull origin master

**3.3.2 Abra una cuenta en los siguientes servicios de hosting de repositorios GIT y realice una tabla de comparación entre ellos destacando sus ventajas y desventajas.**

|  |  |  |
| --- | --- | --- |
|  | **VENTAJAS** |  |
| **GITLAB** | **GITHUB** | **VISUAL STUDIO** |
| Es gratis. Eso significa que los usuarios pueden tener un número ilimitado de repositorios privados. Esto es la versión comunitaria, y los usuarios tendrán que pagar si requieren la versión empresarial. Esta última agrega algunas características adicionales a la funcionalidad básica que mejora la interacción con herramientas en línea, flujo de trabajo y administración de servidores, entre otras. | * Seguimiento de errores. Esta característica pertenece a las funciones de colaboración y permite mejorar la calidad del código al mantener registro de los errores de software detectados en el proyecto. | Es uno de los lenguajes de uso más extendido, por lo que resulta fácil encontrar información, documentación y fuentes para los proyectos. |
| * Seguimiento de errores y edición de código basado en la web. | * Búsqueda rápida: El repositorio proporciona una estructuración conveniente de proyectos que permite una búsqueda y clasificación eficiente. Además una indexación adecuada permite a los usuarios encontrar cualquier cadena de código en los repositorios públicos. | Integra el diseño e implementación de formularios de Windows. |
| GitLab opera bajo una licencia de código abierto | * Trabajo conjunto: GitHub brinda funciones eficientes para la administración de equipos. | Permite usar con facilidad la plataforma de los sistemas Windows, dado que tiene acceso prácticamente total a la API de Windows, incluidas librerías actuales. |
| * Integración con LDAP que permite localizar y acceder a diversos recursos de internet. GitLab EE soporta varios servicios LDAP y sincronización de grupos | Compatibilidad: Los proyectos con el código en GitHub se pueden personalizar fácilmente a cualquier servicio host en la nube. | Si bien permite desarrollar grandes y complejas aplicaciones, también provee un entorno adecuado para realizar pequeños prototipos rápidos. |
|  | **DESVENTAJAS** |  |
| **GITLAB** | **GITHUB** | **VISUAL STUDIO** |
| Interface relativamente lenta | El servicio no es completamente gratuito. Para acceder a todas las funciones de GitHub, se debe actualizar a un usuario Premium.   * Limitaciones de tamaño. Los archivos no pueden ser mayores a 100 MB mientras que el repositorio puede alojar 1 GB de información. | Pobre soporte para programación orientada a objetos |
| * Frecuentes problemas técnicos con los repositorios | * El servicio no es completamente gratuito. Para acceder a todas las funciones de GitHub, se debe actualizar a un usuario Premium. | Incapacidad para crear aplicaciones multihilo, sin tener que recurrir a llamadas de la API de Windows. |