****

**UNIVERSIDAD TECNOLÓGICA DE QUERÉTARO**

**Diplomado en Software Embebido**

Módulo 1: Bases de Ingeniería de Software

Tema: 1.4.4 “Herramientas para Control de Versiones de Software”

Juan Pablo Cruz Gutiérrez

Versión 2.0

1. **Introducción**

En el mundo del Software siempre se buscan maneras para mejorar y optimizar el desarrollo de código, existen cosas tan simples que muchas veces pasan desapercibidas, como un control de versiones, utilizado con el fin de gestionar ágilmente los cambios en un código fuente.

Hay muchas opciones para manejar versiones de software, cada persona o cada empresa puede generar sus propias reglas que pueden sirven como guía.

Entre las formas más básicas que existen para el control de versiones están las siguientes, manejarlas por número, estabilidad o fecha, pero los principales problemas al usar estos métodos, es que son propensos a errores, perdida de datos y son difíciles de manejar cuando hay más de una persona que modifica el proyecto.

Actualmente no existe un estándar o una normativa oficial para el control de versiones, pero existen diferentes herramientas para control de versiones que nos facilitan esta gestión.

¿Qué es una herramienta para control de Versiones?

Es un sistema que permite realizar seguimiento de una colección de archivos, además incluye la funcionalidad de revertir la colección de archivos actual a una colección anterior, cada versión podría considerarse como una fotografía en un momento determinado, algunas de estas herramientas son:

* PlasticSCM
* Git
* CSV
* SVN
* Mercurial
* Bazaar
* Monotone
* Synergy
* PVCS, etc.

Estos programas fueron ideados para gestionar los cambios de código fuente o poder revertirlos ágilmente, cuyo ámbito ha sido ampliado pasando del concepto control de versiones al de gestión de configuración de software, en el que se engloban todas las actividades que pueden realizarse por un equipo sobre un gran proyecto software u otra actividad que genere ficheros digitales (por ejemplo: documentos, ofertas, dibujos, esquemas, etcétera), la tabla 1.4.4.1 muestra un comparativo entre algunas de las herramientas que existen actualmente.

Tabla 1.4.4. 1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Software | Desarrollo | Soporte | Modelo de repositorio | Sistemas Operativos | Costo |
| Plastic SCM | Codice Software | Activo | Cliente servidor y Distribuido | Linux, Windows, OS X | Free for up to 15 users; else starting at $595 per seat |
| Git | Junio Hamano | Activo | Distribuido | POSIX, Windows, OS X | Free |
| CVS | The CVS Team | Hay soporte, pero no se han agregado actualizaciones | Cliente-Servidor | Unix-like, Windows, OS X | Free |
| Subversion (SVN) | Apache Software Foundation[8] | Activo | Cliente-Servidor | Unix-like, Windows, OS X | Free |
| Mercurial | Matt Mackall | Activo | Distribuido | Unix-like, Windows, OS X | Free |
| GNU Bazaar | Canonical Ltd. | Activo | Cliente servidor y Distribuido | Unix-like, Windows, OS X | Free |
| Monotone | Nathaniel Smith, Graydon Hoare | Activo | Distribuido | Unix-like, Windows, OS X | Free |
| Synergy | IBM Rational | Activo | Cliente servidor y Distribuido | Linux, Windows, Unix-like | Non-free Contact IBM Rational[9] |
| PVCS | Serena Software | Activo | Cliente-Servidor | Windows, Unix | Non-free |

Dependiendo de lo complejo que sea un proyecto, se recomienda contar con un sistema de control de versiones, ya que las ventajas que este añade son muy importantes, sobre todo cuando se tiene un equipo de trabajo muy grande, estos sistemas nos ayudan para tener trazabilidad en las modificaciones que se han agregado, o en caso de depuración es útil para excluir repositorios de manera ágil y así identificar más fácil que repositorio hace que nuestro sistema se comporte distinto a lo esperado.

La tabla que se presentó anteriormente contiene solo una breve descripción a continuación se dará una descripción más detalla de estas herramientas.

1. **Tipos de control de versiones**

**El control de versiones local**, es usado por mucha gente y consiste en copiar los archivos a otro directorio (quizás indicando la fecha y hora en que se generó el archivo). Es un método muy simple pero como ya se había mencionado es altamente propenso a errores, pues es fácil olvidar en que directorio te encuentras y guardar accidentalmente en el archivo equivocado o sobrescribir en el archivo diferente al deseado.

Para enfrentar este problema hace tiempo se desarrollaron herramientas que contenían una simple base de datos en la que se llevaba registro de todos los cambios realizados sobre el archivo, ver esto en la Figura 1.4.4.1

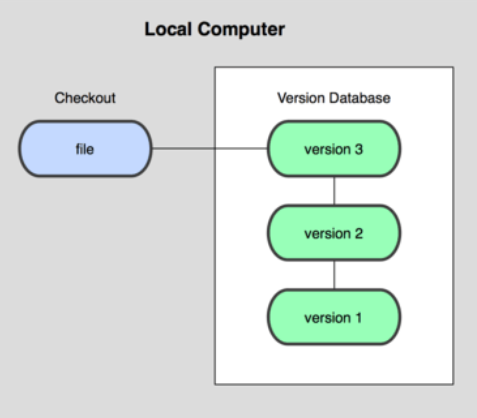


Figura 1.4.4. 1(Método de control de versiones local)

**Control de Versiones Centralizada**, tienen la característica que funciona como un entorno clásico cliente-servidor, es decir se tiene un servidor en el que se aloja el repositorio del proyecto, con toda la información de los cambios.

En este entorno el usuario trabajara con una copia del servidor en una revisión determinada, normalmente se trabaja con la más actualizada, el usuario hace cambios sobre esa copia y cuando se considera que ha terminado con esa modificación la sube al servidor, el cual se encarga de unir los cambios al repositorio central e informar de los errores que se pudieran dar ver Figura 1.4.4.2

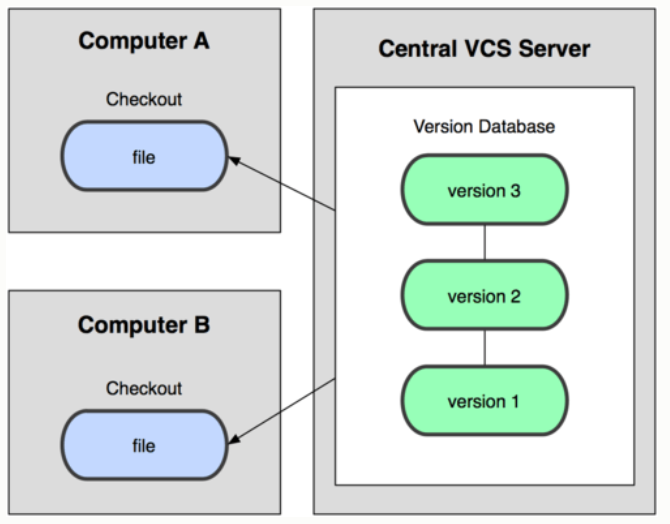


Figura 1.4.4. 2 (Método de control de versiones Centralizada)

**El Control de versiones distribuida** es un sistema donde hay un servidor central y cada usuario puede hacer una copia completa del directorio central, y por ser una copia posee las mismas funcionalidades que el directorio original, es decir, contiene la historia completa de la colección de archivos con la propiedad que el directorio puede intercambiar entre sus versiones, así los usuarios no sólo descargan la última versión de los archivos, también si un servidor muere, y estos sistemas estaban colaborando a través de él, cualquiera de los repositorios de los usuarios puede copiarse en el servidor para restaurarlo. Funcionando, así como una copia de seguridad completa de todos los datos ver Figura 1.4.4.3

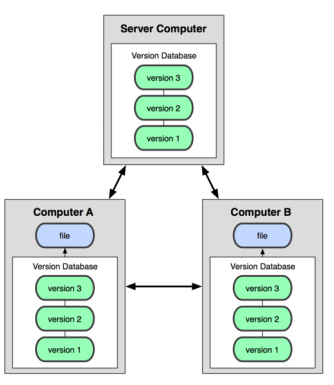


Figura 1.4.4. 3 (Método de control de versiones Distribuida)

En la imagen se muestra como es un sistema de control de versiones distribuida, en cada directorio A y B existe una copia exacta del directorio padre.

1. **Herramientas para control de cambios de Software**

Plastic SCM (Codice Software, 2018): Es un sistema de control de versiones distribuido desarrollado por la empresa Española Códice Software tiene como objetivos fundamentales, dar un mayor soporte al desarrollo paralelo, creación de ramas, seguridad y desarrollo distribuido.

Para favorecer el desarrollo paralelo, dividiendo el desarrollo en distintas ramas, siguiendo una determinada política de uso, protección, desprotección, contenidos. La principal diferencia entre el modelo de desarrollo paralelo de Plastic y otros estriba en que en lugar de realizar una copia de todo (o solo los metadatos) a cada nueva rama que se genera, las ramas son creadas con objetos vacíos, solo cuando un ítem es modificado, se asigna una nueva revisión a la rama. De este modo la rama solo contiene ficheros o directorios que se han modificado con respecto a su rama padre.

Debido a la infraestructura de Plastic para el desarrollo paralelo, puede manejar miles de ramas en un solo repositorio sin perdida notable de rendimiento.

Git (Git-fast version control, 2018): Es un software de código abierto para control de versiones diseñado por Linus Torvalds (Creador de Linux), basado en el modelo de control distribuido ver Figura 1.4.4.3, diseñado para manejar desde pequeños hasta grandes proyectos de manera rápida, eficiente y confiable.

Su propósito es llevar registro de los cambios de archivo de computadora y coordinar el trabajo que varias personas realicen sobre archivos compartidos.

Al principio Git se pensó como un motor de bajo nivel sobre el cual otros pudieran escribir la interfaz de usuario. Sin embargo, Git sea convertido desde entonces en un sistema de control de versiones con funcionalidad plena.

Es muy rápido y eficiente aun con grandes proyectos y tiene un increíble sistema de ramificaciones, es muchos sistemas de control de versiones este proceso puede ser costoso, pues a menudo se requiere crear una nueva copia del código, lo cual puede tomar mucho tiempo cuando se trata de proyectos grandes, muchos usuarios resaltan esto como uno de los puntos más fuertes de Git, haciendo así las operaciones de ramificación casi instantáneo.

CVS (CVS - Concurrent Versions System, 2005-2006): Es uno de los sistemas para control de versiones de software más viejos, es una herramienta fácil de usar, mantiene el registro de todo el trabajo y los cambios en los ficheros (Codigo fuente principalmente, en un único archivo para cada fichero correspondiente), permite que diferentes desarrolladores colaboren a la vez.

Es un sistema centralizado ver Figura 1.4.4.2, en donde un servidor guarda las versiones actuales del proyecto y su historial, del servidor se crea una copia con la que se puede trabajar e ingresar los cambios con comandos GNU, varios usuarios pueden sacar copias del proyecto al mismo tiempo. Posteriormente, cuando incluyan sus modificaciones al proyecto, el servidor trata de acoplar las diferentes versiones. Si esto falla, debido a que dos clientes tratan de cambiar la misma línea en un archivo en particular, entonces el servidor deniega la segunda actualización e informa al cliente sobre el conflicto, que el usuario deberá resolver manualmente.

SVN (Apache Software Foundation, 2017): Apache subversión es una herramienta de código abierto para control de versiones, basada en un repositorio cuyo funcionamiento se asemeja al de un sistema de ficheros. Es software libre bajo una licencia de tipo Apache/BSD.

Es un sistema centralizado y utiliza el concepto de revisión para guardar los cambios producidos en el repositorio. Entre dos revisiones sólo guarda el conjunto de modificaciones, optimizando así al máximo el uso de espacio en disco. SVN permite al usuario crear, copiar y borrar carpetas con la misma flexibilidad con la que lo haría si estuviese en su disco duro local. Dada su flexibilidad, es necesaria la aplicación de buenas prácticas para llevar a cabo una correcta gestión de las versiones del software generado.

Subversion puede acceder al repositorio a través de redes, lo que le permite ser usado varios usuarios en diferentes computadoras.

Mercurial (Mercurial, 2017): Es un sistema de control de versiones distribuido, esta implementado en Python, pero también incluye una parte de implementación en C. Mercurial fue desarrollado para ser usado sobre GNU/Linux pero ha sido adaptado para Window y Mac OS y la mayoría de otros sistemas Unix.

Las principales metas de desarrollo de Mercurial incluyen un gran rendimiento y escalabilidad, desarrollo completamente distribuido, sin necesidad de un servidor, es software libre, ayuda para tener una gestión robusta de archivos tanto de texto como binarios y capacidades avanzadas de ramificación e integración, todo ello manteniendo sencillez conceptual.

Para el acceso a repositorios mediante red, Mercurial usa un protocolo eficiente, basado en HTTP, que persigue reducir el tamaño de los datos a transferir, así como la proliferación de peticiones y conexiones nuevas.

GNU Bazaar (Bazaar, 2016): Es un sistema de control de versiones del tipo distribuido, es multiplataforma, que ayuda a dar seguimiento entre el historial del proyecto a través del tiempo y facilita el trabajo conjunto en proyectos de software, es parte del proyecto GNU, es un software libre, al igual que mercurial es basado en Python.

Monotone (monotone, 2018): Es una herramienta para control de versiones de software, de código abierto, registra revisiones de ficheros, agrupa conjuntos de revisiones y mantiene históricos tras cambios de nombres, su principio de operación es distribuido haciendo un uso intensivo primitivas criptográficas para trazar revisiones de ficheros y para autentificar acciones de usuarios, a pesar que es una herramienta muy estable estas acciones hacen que la herramienta sea lenta.

Cada usuario mantiene su propio almacén de revisiones históricas en una base de datos SQLite local.

Synergy (Rational Synergy, 2012): es una herramienta de control de cambios de software distribuida, que proporciona capacidades de gestión en el desarrollo de software, también proporciona el repositorio de la herramienta de administración de cambios conocida como Cambio Racional. Juntas, estas dos herramientas forman un entorno de gestión de configuración y gestión de configuración integrado que se utiliza en organizaciones de desarrollo de software que necesitan procesos controlados de SCM y una comprensión de lo que hay en una compilación de su software.

1. **GIT**

Para el desarrollo de este Modulo se ha decidido usar GIT, debido a que es usado por muchos proyectos populares de código abierto como Android o Eclipse además cuenta con soporte y es gratuito.

la principal diferencia entre Git y cualquier otro sistema de control de versiones es cómo Git modela sus datos. Conceptualmente, la mayoría de los demás sistemas almacenan la información como una lista de cambios en los archivos. Estos sistemas modelan la información que almacenan como un conjunto de archivos y las modificaciones hechas sobre cada uno de ellos a lo largo del tiempo. En cambio, Git modela sus datos más como un conjunto de instantáneas de un mini sistema de archivos. Cada vez que confirmas un cambio, o guardas el estado de tu proyecto en Git, él básicamente hace una foto del estado de todos tus archivos en ese momento, y guarda una referencia a esa instantánea. Para ser eficiente, si los archivos no se han modificado, Git no almacena el archivo de nuevo —sólo un enlace al archivo anterior idéntico que ya tiene almacenado, ver figura 1.4.4.3

Luego de clonar o crear un repositorio el usuario tiene una copia completa del

repositorio, y puede realizar operaciones de control de versiones contra este

repositorio local como, por ejemplo :

* Modificar una serie de archivos en el directorio de trabajo (Working directory).
* Añadír modificaciones de los archivos al área de preparación (Staging area).
* Confirmar los cambios y tomarlos tal y como están en el área de preparación, y almacenarlos de manera permanente en el directorio de Git (git directory)

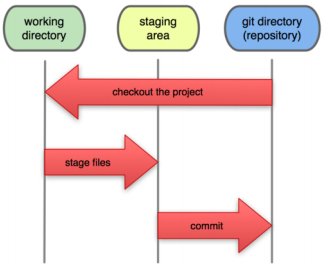


Figura 1.4.4. 4 (Operaciones locales)

Hay dos tipos de repositorios en Git:

• Repositorios “bare”, son generalmente usados en los servidores, aunque puede generarse en cualquier directorio local de quien desee crearlo. Se utiliza para compartir cambios provenientes de diferentes desarrolladores, no se utiliza para crear u compartir cambios desde sí mismo, ya que no

cuenta con esas funcionalidades.

• Repositorios de trabajo, permiten crear archivos nuevos, generar cambios

a los archivos existentes y a su vez crear nuevas versiones en el repositorio.

1. **Comandos de Git**

**Git init** - inicializar una carpeta como un repositorio Git.

Para crear un repositorio en una carpeta existente de archivos, podes ejecutar el comando git init en esa carpeta

$ git init

Ahora se puede ver que hay una subcarpeta oculta llamada “.git” en el proyecto.

Este es tu repositorio donde se almacenan todos los cambios del proyecto.

**Git clone** - copiar un repositorio Git.

Si tienes que colaborar con alguien en un proyecto, o si deseas obtener una copia de un proyecto, debes ejecutar el comando git clone [url] con la URL del proyecto que deseas copiar, ejemplo:

$ git clone git://github.com/schacon/simplegit.git

**Git add** - agregar los contenidos de archivos al área de preparación.

En Git es necesario agregar previamente los cambios realizados al área de preparación, para luego poder hacer el commit correspondiente.

$ git add **file.ext**

**Git status** – Indica el estado de los archivos en el directorio de trabajo y en el área de preparación.

Se puede usar este comando para ver el estado de su área de preparación comparado con el código que se encuentra en su directorio de trabajo. Usando la opción “-s” nos mostrará la forma abreviada del informe.

$ git status -s

**Git diff** - muestra las diferencias de cambios que no están en el área de preparación.

Sin ningún argumento adicional, un simple git diff mostrará en formato unificado, qué código o contenido ha cambiado en el proyecto desde el último commit

$ git diff

**Git diff –stat** – muestra una lista de cambios en lugar de un diff completo por archivo

$ git diff –stat

La opción - stat, lo que nos dará un resumen de los cambios en su lugar.

**Git commit** – graba una instantánea del área de preparación.

Para guardar los cambios es necesario seguir el siguiente proceso, donde se finaliza con commit que es la confirmación de los cambios

$ git config - global user.name 'Tu nombre'

$ git config --global user.email [tumail@dominio.com](mailto:tumail@dominio.com)

$ git add **file.ext**

$ git status -s

$ git commit -m ' Agrega mis cambios al archivo'

**Git reset** – deshace cambios y commits

Se usa para deshacer el último commit y sacar del área de preparación los archivos modificados.

$ git reset HEAD -- file.ext

lo que hace en realidad es resetear las entradas del archivo en el “index” para que sea igual que en el commit anterior.

**Git branch** (nombre del branch) - crea una nueva rama

$ git branch rama

$ git Branch sin argumento lista las ramas existentes.

**Git checkout** -b (nombre del branch) - crear y cambiar inmediatamente a una rama

$ git checkout -b nueva\_rama

Con argumento -b crea una nueva rama y se cambia de inmediato a esta nueva rama

$ git checkout rama

Se cambia a la rama especificada.

**Git merge** - fusionar una rama en tu contexto actual

Una vez que tengas el trabajo aislado en una rama, es muy probable que desees incorporarlo en la rama principal. Puedes combinar cualquiera de las ramas en su rama actual con el comando git merge

$ git merge rama

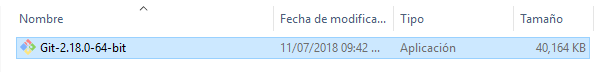
Hace el merge de la rama principal master con la rama que se le indique

1. **Practica Usando GIT**

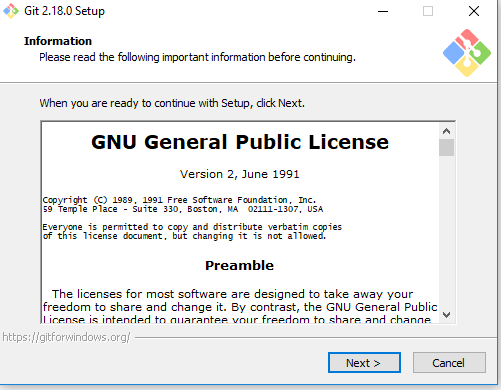
A continuación, se describen varias practicas con las que se pretende, se entienda la aplicación de los conceptos anteriores y se pongan en práctica los comandos mostrados.

1. **Descargar la herramienta:** puede ser directo desde la página de descargas de Git <https://git-scm.com/downloads>, o entrando a la plataforma de moodle, en el módulo 1 sección 1.4.4. Herramientas para Control de Versiones de Software entrar a la carpeta Herramientas, y descargar la carpeta GitBash.
2. **Instalación:** Una vez que se tiene el ejecutable ver figura 1.4.4.5 (la versión puede variar)

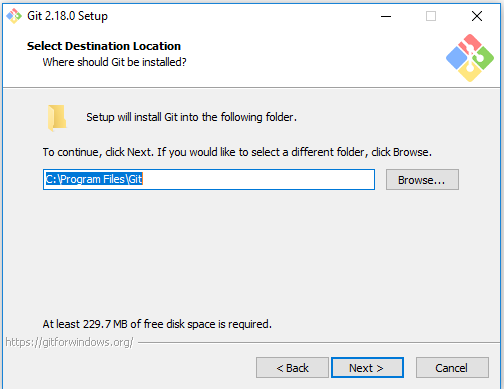
* Aquí se muestran los pasos a seguir para la instalación de Git.



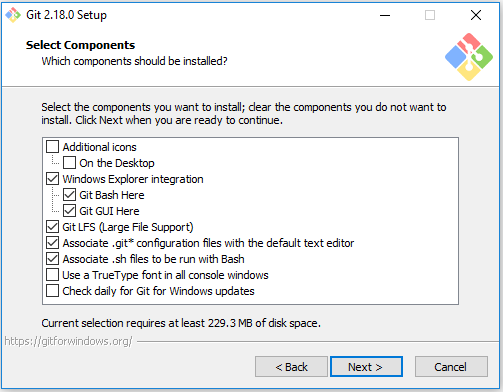
**Paso 1**



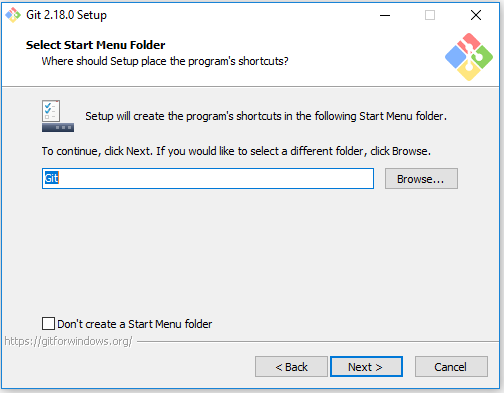
**Paso 2**



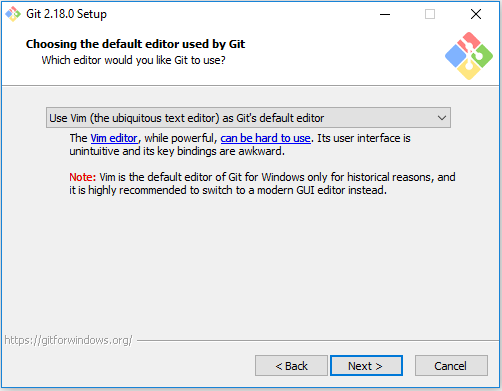
**Paso 3**



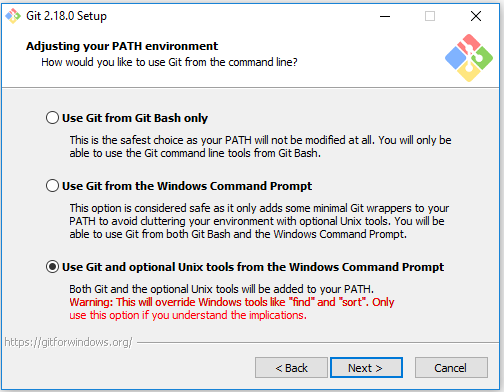
**Paso 4**



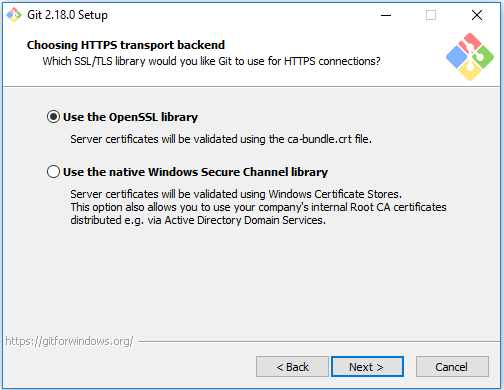
**Paso 5**

****

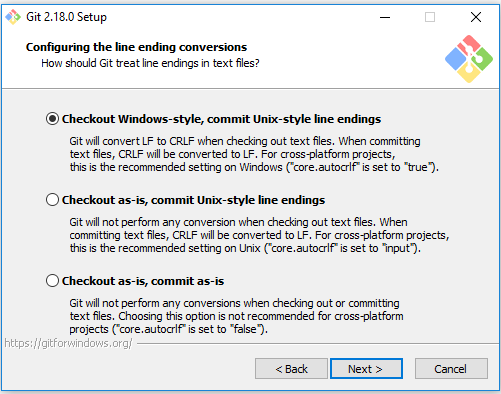
**Paso 6**



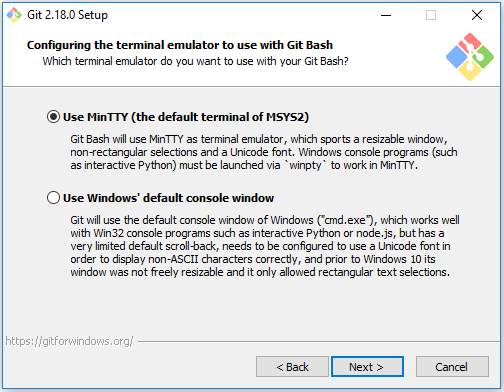
**Paso 7**

****

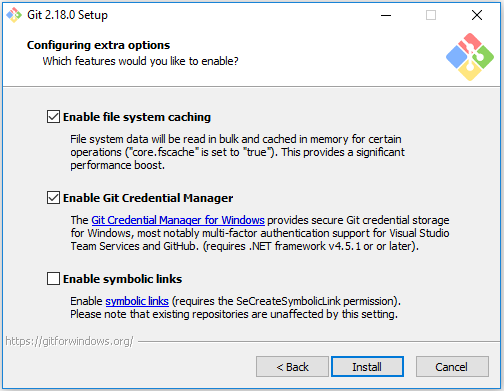
**Paso 8**

****

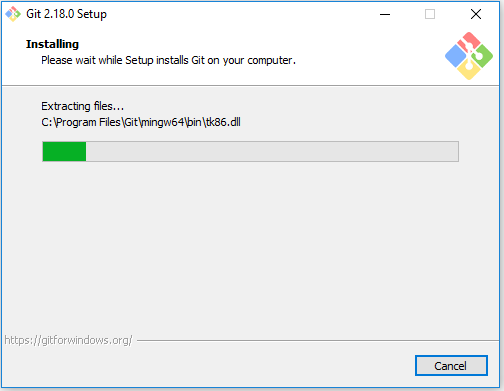
**Paso 9**

****

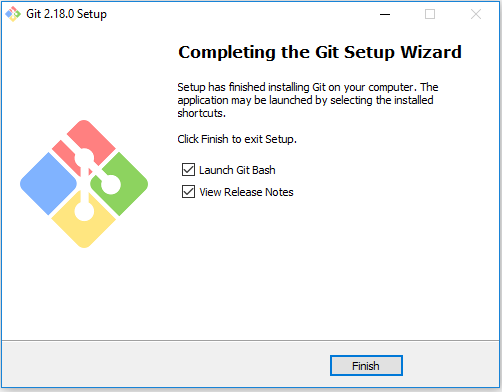
**Paso 10**

****

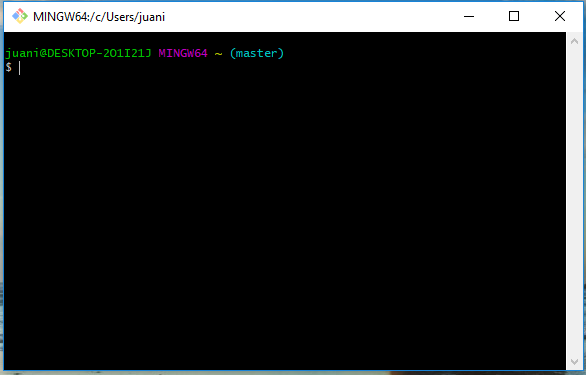
**Paso 11**

****

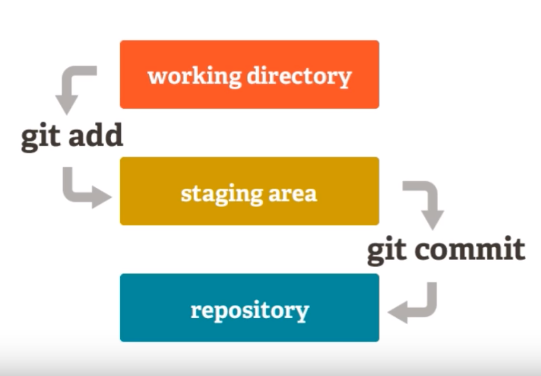
**Paso 12**

****

**Paso 13**



**Paso 14**



1. **Preguntas**
2. ¿Qué es un control de Versiones?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. ¿Qué es una Herramienta para control de versiones?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. ¿Cómo podría llevar un control de versiones local?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. ¿Qué es un sistema de control de versiones centralizado?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. ¿Qué es un sistema de control de versiones distribuido?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. ¿Por qué se recomienda usar un control de versiones?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. ¿Menciona al menos 4 herramientas para control de cambios?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. ¿Para qué sirve el comando **Git init**?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. ¿Escriba la secuencia de comandos básica, para agregar cambios a un archivo, usando Git?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. ¿Qué comando se usa para deshacer el último commit?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Bibliography

*Apache Software Foundation*. (2017). Obtenido de https://subversion.apache.org/

*Bazaar*. (2016). Obtenido de http://bazaar.canonical.com/en/

*Codice Software*. (2018). Obtenido de https://www.plasticscm.com/

*CVS - Concurrent Versions System*. (2005-2006). Obtenido de https://www.nongnu.org/cvs/

*Git-fast version control*. (2018). Obtenido de https://git-scm.com/

Github, S. (s.f.). Obtenido de https://services.github.com/on-demand/downloads/github-git-cheat-sheet.pdf

*Mercurial*. (2017). Obtenido de https://www.mercurial-scm.org/

*monotone*. (2018). Obtenido de https://www.monotone.ca/

PVCS. (s.f.). Obtenido de https://en.wikipedia.org/wiki/PVCS

Rational, S. (s.f.). Obtenido de https://www.ibm.com/us-en/marketplace/rational-synergy

Wikipedia, C. (s.f.). Obtenido de https://en.wikipedia.org/wiki/Comparison\_of\_version\_control\_software