**“EXPOSICIÓN DE CONTROL DE VERSIONES DISTRIBUIDO”**

**UNIDAD I**

**ALUMNO(S):**

**LEONEL OSVALDO GUTIÉRREZ LUGO (20550187)**

**YENNYFER GPE. AMAYA CORRAL (20550208)**

**JUAN MANUEL CRISTERNA PRADO (C18060599)**

**DOCENTE:**

**CARLOS HUMBERTO RUBIO RASCON**

**CHIHUAHUA, CHIHUAHUA.**

**29 DE AGOSTO DEL 2022**

Índice

[Introducción. 2](#_Toc112679671)

[Conceptos basicos 2](#_Toc112679672)

[Objetivo 2](#_Toc112679673)

[Desarrollo 3](#_Toc112679674)

[Breve Historia 3](#_Toc112679675)

[Comandos Básicos 4](#_Toc112679676)

[Sistema distribuido vs centralizado 5](#_Toc112679677)

[Conclusión 6](#_Toc112679678)

[Referencias 6](#_Toc112679679)

# Introducción.

Los Sistemas de Control de Versiones permiten gestionar los cambios realizados en el código fuente de programas o documentos.

[GitHub](https://github.com/) es una empresa de alojamiento de proyectos de software basados en git fundada en 2008. El alojamiento es gratuito para los proyectos libres y públicos y de pago para el resto. GitHub se ha convertido en el mayor repositorio de proyectos de software libre.

El control de versiones distribuido permite a muchos [desarrolladores](https://es.wikipedia.org/wiki/Desarrollador_de_software) de software trabajar en un proyecto común sin necesidad de compartir una misma [red](https://es.wikipedia.org/wiki/Red_inform%C3%A1tica). Las revisiones son almacenadas en un sistema de [control de versiones](https://es.wikipedia.org/wiki/Control_de_versiones) [distribuido](https://es.wikipedia.org/wiki/Computaci%C3%B3n_distribuida) (DVCS, por sus siglas en inglés).

# Conceptos basicos

**Repositorio:**

El repositorio es el lugar en el que se almacenan los datos actualizados e históricos de cambios.

**Revisión:**

Una revisión es una versión determinada de la información que se gestiona. Hay sistemas que identifican las revisiones con un contador. Hay otros sistemas que identifican las revisiones mediante un código de detección de modificaciones.

**Etiqueta:**

Los tags permiten identificar de forma fácil revisiones importantes en el proyecto. Por ejemplo, se suelen usar tags para identificar el contenido de las versiones publicadas del proyecto.

**Rama:**

Un conjunto de archivos puede ser ramificado o bifurcado en un punto en el tiempo de manera que, a partir de ese momento, dos copias de esos archivos se pueden desarrollar a velocidades diferentes o en formas diferentes de forma independiente el uno del otro.

**Tronco:**

La única línea de desarrollo que no es una rama (a veces también llamada línea base, línea principal o máster).

# Objetivo

Al finalizar la lectura de este documento se tendrá un entendimiento básico del control de versiones distribuido y cómo funcionan algunos comandos básicos del sistema git con el fin de poder comenzar con la práctica.

# Desarrollo

Se llevaron a cabo varias investigaciones para recabar la información solicitada en clase, tal como la historia, comandos más usados, etc. Tras recabar y analizar, se obtuvo la información indicada la cual se agregó a este archivo y la presentación.

# Breve Historia

**CVS:** primer sistema de control de versiones con acceso a redes de área amplia para desarrolladores. Con el nombre de «Concurrent Versions System», cada fichero contaba con su propia versión que era un número y, por lo tanto, sabíamos que la versión 6 del fichero era posterior a la versión 5 del mismo. CVS utiliza una arquitectura cliente-servidor: un servidor guarda la(s) versión(es) actual(es) del proyecto y su historial. Los clientes se conectan al servidor para sacar una copia completa del proyecto. El protocolo CVS no provee una manera de que los directorios puedan ser renombrados, cada archivo en cada subdirectorio debe ser eliminado y re-agregado con el nuevo nombre. Soporte limitado para archivos Unicode con nombres de archivo no ASCII. sólo versiona ficheros, no directorios; ofrece ramificaciones, etiquetado, y un buen rendimiento en la parte del cliente, pero no maneja muy bien ficheros grandes ni ficheros binarios.

**SUBVERSION:** La creación de repositorios es más fácil. Existía una versión global a nivel de repositorio, lo que te podía dar una idea de hacía cuánto se había creado un fichero. La creación de ramas y tags. Cuando querías hacer un commit y no podías, él mismo hacía un update del repositorio local.

**BitKeeper**: En el 2005, herramienta dejó de ser ofrecida de manera gratuita. 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 mientras usaban BitKeeper.

**GIT:**Muchos de nosotros descubrimos qué era eso de utilizar un sistema de control de versiones distribuido y cómo el no depender de un repositorio central te quitaba la presión de tener que estar todo el rato actualizando tu código con lo hecho por el resto de compañeros. La facilidad de crear ramas de desarrollo te dejaba alternar entre escribir features nuevas o corregir bugfixes en cuestión de segundos, sin tener que depender, nuevamente, de la lentitud de un repositorio central.

**Bazaar:** Bazaar es un sistema de control de versiones distribuido patrocinado por Canonical Ltd., diseñado para facilitar la contribución en proyectos de software libre y opensource. Puede ser usado por un usuario único trabajando en múltiples ramas de un contenido local, o por un equipo colaborando a través de la red.

Bazaar está escrito en lenguaje de programación Python y tiene versiones empaquetadas para la mayoría de distribuciones GNU/Linux, así como Mac OS X y MS Windows. Bazaar es software libre y parte del proyecto GNU.

# Comandos Básicos

En este proyecto nos basaremos en el sistema de control de versiones distribuido con más control del mercado mundial del momento, git. Este es, a día de hoy, un requerimiento para el desarrollo no sólo de software, sino también de cualquier tupo de proyecto que necesite un control de versiones en el que se necesite un registro práctico y fácil de entender.

Ahora bien, aquí mostramos algunos de los comandos básicos de git que nos servirán como introducción a este.

* 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
* git init [nombre del proyecto]. Creará un nuevo repositorio local GIT.
* git status. muestra la lista de los archivos que se han cambiado junto con los archivos que están por ser preparados o confirmados.
* 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 <tempt.txt>
* git commit.. Una vez que se llega a cierto punto en el desarrollo, queremos guardar nuestros cambios. Git commit es como establecer un punto de control en el proceso de desarrollo al cual puedes volver más tarde si es necesario. También necesitamos escribir un mensaje corto para explicar qué hemos desarrollado o modificado en el código fuente: git commit –m “mensaje”
* git push. Después de haber confirmado tus cambios, el siguiente paso que quieres dar es enviar tus cambios al servidor remoto. Git push envía tus commits al repositorio remoto: git push <nombre-remoto> <nombre-de-tu-rama>
* 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: git checkout <nombre-de-la-rama>
* git pull. Se utiliza para recibir actualizaciones del repositorio remoto. Este comando es una combinación del git fetch y del git merge lo cual significa que cundo usemos el git pull recogeremos actualizaciones del repositorio remoto (git fetch) e inmediatamente aplicamos estos últimos cambios en local (git merge). Un ejemplo sería: git pull <nombre-remoto>
* git merge. Cuando ya hayas completado el desarrollo de tu proyecto en tu rama y todo funcione correctamente, el último paso es fusionar la rama con su rama padre (dev o master). Esto se hace con el comando git merge. Git merge básicamente integra las características de tu rama con todos los commits realizados a las ramas dev (o master): git merge <nombre-de-la-rama>
* git stash. guardará momentáneamente los cambios que no están listos para ser confirmados.
* git revert. En ocasiones necesitaremos deshacer los cambios que hemos hecho. Una manera segura para deshacer nuestras commits es utilizar git revert.

# Sistema distribuido vs centralizado

Ahora, desde que muchos grandes proyectos optaran por los sistemas de control de versiones distribuidos, cada vez hay más gente que se va cambiando a ellos y, sobre todo, proyectos grandes.

Centralizado (limitaciones y características):

* El sistema servidor es un repositorio, como los que mantienen los clientes, pero perfectamente sincronizado y sin que dé lugar a conflictos. Es la copia maestra de los datos.
* Cuando un sistema web quiere hacer un listado, puede tomar los datos de este servidor y siempre serán seguros, con lo que no tendrá que resolver conflictos, ni tendrá que hacer mezclas.
* Una copia local debe de poder mezclarse con el repositorio central cuando queramos publicar un conjunto de cambios o cuando queramos tomar la última versión publicada en concordancia con nuestra copia local.
* Es normal ver en muchos de estos sistemas ramificaciones, versiones, etiquetas, o similares, a modo de tener varias copias según nos interese. Estas ramificaciones están en el servidor y en algunos casos puede llegar a ser muy costosa su diferenciación.
* Distribuido (ventajas):
* Disponer de forma distribuida de la información del repositorio al completo, tanto de forma local, como a través de los demás componentes del grupo.
* Cada cambio se va replicando entre los demás equipos distribuidos, a modo de que puedan emplear esos datos y actualizarlos en sus sistemas.
* El sistema de control de versiones distribuido ha sido pensado con la forma de trabajo basada en ramas, unión central en una sola versión (trunk) y liberaciones (o tags). Con lo que cada rama puede identificarse como cada copia distribuida que se use.
* Una máquina servidora puede emplearse, al estar siempre conectada, como otro punto de sincronización, con la ventaja de que, aunque cayera, mientras haya más miembros conectados, el sistema siempre se mantiene activo y con buen ancho de banda.

# Conclusión

Tras las investigaciones realizadas, se llegó a la conclusión que el uso de los sistemas distribuidos es más indicado, sobre todo, para proyectos con gran afluencia de desarrolladores. Por lo que, se entiende el porqué de su aplicación del modelo en los grandes proyectos ya que tiene ventajas tales como un historial de cambios, la posibilidad de que varias personas trabajen en el proyecto al mismo tiempo y el enlazamiento con software de gestión de proyecto, etc.

# Referencias

* Chacón, Scott; Straub, Ben (2014). "Introducción - Acerca del control de versiones" . Pro Git (2ª ed.).
* "Introducción al control de versiones distribuidas (ilustrado)" . www.betterexplained.com
* "Qué es el control de versiones: centralizado frente a DVCS" . www.atlassian.com
* “En resumen: control de versiones” https://www.encora.com
* “Sistemas de control de versiones: ¿Centralizados o distribuidos?” <https://altenwald.org>
* Sergio Gómez. (2022). *Sistemas de control de versiones - Taller de Git*. aulasoftwarelibre. https://aulasoftwarelibre.github.io/taller-de-git/cvs/