TEMA VI

CONTROL DE VERSIONES

1. INTRODUCCIÓN

Con el término versión, se hace referencia a la evolución de un único elemento, o de cada elemento por separado, dentro de un sistema en desarrollo.
Siempre que se está realizando una labor, sea del tipo que sea, es importante saber en cada momento de qué estamos tratando, qué hemos realizado y qué nos queda por realizar. En el caso del desarrollo de software ocurre exactamente lo mismo. Cuando estamos desarrollando software, el código fuente está cambiando continuamente, esto hace que, en el desarrollo de software actual, sea muy importante que haya sistemas de control de versiones.

Un sistema de control de versiones bien diseñado facilita al equipo de desarrollo su labor, permitiendo que varios desarrolladores trabajen en el mismo proyecto (incluso sobre los mismos archivos) de forma simultánea, sin que se pisen unos a otros. Las herramientas de control de versiones proveen de un sitio central donde almacenar el código fuente de la aplicación, así como el historial de cambios realizados a lo largo de la vida del proyecto. También permite a los desarrolladores volver a una versión estable previa del código fuente si es necesario. En los entornos de desarrollo modernos, los sistemas de control de versiones son una parte fundamental.

Pueden coexistir varias versiones alternativas en un instante dado y hay que disponer de un método, para nombrar las diferentes versiones de manera sistemática y organizada.

Una herramienta de control de versiones es un sistema de mantenimiento de código fuente (grupos de archivos en general) muy útil para grupos de desarrolladores que trabajan cooperativamente, permite al equipo trabajar y modificar concurrentemente ficheros organizados en proyectos. Esto significa que dos o más personas pueden

modificar un mismo fichero sin que se pierdan los trabajos de ninguna. Además, las operaciones más habituales son muy sencillas de usar.

El sistema de control de versiones está formado por un conjunto de componentes:

➔ **Repositorio**: es el lugar de almacenamiento de los datos de los proyectos. Suele ser un directorio en algún ordenador.

➔ Revisión: es cada una de las versiones parciales o cambios en los archivos o repositorio completo. La evolución del sistema se mide en revisiones. Cada cambio se considera incremental.

➔ Etiqueta: información textual que se añada a un conjunto de archivos o a un módulo completo para indicar alguna información importante.

➔ **Rama**: revisiones paralelas de un módulo para efectuar cambios sin tocar la evolución principal. Se suele emplear para pruebas o para mantener los cambios en versiones antiguas.

El **repositorio** es un almacén general de versiones. En la mayoría de las herramientas de control de versiones, suele ser un directorio. Con el repositorio, se va a conseguir un ahorro de espacio de almacenamiento, ya que estamos evitando guardar por duplicado, los elementos que son comunes a varias versiones. El repositorio nos va a facilitar el almacenaje de la información de la evolución del sistema, ya que, aparte de los datos en sí mismo, también almacena información sobre las versiones, temporización, etc.

Cuando usamos un sistema de control de versiones, trabajamos de forma local, sincronizándonos con el repositorio, haciendo los cambios en nuestra copia local y después en repositorio. Ya veremos después como se hace la sincronización.

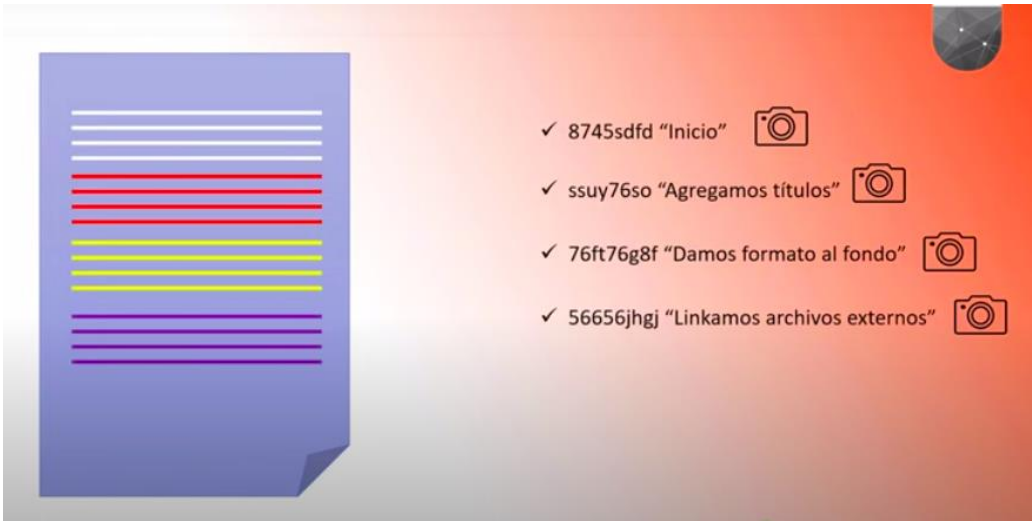
2. GIT

2.1 Introducción

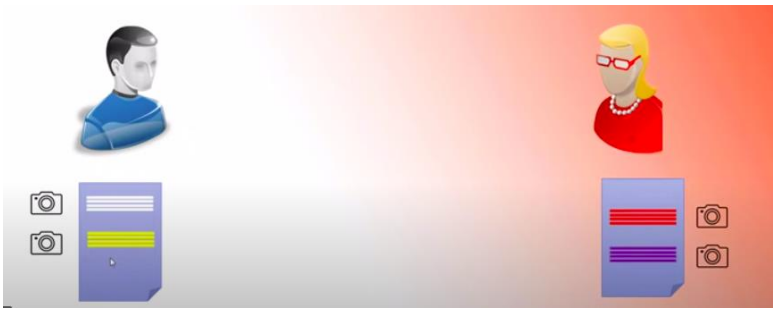
Software desarrollado por Linus Torvalds. Es libre, nos sirve para facilitar el mantenimiento y desarrollo de aplicaciones y el trabajo en equipo. Tiene varias

funcionalidades imprescindibles tal y como concebimos hoy en día el desarrollo de aplicaciones:

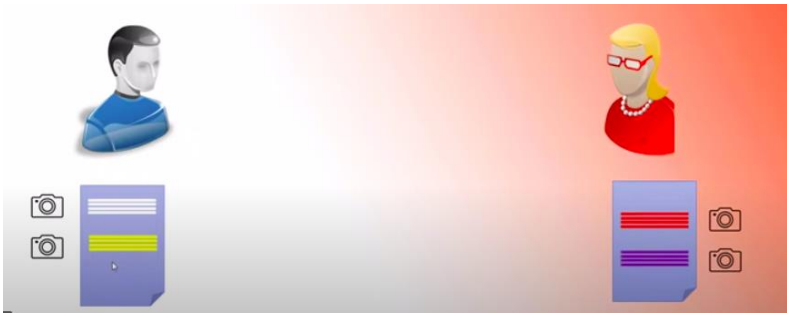
1. Repositorio: la herramienta guarda un registro con los cambios que vamos haciendo, de manera que su principal función es revertir un archivo a un estado concreto del tiempo en el que las cosas iban como tú quieres. Lo realmente importante es que podemos hacer ésto con todos los archivos de código de un proyecto, que estarán implementados por diferentes personas. Hay frameworks que cargan muchos archivos con muchas dependencias entre ellos, sería prácticamente imposible que el programador controlara todas las versiones con todos los cambios que se van haciendo.



2. Las ramas: al desarrollar una aplicación en equipo, tendremos el trabajo repartido y todos los programadores codifican a la vez sobre los mismos archivos. Git divide en ramas o flujos de trabajo lo que cada programador codifica. De esta manera cada integrante del equipo trabaja con su "versión", va implementando su código en su rama y hará las instantáneas que necesite. Cada programador podrá volver la versión que necesite.



Cuando el trabajo está terminado, git debe ocuparse de unir lo que separó. Es decir, con todas las ramas monta de nuevo todos los archivos de la aplicación



3. Cuándo se trabaja en equipo y cada programador codifica un archivo de forma independiente, no habrá conflictos, pero esto no es siempre así. Imaginemos que dos programadores están codificando en el mismo archivo, obviamente es necesario especificar claramente de qué se ocupa cada uno. Aun así, puede suceder que ambos hagan modificaciones sobre la misma parte de código. En este caso, git detecta y avisa de que varias personas están haciendo o han hecho modificaciones sobre el mismo código.

Aunque tiene más funcionalidades, éstas tres son las más importantes.

2.2 Instalación

En el siguiente enlace podemos descargar o consultar comando de instalación para linux

https://git-scm.com/downloads

En nuestra máquina virtual tenemos la versión 2.34 (git –version)
Podemos actualizarla

```
sudo add-apt-repository -y ppa:git-
core/ppa
sudo apt-get update
sudo apt-get install git -y
```

2.3 Repositorios

Las tres tareas básicas con repositorios son:

a Crearlos
b Agregar archivos y directorios
c Respaldar archivos y directorios

Estas operaciones las haremos a través de comandos de consola:

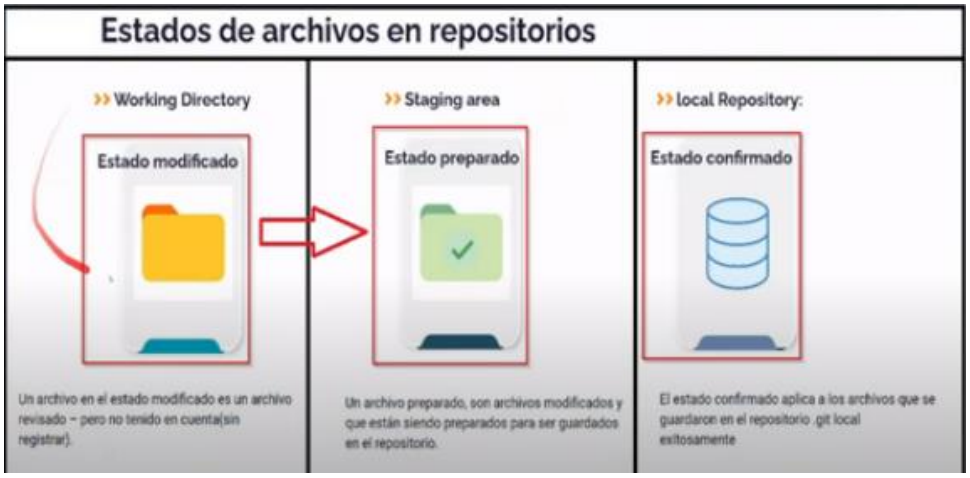
git init: crea dos áreas

- área de ensayo (staging area): es temporal, veremos que archivos tienen seguimiento por parte de git y en qué estado están
- repositorio local: aquí se almacenan las instantáneas que va haciendo git

El repositorio se crea en la ruta en la que ejecutamos el comando. Añade al área de ensayo todo lo que haya en esa ruta, pero no hace seguimiento del contenido hasta que no hayamos ejecutado el siguiente comando.

git add: para decirle a git qué archivos queremos añadir y que se haga seguimiento de ellos. Nosotros seguimos viéndolos en nuestro directorio de trabajo.
Si queremos añadir todos **add .**

git commit: git lleva los archivos al repositorio local



Cada vez que modifico un archivo y quiero llevarlo al repositorio tengo que añadirlo primero al área de ensayo y luego hacerle un commit. Puedo realizar las dos operaciones en un sólo paso con el comando:

git commit -am "descripción"

Para quitar un archivo del staging

git reset: indicamos la rama y el archivo o carpeta que **queremos** sacar del área de ensayo

Ejemplo**: git reset HEAD .metadata**

HEAD es el puntero a la referencia de bifurcación actual, que es, a su vez, un puntero al último commit realizado en esa rama. Eso significa que HEAD será el padre del próximo commit que se cree. En general, es más simple pensar en HEAD como la instantánea de tu último commit.

Otros comandos útiles

git status -s: nos muestra el estado de los archivos o directorios del área de ensayo

git log - -online: nos muestra todas las instantáneas que tenemos en el repositorio, con los códigos que las identifican

Para eliminar cambios

git reset : elimina cambios

En general, es útil para revertir cambios **locales** que nunca se publicaron en un origen compartido. Eliminar un commit de un repository en el que otros desarrolladores web pueden haber seguido desarrollando crea serios problemas para la colaboración.

Los posibles casos de uso pueden ser:

git reset - - hard codigo: restaura a esa instantánea y desecha los demás commits. Elimina los cambios tanto en el staging área como en tu directorio de trabajo. Hemos perdido esos cambios.

Lo utilizamos en el caso de que nos demos cuenta de que a partir de un determinado commit en adelante, y solo si estos commit han permanecido en su clon local, el trabajo se descarta por completo sin dudarlo, lleva el proyecto de vuelta a este "buen" commit, desechando el resto.

git reset - -soft código: no toca los cambios en tu área de trabajo ni en el área de ensayo. Sólo borra el historial de los mismos

Recuperar cambios

git checkout código: restaura esa instantánea el directorio de trabajo, sin eliminar los commits. Aunque los posteriores a la nueva posición de HEAD no los lista, no los he perdido y puedo recuperarlos.

Si hago cambios en mi área de trabajo y no los confirmo, no puedo irme al commit en el que está mi repositorio. O bien confirmo los cambios y luego elimino este commit o bien restauro un commit anterior y después vuelvo a restaurar el último.