

**MP\_0487. Entornos de desarrollo**

**UF4. Control de versiones**

**4.1. Desarrollo en grupos  
de trabajo**

# Índice

---

☰	Objetivos	3
☰	Definición de control de versiones	4
☰	Tipos de control de versiones	6
☰	Repositorios	10
☰	Publicación de cambios «commit»	12
☰	Despliegue o «check-out»	14
☰	Ramas «branching»	16
☰	Otras funciones	18
☰	Resumen	21

# Objetivos

---

Con esta lección perseguimos los siguientes objetivos:

- 1 Comprender la importancia del control de versiones de software.
  - 2 Identificar las principales herramientas para control de versiones.
  - 3 Integrar las pautas para el control de versiones en el desarrollo habitual de un proyecto.
- 

¡Ánimo y adelante!

# Definición de control de versiones

Un sistema de control de versiones (o SCV) es un sistema que realiza copias incrementales de archivos; a cada una de estas copias o registros se le denomina versión.

De esta forma se pueden obtener estados previos del conjunto de archivos correspondientes a una versión anterior.

Estas versiones son generadas a partir de modificaciones realizadas por uno o varios usuarios que trabajan en línea con el mismo conjunto de archivos. El hecho de guardar las copias le va a permitir al usuario:

- 1 Restaurar los archivos o todo un conjunto de ellos de nuevo a un estado anterior.
- 2 Comparar los cambios de la versión actual respecto a otra anterior.
- 3 Encontrar la causa del error de un programa al permitir averiguar qué archivos fueron cambiados y por quién.
- 4 Recuperar archivos que fueron cambiados de manera errónea o perdidos.

En el mercado existen una gran variedad de sistemas de control de versiones que incluyen diversas funcionalidades. No obstante, de manera general, se pueden considerar las siguientes características en un sistema típico:

- **Acceso remoto:** Proporcionar acceso remoto de elementos del repositorio a los usuarios con permisos para modificar, leer, mover, borrar etc.
- **Registro de modificaciones:** Crear un registro de modificaciones efectuadas sobre la estructura de archivos que permita obtener estados previos.
- **Fusionar ramas:** Incorporar un mecanismo que permita fusionar dos versiones de estructuras de archivos diferentes en una sola.
- **Herramienta de software:** Proporcionar un software cliente que podrá ser uno del propio sistema, uno de otro proveedor o bien un navegador que sirva de interfaz del cliente.
- **Control de cambios:** Controlar que los usuarios siempre trabajan sobre la última versión del archivo. Para ello podrán utilizar sistemas de bloqueo o colaborativos, siendo este último el más aconsejado.
- **Notificaciones:** Disponer de un sistema eficaz de notificación al usuario de que un archivo, o grupo de archivos en su conjunto, ha sido guardado de forma satisfactoria y no ha habido conflictos.

# Tipos de control de versiones

---

En su forma más básica, el control de versiones es un mecanismo que permite registrar los cambios de un conjunto de archivos y directorios, conservando cada uno de sus estados o versiones.

Bajo este concepto se podría incluir **un sistema manual**, en el que cada vez que un archivo es actualizado se guarde con distinto nombre en un directorio en el que se indique la fecha de actualización. Este modo manual puede llevar con facilidad a errores en el guardado con pérdida.

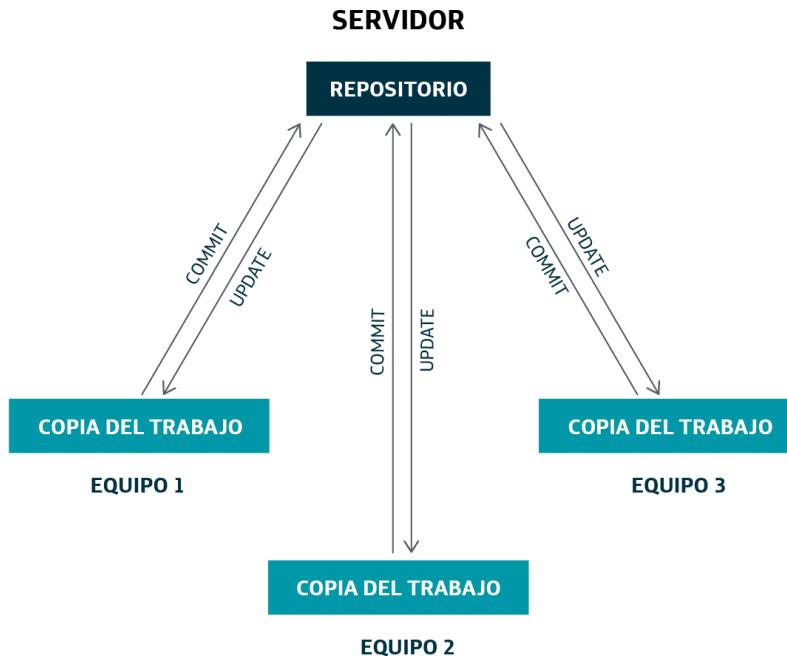
Para evitar este problema se crearon los primeros **programas de control de versiones** de forma local, como el **RCS (sistema de revisión de control)** escrito por Walter F. Tichy y disponible para Mac OS X y compuesto por una BBDD y un sistema de bloqueo de archivos a nivel local.

Posteriormente se introdujeron los sistemas actuales, que permiten la gestión remota, siendo los más habituales los de tipo **distribuido y centralizados**, como veremos a continuación.

## Sistemas centralizados

Los sistemas centralizados son aquellos en los que existe un **servidor que contiene el repositorio completo** y varios clientes que acceden a dicho repositorio para realizar una copia de trabajo de una versión sobre la que trabajar.

La secuencia de trabajo habitual desde un sistema centralizado será de la siguiente forma:



1

Un usuario, por medio de una orden “*update*”, obtiene una copia de trabajo en su equipo de la versión requerida, procedente del repositorio del servidor.

2

El usuario realiza los cambios oportunos en la copia de forma local y, una vez finalizado, por medio de una orden “*commit*” envía los cambios realizados en la versión al servidor.

3

Un nuevo usuario **solicita una copia de la misma versión del servidor**, que ya vendrá con los cambios realizados por el anterior usuario, y vuelve a realizar el mismo proceso. De esta forma, si uno de los usuarios realiza cambios en el sistema de archivos, dichos cambios podrán ser visualizados por el resto de usuarios.

A continuación se enuncian las principales ventajas y desventajas de los sistemas de control de versiones centralizados.

## Ventajas

- El repositorio se gestiona de manera más sencilla.

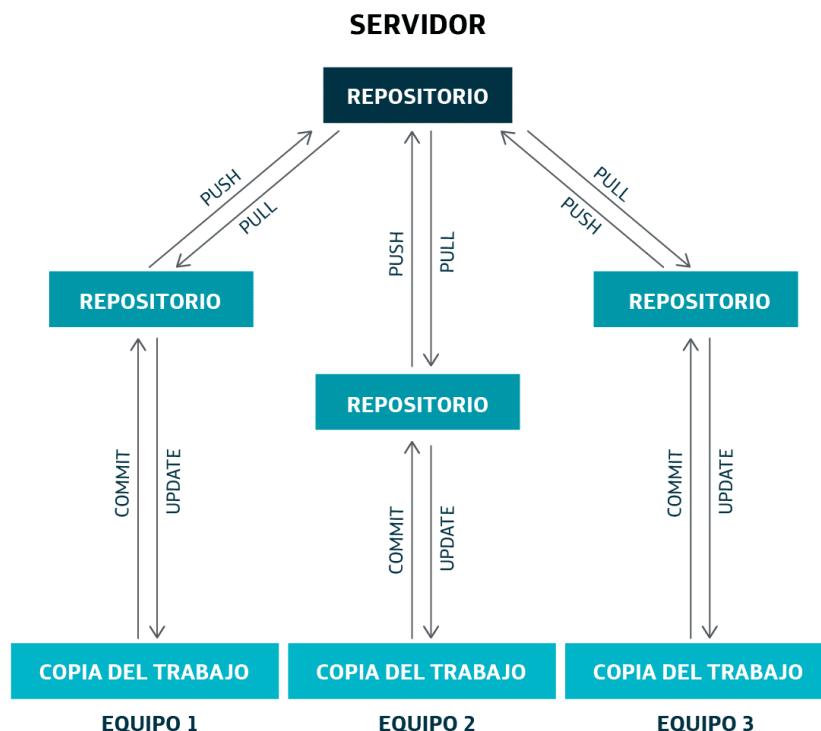
## Desventajas

- Ante una caída del servidor central no será posible respaldar los datos.
- Las copias del repositorio central deben ser más frecuentes al contar con un único repositorio.

## Sistemas distribuidos

En este tipo de sistema de control de versiones los usuarios disponen en sus equipos locales de una copia del repositorio completo, por lo que en su manera normal de trabajo no lo harán sobre el repositorio central, sino sobre el de copia de su equipo. Cada máquina contiene en todo momento una copia completa del repositorio.

El funcionamiento podría describirse de la siguiente manera.



- 1 Un usuario accede por primera vez al sistema y, por medio de la orden “*pull*”, descarga el repositorio completo desde el servidor hasta su equipo local.
- 2 El usuario, por medio de la orden “*update*”, obtiene una copia de trabajo procedente del repositorio de su equipo.
- 3 El usuario realiza los cambios oportunos en la copia de trabajo y, una vez finalizado, por medio de una orden “*commit*” envía los cambios realizados al repositorio local de su equipo.
- 4 Si desea que los cambios de su repositorio local se hagan efectivos en el servidor deberá hacerlo por medio de la orden “*push*”.

A continuación se enuncian las principales ventajas y desventajas de los sistemas de control de versiones distribuidos.

#### VENTAJAS

- Ante una pérdida del servicio del servidor se puede recuperar el repositorio íntegro desde cualquiera de los equipos.
- Las operaciones normales de trabajo, en las que no se requiere actualizar los repositorios, se realizan de forma más rápida y con menor tráfico de datos por la red.
- Permite el trabajo privado de versiones en modo “borrador”, en las que no se quiera compartir cambios con el resto de usuarios.

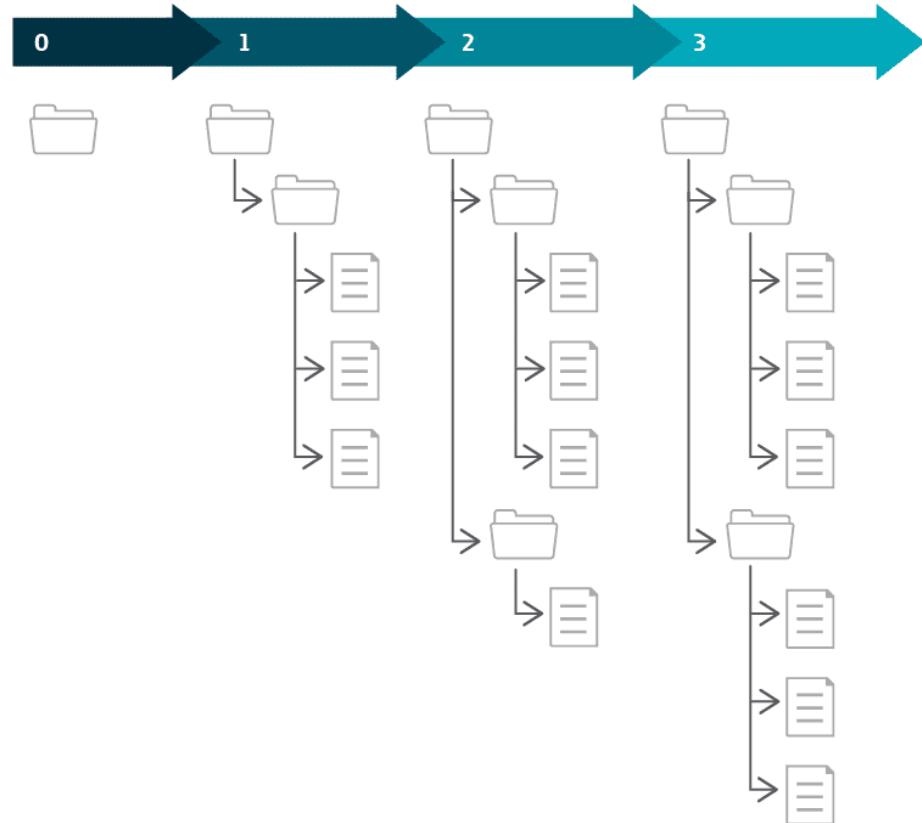
#### DESVENTAJAS

- Descargar la copia inicial del repositorio es un proceso que puede llevar tiempo.
- Cuando se propagan los cambios del repositorio de un usuario se obtienen elevados tiempos de actualización y mayor tráfico de datos.
- Se requiere más espacio en disco duro en los equipos de los usuarios.
- Al trabajar con repositorios locales las actualizaciones hacia el resto de usuarios suelen realizarse menos frecuentemente.

# Repositorios

Los repositorios se depositan en **almacenes de datos**, que se componen tanto de los archivos reales como de registros de las transacciones realizadas.

Se define un repositorio como una secuencia de estados de las estructuras de archivos y directorios que se almacenan a lo largo del tiempo.



En la imagen anterior vemos el proceso típico que se realiza en el control de versiones, donde podemos comprobar que:

- Inicialmente la estructura de archivos se compone de una única carpeta y es la primera que es guardada por el usuario.
- En la siguiente versión guardada el sistema de archivos se compone del directorio del estado o, que en este caso contiene otro directorio con 3 archivos.
- Sucesivamente se van guardando de forma incremental cada una de las estructuras en versiones, a las que el usuario puede acceder cuando lo requiera.



Cada vez que un usuario genera cambios en una versión quedan almacenadas una serie de propiedades asociadas a dicho cambio, como pueden ser el nombre del autor que realiza el cambio, la fecha en que se ha realizado o los comentarios que el autor del cambio desee incorporar.

## Publicación de cambios «commit»

---

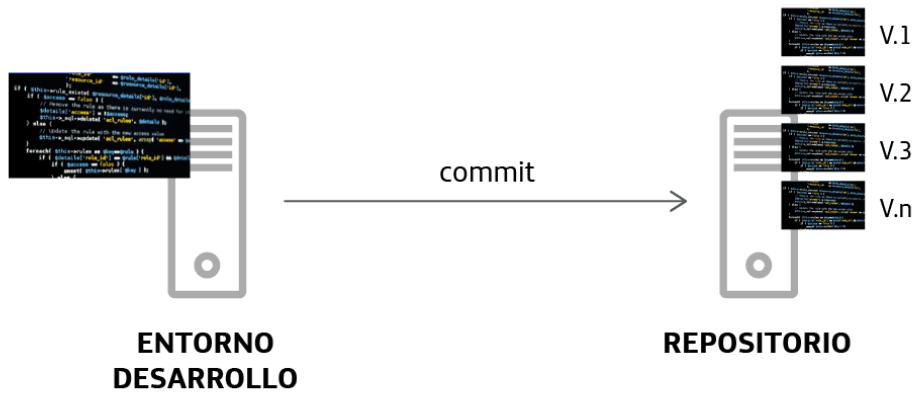
Una copia de trabajo se compone **de una estructura de árbol con archivos que pueden ser editados** por el usuario de forma local, sin intercambiar ningún tipo de cambios con el servidor hasta que el usuario lo estime oportuno.

Cuando un usuario desea trabajar sobre una estructura de archivos determinada o versión no lo hace directamente sobre los archivos guardados en el servidor. En lugar de esto lo que solicita al servidor es una copia de una versión, que es descargada en el equipo del usuario y que se denomina copia de trabajo.

Una vez que el usuario ha realizado los cambios deseados en su copia de trabajo y comprueba que funciona de forma correcta requerirá dejar almacenados dichos cambios en el servidor.

Este proceso de envío de los cambios realizados en la copia de trabajo hacia el servidor para que queden guardados y puedan ser utilizados por otros usuarios se denomina “*check-in*” o “*commit*”.

Esta operación provocará que el sistema cree una nueva versión (cuyo número será el que existía en la estructura del repositorio +1) de la estructura de archivos y que se genere un nuevo registro de la BBDD con los datos del cambio introducido.



Una operación “**commit**” permite incorporar los cambios que el usuario ha realizado en su copia de trabajo al servidor.

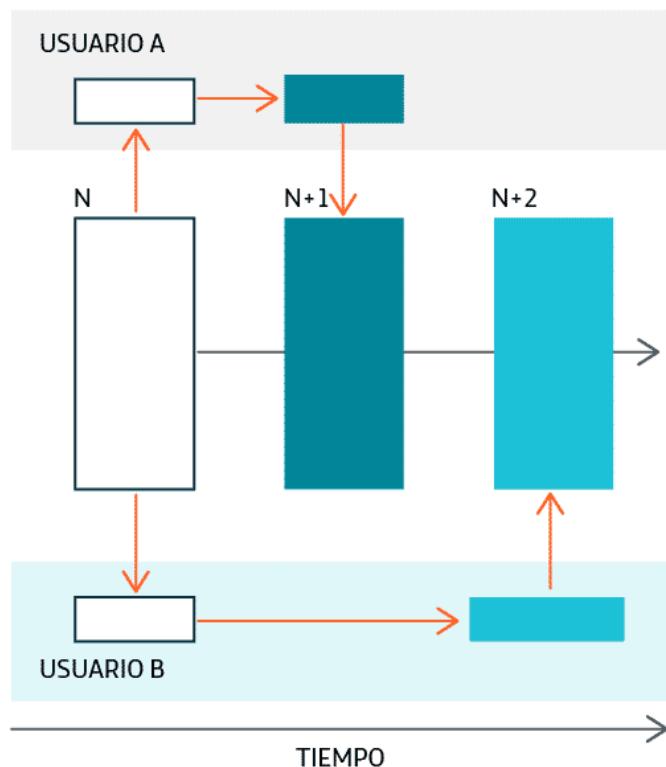
---

Cuando un usuario realiza una operación de **commit** el sistema suele permitir adjuntar un comentario descriptivo del cambio realizado, que quedará guardado en el registro de cambios y que será de ayuda para futuras revisiones.

## Despliegue o «check-out»

Se denomina **despliegue** o **check-out** al proceso en el que un usuario realiza una copia de trabajo local en su equipo a partir de una versión procedente del repositorio.

Cuando trabajamos en equipo puede darse la situación de que dos desarrolladores suban cambios procedentes del mismo repositorio, lo que provoca que existan dos versiones diferentes con parte del código definitivo en las dos últimas versiones.



Como vemos en la figura, la situación final es que los cambios realizados por el **usuario A** se encuentran solo contemplados en la **versión N+1** y no se verán reflejados en la última. Por tanto, cualquier otro usuario que acceda al repositorio no verá los cambios realizados por A.

---

**Para evitar el problema** de pérdidas de versiones el sistema puede optar por adoptar alguna de las siguientes soluciones:

#### **Check-out exclusivo**

De tal forma que una vez que el usuario A realiza su despliegue bloquea la versión que es descargada, y por tanto se impide que cualquier otro usuario haga cambios en la versión. El bloqueo de la versión durará hasta que el usuario A haya finalizado los cambios y realice un “commit”.

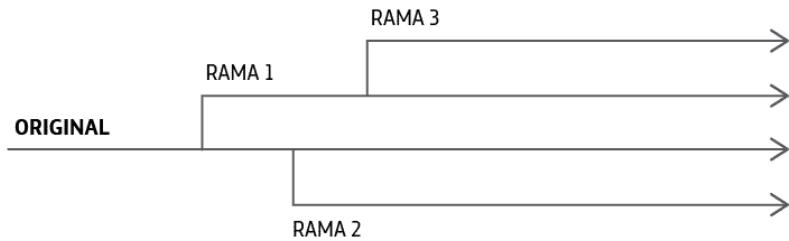
#### **Check-out colaborativo**

En el que el usuario hará un despliegue de una versión sin bloquearla, por lo que permitirá que cualquier otro usuario realice cambios en la versión, que serán combinados.

## Ramas «branching»

Existen situaciones dentro del ciclo de vida de un proyecto en las que se requiere la realización de una versión en paralelo. Es decir, a partir de una versión común, evolucionarla por dos vías o “**ramas**” (*branches*) distintas.

Las modificaciones realizadas en los entornos paralelos podrán ser fusionadas posteriormente en un proceso denominado **fusión de cambios** o “*merge*”. La bifurcación de una evolución puede deberse a:



1 Necesidad de evolucionar un software.

2 Necesidad de realizar numerosas modificaciones.

3 Necesidad de crear aplicativos diferentes.

- i** Estas operaciones en un sistema de control de versiones como SVN supone la realización de una copia de una versión en otra para que cada una siga por su parte.

## Fusiones "merging"

Una **fusión** o “*merging*” se produce cuando dos ficheros o conjuntos de ficheros son combinados para formar uno solo. Esto puede suceder por dos motivos:

### Un usuario trata de actualizar cambios

Bien procedentes de los usuarios que trabajan sobre una misma versión en un sistema por medio de “*update*”, o bien cuando son enviados al repositorio por medio de “*commit*” en un sistema de despliegue colaborativo, tal y como se vio en secciones anteriores.

### El usuario ejecuta la orden “merging”

Por el que una línea de desarrollo sincroniza su contenido con otra línea distinta de desarrollo.

Este es un proceso típico de la fusión de una rama hija cuando es reintegrada en la rama principal, dando lugar a una nueva revisión.

En ambos casos se deberán resolver de forma correcta los cambios que entren en conflicto.

---

Aunque el sistema realice una combinación de forma automática, una fusión siempre deberá ser verificada de forma manual. El hecho de que el programa haya sido capaz de añadir líneas o eliminar archivos sin que se detecte conflicto, no es sinónimo de que el código vaya a poder ejecutarse de manera correcta.

## Otras funciones

---

En los puntos anteriores hemos visto las principales acciones que se realizan en la gestión del control de versiones, pero existen otras a tener en cuenta. A continuación exponemos algunas de ellas.

### **Etiquetado («tagging»)**

Una etiqueta es un estado o versión de una estructura de archivos determinada, a la que se denomina con un nombre identificativo.

Uno de los motivos de poner una etiqueta o marca a una versión es la de poder identificar un hito o punto importante en el desarrollo del proyecto, como puede ser la finalización de una versión o una entrega del producto al cliente.

En este caso se haría necesario mantener inalterada y localizada la versión a partir de la cual se obtuvo la versión de producción, por si fuera necesario realizar algún cambio debido a errores detectados en la entrega.

Para la mayoría de los sistemas de control de versiones, una etiqueta es tratada como la creación de una rama a partir de la copia de una versión principal del desarrollo y es alojada en un directorio donde no se evolucionan nuevas versiones.



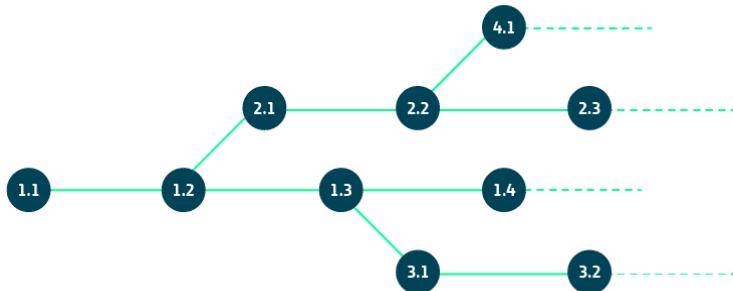
## Líneas de base («*baseline*»)

Una **línea de base** o “*baseline*” la componen aquellas secuencias de versiones del desarrollo que han sido validadas y que serán utilizadas como base para las futuras modificaciones.

Es esta línea base sobre la que se crearán las distintas ramas requeridas y sobre la que se deberán fusionar posteriormente en los casos que se considere necesario.

También será necesario trabajar sobre esta línea base cuando se detecte que ha habido un problema con la aplicación y se deseé hacer una revisión de su histórico.

A partir de esta línea también serán generadas las distintas versiones etiquetadas.



## Actualizaciones «*update*» o «*sync*»

Como ya se ha comentado, cuando un usuario trabaja con un sistema de control de versiones en realidad lo está haciendo sobre una copia de trabajo que se descargó inicialmente desde el repositorio.

Esto significa que si se trabaja de forma colaborativa con otros usuarios puede suceder que algunos cambios recientes no estén contemplados en dicha copia.

Para ello se incorpora un comando de “*update*” o “*sync*”, cuya función es la de comparar la actual copia de trabajo con la de la última versión del repositorio, de tal forma que los posibles cambios puedan ser actualizados en la copia de trabajo del usuario.

## Congelaciones

En los procesos de desarrollo pueden darse circunstancias en las que se desee que el producto no sufra nuevas modificaciones y quede “congelado” en la versión en la que se encuentra.

Una de las soluciones consistiría en situar una copia en una rama especial, tal y como se realiza con las versiones. Sin embargo, para asegurar que ningún usuario, por desconocimiento o error, pueda llegar a añadir nuevos cambios se puede:

### Deshabilitar la funcionalidad de “commit”

Por medio de las herramientas administrativas y solo para la rama considerada. En este caso se permite el acceso en modo de “solo lectura”.

### Deshabilitar permisos de despliegue de la versión

Por lo que solo los usuarios con autorización podrán acceder.

## Resumen

---

Has finalizado esta lección.

En esta lección hemos aprendido la importancia que tiene en nuestra profesión el control de versiones para poder garantizar la evolución de nuestros desarrollos en el tiempo, pudiendo recuperar versiones anteriores y así, mediante la comparación, solucionar posibles problemas.

Existen diferentes formas de estructurar los componentes de los sistemas de control de versiones, ya sea centralizados o distribuidos, pero todos tienen la misma filosofía de trabajo, procediendo a guardar el estado en el que se encuentra el código en un determinado momento, definiendo así una versión del mismo.

Estas versiones son generadas a partir de modificaciones realizadas por uno o varios usuarios que trabajan en línea con el mismo conjunto de archivos.



**PROEDUCA**