

DESPLIEGUE DE APLICACIONES WEB
TÉCNICO EN DESARROLLO DE APLICACIONES WEB

Control de versiones

ÍNDICE

/ 1. Introducción y contextualización práctica	3
/ 2. El control de versiones	4
/ 3. Clasificación	4
/ 4. Herramientas control de versiones	6
/ 5. Caso práctico 1: “Issues en GitHub/GitLab”	7
/ 6. GitLab	7
6.1. Operaciones avanzadas	9
6.2. Securizando GitLab	10
6.3. GitLab LDAP	11
/ 7. Caso práctico 2: “Acceso a GitLab LDAP”	13
/ 8. Resumen y resolución del caso práctico de la unidad	14
/ 9. Bibliografía	14
/ 10. Webgrafía	14

OBJETIVOS

Descubrir qué son y para qué sirven los sistemas de control de versiones.

Conocer cómo funcionan.

Aprender a instalar sistemas de control de versiones.

Securizar un servidor de control de versiones.

/ 1. Introducción y contextualización práctica

En esta unidad, estudiaremos los sistemas de control de versiones. Analizaremos para qué sirven, qué tipos hay, y qué diferencias existen entre ellos. Conoceremos qué *software* podemos utilizar, y aprenderemos a instalar y configurar un servidor de control de versiones.

Por otro lado, explicaremos las operaciones que podemos realizar en los sistemas de control de versiones y qué medidas de seguridad debemos tomar para proteger el servidor.

Planteamiento del caso práctico inicial

A continuación, vamos a plantear un caso a través del cual podremos aproximarnos de forma práctica a la teoría de este tema.

Escucha el siguiente audio donde planteamos la contextualización práctica de este tema, encontrarás su resolución en el apartado Resumen y resolución del caso práctico.

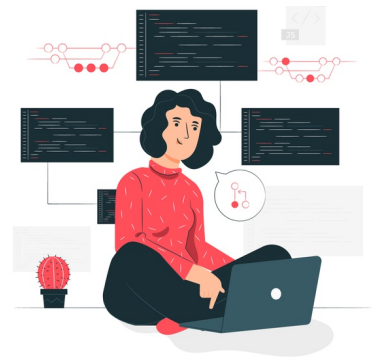


Fig. 1. Control de versiones.



Audio intro. "Caso práctico inicial"

<https://bit.ly/37wdexr>





/ 2. El control de versiones

En el proceso de desarrollo de una aplicación o de cualquier otro tipo de *software*, es habitual realizar cambios para corregir errores o añadir nuevas funcionalidades. Puede ocurrir que alguna de estas modificaciones provoque otros fallos y necesitemos volver al punto de partida.

Antes de realizar cambios, es necesario realizar una copia de seguridad de los ficheros de configuración de un servicio (por ejemplo, Apache). Este procedimiento se considera una **buena práctica** y, además, es una forma de establecer un **control de versiones manual**. Este sistema no es demasiado funcional, ya que, realmente, no tendremos un control sobre los cambios que hemos realizado.



Fig. 2. Backup de archivos PHP.

Por este y otros motivos aparecen los sistemas de control de versiones, que permiten detectar qué cambios hemos realizado en un archivo, generar un registro y tener una copia del proyecto a buen recaudo para evitar perder trabajo.

Además, los sistemas de control de versiones pueden establecer un modo de funcionamiento basado en la arquitectura cliente-servidor que permitirá que los componentes de un equipo de desarrollo puedan acceder a los archivos, minimizando el riesgo de conflictos.

Las características generales de un sistema de control de versiones son las siguientes:

- Deberá implementar los mecanismos necesarios para poder **almacenar y gestionar** los proyectos de manera segura.
- Deberá ser capaz de **realizar modificaciones** en los proyectos que almacene, disponiendo de las herramientas adecuadas.
- Deberá generar un **registro de eventos**, identificando qué modificaciones se han realizado en cada elemento. Además, se podrá hacer **rollback al estado original** si fuese necesario.

Teniendo en cuenta estos aspectos, será necesario incluir un sistema que permita generar informes de las modificaciones realizadas en cada archivo. Estos informes servirán para obtener una visión general del estado de cada elemento o del proyecto.



Audio 1. "El primer software de control de versiones"

<https://bit.ly/39w1MEB>



/ 3. Clasificación

Como sucede con cualquier otro *software*, dependiendo de la arquitectura que utilice, podemos diferenciar dos tipos de control de versiones: centralizados y distribuidos.

Sistemas de control de versiones centralizados

Los sistemas de control de versiones centralizados o **VCS** (*Version Control Systems*) utilizan una arquitectura cliente-servidor para gestionar los cambios.



Los clientes se conectarán al servidor para **descargar una copia** del proyecto de manera periódica. El servidor actuará de **repositorio maestro** y almacenará la información que los clientes envíen.

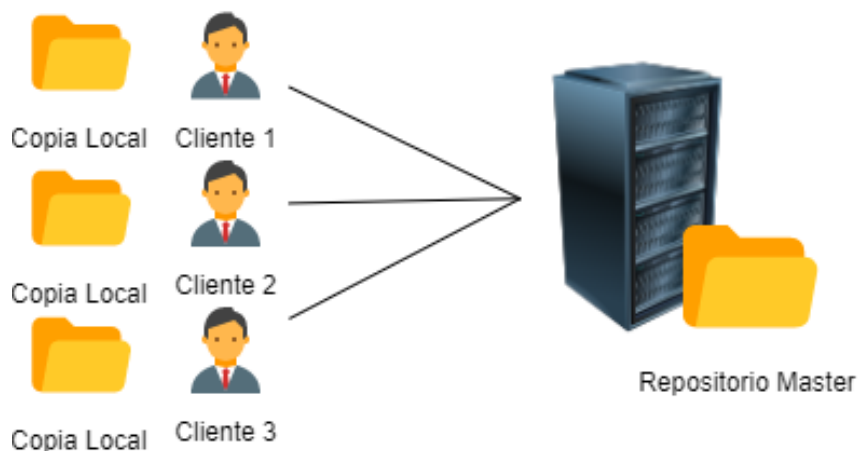


Fig. 3. Control de versiones centralizado.

La información original se alojará en el servidor y los clientes descargarán una copia en local. Las modificaciones se realizarán sobre esta copia, que se sincronizará con el repositorio principal actualizando los cambios.

Sistemas de control de versiones distribuidos

Los sistemas de control de versiones distribuidos o **DVCS** (*Distributed Version Control Systems*) utilizan una topología de red en malla, de modo que cada cliente conectado a la red tendrá su propio repositorio. El funcionamiento es similar a las redes P2P y, de la misma manera, la red será más eficiente con el aumento del número de hosts.

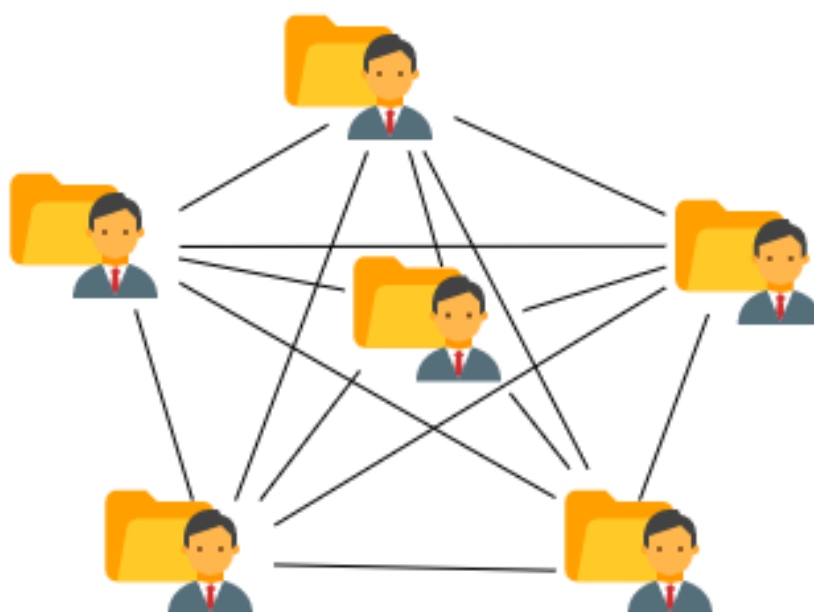


Fig. .4 Control de versiones distribuido.

El mayor problema de este modelo es la sincronización entre los diferentes repositorios, ya que una modificación en cualquiera de ellos se replicará en el resto de nodos. Este proceso de sincronización podría dar lugar a problemas de redundancia de datos o, incluso, provocar que se pierda información.




/ 4. Herramientas control de versiones

Actualmente, podemos encontrar una gran variedad de herramientas que realizan funciones de control de versiones, tanto de *software* libre como con licencia comercial.

A continuación, expondremos algunos de los diferentes sistemas de control de versiones que podemos utilizar, diferenciando entre el tipo de arquitectura y el tipo de licencia que utilizan:


	CVS	DCVS	OPEN SOURCE	LICENCIA COMERCIAL
CVS	x		x	
Subversion	x		x	
Mercurial		x	x	
Monotone		x	x	
Perforce	x			x
Git		x	x	
AutoDesk Vault	x			x

Tabla 1. Comparativa de sistemas de control de versiones.



Vídeo 1. "Creación de repositorio en GitHub"

<https://bit.ly/2VrmfSB>



De entre todos los sistemas de control de versiones que hemos nombrado, vamos a prestar especial atención a Git.



Fig. 5 Logo de Git.

Git es un sistema distribuido de control de versiones creado por **Linus Torvalds** en el año 2005 para poder gestionar el desarrollo del Kernel de Linux. Como no podía ser de otra manera, Git es un proyecto de *software* libre distribuido con licencia GNU.

De Git han surgido otros proyectos, como GitHub o GitLab.

GitHub es un sistema de control de versiones distribuido escrito en *Ruby on Rails* que apareció en 2008. Actualmente, GitHub es la mayor comunidad de desarrolladores, superando los 40 millones de usuarios.

GitHub solo permite crear repositorios en su infraestructura. Por este motivo, si queremos alojar nuestro propio servidor de control de versiones, deberemos instalar **GitLab**.

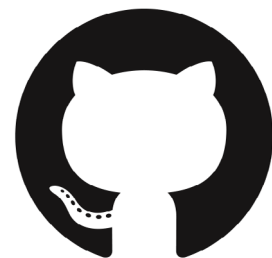


Fig. 6. Logo GitHub.



/ 5. Caso práctico 1: “Issues en GitHub/GitLab”

Planteamiento: Isabel está instalando un módulo en una aplicación web que ha descargado de GitHub, pero está teniendo problemas para instalarlo. Ha revisado el repositorio y no ha encontrado ningún error relacionado con su problema.

Nudo: ¿Cómo podrá ponerse en contacto con el desarrollador del módulo?

Desenlace: Isabel puede hacer una consulta pública en el repositorio del desarrollador del módulo utilizando el apartado **issues** de GitHub.

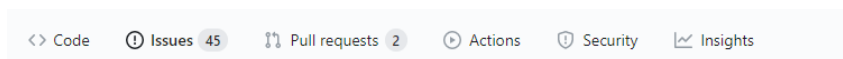


Fig. 7 Pestaña issues en GitHub.

Desde esta pestaña, Isabel podrá crear una consulta y, tanto el desarrollador como cualquier otra persona de la comunidad GitHub, podrá responderle y ayudarla a resolver el problema.

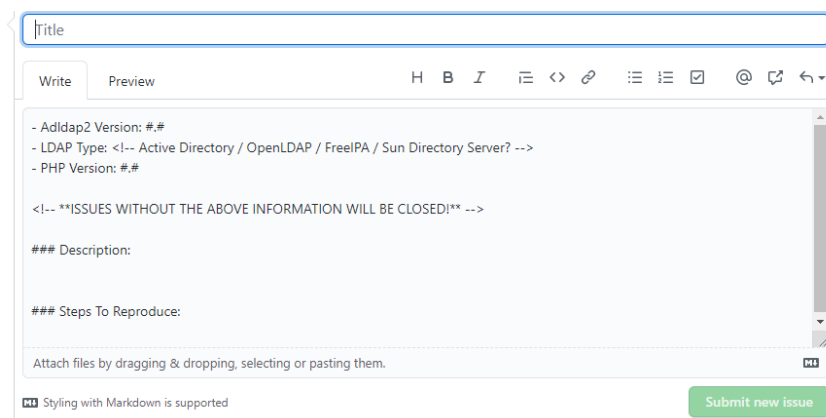


Fig. 8. New Issue en GitHub.

/ 6. GitLab

GitLab es un *software* basado en Git que aparece en 2014 y actúa como **sistema de control de versiones**. Además, GitLab permite **crear wikis** para realizar documentación complementaria.

Veremos cómo es la instalación de GitLab sobre Ubuntu 18.04, comenzando por actualizar los repositorios de Ubuntu. A continuación, instalaremos las dependencias necesarias:

```
sudo apt-get update && sudo apt-get upgrade
sudo apt-get install -y curl openssh-server ca-certificates tzdata
```

Código 1. Instalación de GitLab. <https://about.gitlab.com/install/?version=ce#ubuntu>

El siguiente paso será instalar postfix, un servidor de correo para poder enviar notificaciones. Durante la instalación, seleccionaremos la opción **“servidor local”**:

```
sudo apt-get install -y postfix
```

Código 2. Instalación de GitLab. <https://about.gitlab.com/install/?version=ce#ubuntu>



Seguidamente, descargaremos un script desde el repositorio de GitLab, que preparará el sistema para la instalación. Una vez terminado, instalaremos GitLab utilizando `apt`.

```
curl -sS https://packages.gitlab.com/install/repositories/gitlab/gitlab-ce/script.deb.sh | sudo bash
sudo apt-get install gitlab-ce
```

Código 3. Instalación de GitLab. <https://about.gitlab.com/install/?version=ce#ubuntu>

Para finalizar, editaremos el fichero de configuración de GitLab, **gitlab.rb**, y buscaremos la línea “**external_url**”. En este campo, introduciremos la IP del servidor. Posteriormente, ejecutaremos el comando **gitlab-ctl reconfigure** para aplicar los cambios.

```
sudo nano /etc/gitlab/gitlab.rb
## GitLab URL
external_url 'http://192.168.2.198'
sudo gitlab-ctl reconfigure
```

Código 4. Instalación de GitLab. <https://about.gitlab.com/install/?version=ce#ubuntu>

En este punto, ya podremos acceder con un navegador web al servidor GitLab, y lo primero que nos pedirá será introducir la contraseña que utilizará el usuario **root**. Al iniciar sesión, accederemos a la página principal, donde encontraremos un botón que ejecutará el asistente para crear nuevos proyectos.

Fig. 9. Creación de proyectos utilizando el asistente.

Tras la creación del repositorio, podemos realizar operaciones como clonar, crear un nuevo archivo o añadir una wiki.

The repository for this project is empty
You can get started by cloning the repository or start adding files to it with one of the following options.

[Clone](#) [New file](#) [Add README](#) [Add LICENSE](#) [Add CHANGELOG](#) [Add CONTRIBUTING](#)

Fig. 10. Desde la interfaz web podemos realizar diversas operaciones sobre el repositorio.



Vídeo 2. “Creación de proyectos y wiki”
<https://bit.ly/3ojhCGu>





6.1. Operaciones avanzadas

A continuación, comentaremos algunas operaciones que podemos ejecutar desde la línea de comandos para trabajar con GitLab:

- **Operaciones sobre el servicio:**

Comando	Descripción
gitlab-ctl [start stop restart]	Inicia o para el servicio GitLab
gitlab-ctl status	Muestra el estado de los servicios
gitlab-ctl check-config	Realiza un chequeo de la configuración
gitlab-ctl reconfigure	Reconfigura GitLab. Es necesario ejecutar este comando después de aplicar cambios en el fichero de configuración
gitlab-ctl renew-le-certs	Conectará con Let's Encrypt y renovará el certificado existente

Tabla 2. Operaciones sobre el servicio

- **Operaciones sobre repositorios:**

Crear una copia local de un repositorio en GitLab:

```
git clone http://192.168.2.198/gitlab/repo1.git
Cloning into 'repo1'...
Username for 'http://192.168.2.198': gitlab
Password for 'http://gitlab@192.168.2.198':
remote: Enumerating objects: 6, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 6 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (6/6), done.
usuario@gitlab:~$ cd repo1/
usuario@gitlab:~/repo1$ ls
2 README.md
```

Código 5. GitLab.



Subir ficheros a un repositorio:

```
usuario@gitlab:~$ git clone http://192.168.2.198/gitlab/repo1.git
Cloning into 'repo1'...
Username for 'http://192.168.2.198': gitlab
Password for 'http://gitlab@192.168.2.198':
remote: Enumerating objects: 6, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 6 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (6/6), done.
usuario@gitlab:~$ cd repo1/
usuario@gitlab:~/repo1$ ls
2 README.md
usuario@gitlab:~/repo1$ git clone http://192.168.2.198/gitlab/repo2.git
Cloning into 'repo2'...
Username for 'http://192.168.2.198': gitlab
Password for 'http://gitlab@192.168.2.198':
warning: You appear to have cloned an empty repository.
usuario@gitlab:~/repo1$ cd repo2/
usuario@gitlab:~/repo1/repo2$ touch README.md
usuario@gitlab:~/repo1/repo2$ git add README.md
usuario@gitlab:~/repo1/repo2$ git commit -m "añadir README"
[master (root-commit) 87558f3] añadir README
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 README.md
usuario@gitlab:~/repo1/repo2$ git push -u origin master
Username for 'http://192.168.2.198': gitlab
Password for 'http://gitlab@192.168.2.198':
Counting objects: 3, done.
Writing objects: 100% (3/3), 219 bytes | 219.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To http://192.168.2.198/gitlab/repo2.git
 * [new branch]    master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
```

Código 6 Instalación de GitLab - <https://about.gitlab.com/install/?version=ce#ubuntu>

Podremos encontrar más información en la web oficial de GitLab:



Enlaces de interés...

https://docs.gitlab.com/ee/administration/operations/moving_repositories.html

6.2. Securizando GitLab

Uno de los motivos que puede llevar a instalar un servidor GitLab privado es asegurar que la información que almacenamos no sea accesible a terceros. Para ello, un aspecto crítico en la configuración del servidor GitLab es la seguridad, especialmente si dicho servidor estuviese expuesto a internet.



Siguiendo una serie de recomendaciones, conseguiremos proteger el repositorio de ataques o accesos indebidos como, por ejemplo:

- **Limitar acceso por IP:** una acción aparentemente sencilla puede ser configurar el firewall del propio sistema operativo para autorizar direcciones IP concretas.
- **Seguridad en las contraseñas:** utilizar contraseñas robustas nos evitará que puedan ser hackeadas mediante técnicas de fuerza bruta.

How secure is your password?

Tip: Try to make your passwords at least 15 characters long Show password: ☒

passwd

Very Weak

6 characters containing: ✓ Lower case ✗ Upper case ✗ Numbers ✗ Symbols

<p>Time to crack your password:</p> <p>0.15 seconds</p>	<p>Review: Oh dear, using that password is like leaving your front door wide open. Your password is very weak because it is a common password.</p>
--	---

Your passwords are never stored. Even if they were, we have no idea who you are!

Fig. 11. Password strength test. <https://www.my1login.com/resources/password-strength-test/>

- **Niveles de servicio:** en GitLab, podemos definir niveles de servicio o roles por usuario. Evitar el uso de usuarios administradores y gestionar los diferentes perfiles es una buena práctica de seguridad.

Access level ☒ Regular

Regular users have access to their groups and projects

☐ Admin

Administrators have access to all groups, projects and users and can manage all features in this installation

Fig. 12. Niveles de servicio.

- **Desactivar la opción de registros públicos:** esta opción parece evidente y, no obstante, puede suponer una gran brecha de seguridad en caso de estar activada.

Sign-up restrictions

Configure the way a user creates a new account.

☐ Sign-up enabled

When enabled, any user visiting http://192.168.2.198/users/sign_in will be able to create an account.

Fig. 13. Desactivación de registro público.

- **Forzar el uso de HTTPS:** enviar el tráfico cifrado mejora la seguridad de la conexión.

6.3. GitLab LDAP

El uso de un servidor LDAP es una manera muy práctica de gestionar los accesos a cualquier aplicación, en este caso, a un servidor GitLab. De esta forma, podremos crear un grupo en el directorio activo y permitir que inicien sesión, únicamente, unos determinados usuarios.



El proceso de configuración del **bind LDAP** debe hacerse modificando el fichero **gitlab.rb**. Una vez localicemos el bloque que contiene las líneas para configurar LDAP, comenzaremos la configuración quitando comentarios según proceda, y cumplimentando los parámetros de acuerdo a los requisitos de nuestro servidor.

```
sudo nano /etc/gitlab/gitlab.rb
gitlab_rails['ldap_enabled'] = true
# gitlab_rails['prevent_ldap_sign_in'] = false
#### **remember to close this block with 'EOS' below**
gitlab_rails['ldap_servers'] = YAML.load <<-'EOS'
main: # 'main' is the GitLab 'provider ID' of this LDAP server
  label: 'ldap'
  host: '192.168.2.3'
  port: 389
  uid: 'sAMAccountName'
  bind_dn: 'CN=gitlab,CN=Users,DC=ad,DC=empresa,DC=com'
  password: '*****'
  encryption: 'plain' # "start_tls" or "simple_tls" or "plain"
  verify_certificates: false
  smartcard_auth: false
  active_directory: true
  allow_username_or_email_login: false
  lowercase_usernames: false
  block_auto_created_users: false
  base: 'CN=Users,DC=ad,DC=empresa,DC=com'
# user_filter: ""
EOS
```

Código 7. Configuración autenticación de LDAP modificando el fichero `/etc/gitlab/gitlab.rb`. <https://docs.gitlab.com/ee/administration/auth/ldap/>

Debemos prestar especial atención a terminar el bloque de configuración con **EOS** y a respetar los espacios, así como a no tabular, ya que esto provocaría que no se procese correctamente la configuración.

A continuación, reconfiguraremos GitLab y comprobaremos que el servidor de control de versiones es capaz de comunicar con el servidor LDAP, y obtener el listado de usuarios que podrán iniciar sesión con el comando **gitlab-rake**:

```
sudo gitlab-ctl reconfigure
sudo gitlab-rake gitlab:ldap:check
Checking LDAP ...

LDAP: ... Server: ldapmain
LDAP authentication... Success
LDAP users with access to your GitLab server (only
showing the first 100 results)
  DN: cn=grupo de replicación de contraseña rodca
  permitida,cn=users,dc=ad,dc=empresa,dc=com
```

Código 8. Configuración autenticación de LDAP en GitLab.



Por último, podremos acceder a la interfaz web de GitLab y comprobaremos que nos aparece una nueva pestaña que permitirá iniciar sesión con usuarios LDAP.

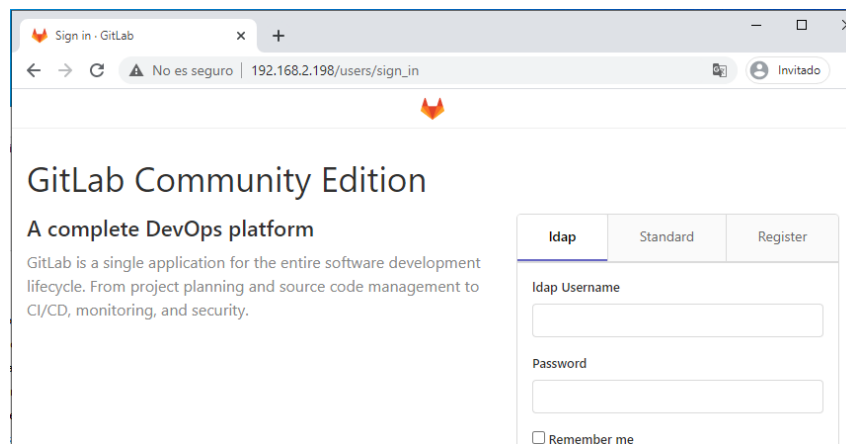


Fig. 14. Inicio de sesión LDAP en GitLab.

/ 7. Caso práctico 2: “Acceso a GitLab LDAP”

Planteamiento: Gonzalo está configurando el nuevo servidor GitLab de la empresa donde trabaja y debe realizar la autenticación de usuarios a través del Directorio Activo de la misma. Además, por motivos de calidad y seguridad, solo podrán iniciar sesión los usuarios pertenecientes al grupo “Desarrolladores”.

Nudo: ¿Qué configuración deberá realizar en el servidor GitLab?

Desenlace: Para poder configurar el inicio de sesión utilizando usuarios del directorio de la empresa, Gonzalo deberá editar el fichero **gitlab.rb** del servidor GitLab “descomentando” y editando las siguientes líneas:

```
gitlab_rails['ldap_enabled'] = true
gitlab_rails['ldap_servers'] = YAML.load <<-'EOS'
main: # 'main' is the GitLab 'provider ID' of this LDAP server
  label: 'ldap'
  host: 'IP_SERVIDOR_LDAP'
  port: 389
  uid: 'sAMAccountName'
  bind_dn: 'usuario@dominio'
  password: 'contraseña'
  encryption: 'plain' # "start_tls" or "simple_tls" or "plain"
  verify_certificates: false
  smartcard_auth: false
  active_directory: true
  allow_username_or_email_login: false
  lowercase_usernames: false
  block_auto_created_users: false
  base: 'CN=Users,DC=dominio,DC=com'
  user_filter: (memberof=CN=GRUPO,DC=dominio,DC=com)'
EOS
```

Código 9. Configuración autenticación de LDAP modificando el fichero `/etc/gitlab/gitlab.rb`.
<https://docs.gitlab.com/ee/administration/auth/ldap/>

Es muy importante que el campo **user_filter** se complete correctamente con la información del grupo.



/ 8. Resumen y resolución del caso práctico de la unidad

En esta unidad, hemos aprendido a trabajar con los **sistemas de control de versiones**, realizando operaciones como la creación de un repositorio. Del mismo modo, hemos visto cómo **instalar y configurar** un servidor de control de versiones y los comandos necesarios para administrar el servicio, así como las ordenes necesarias para ejecutar operaciones avanzadas sobre el repositorio.

Por otra parte, hemos hablado de los puntos básicos que tendremos que revisar para evitar brechas de seguridad en el servidor **GitLab**.

Resolución del caso práctico inicial

En el caso práctico inicial se plantea la siguiente situación:

Dos desarrolladoras trabajan en una aplicación de forma común y utilizan alojamiento en la nube para almacenar los archivos. Por un problema de sincronización, han perdido el último cambio que han realizado y se han planteado buscar otro sistema.

La solución más acertada es utilizar un sistema de control de versiones distribuido. De esta manera, aunque trabajen con el repositorio local, al hacer un *push* al servidor, este detectará automáticamente los cambios y podrá establecer un registro, así como guardar versiones antiguas.



Fig. 15. El servicio de directorio como sistema de autenticación centralizado.

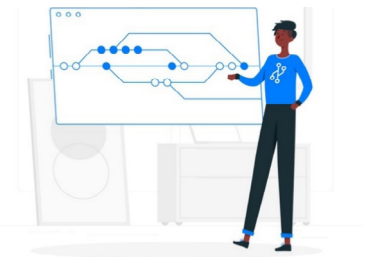


Fig. 16. Un sistema de control de versiones nos permitirá tener copias del estado anterior de un archivo.

/ 9. Bibliografía

Cabello, A.L.C. (2020). *Implantación de aplicaciones web en entornos internet, intranet y extranet*. ifcd0210 - desarrollo de aplicaciones con tecnologías web. IC Editorial.

/ 10. Webgrafía

https://www.ecured.cu/Sistemas_de_control_de_versiones#Clasificaci.C3.B3n

https://es.wikipedia.org/wiki/Control_de_versiones

<https://docs.github.com/es/enterprise/2.19/admin/authentication/using-ldap>

<https://en.wikipedia.org/wiki/Git>

<https://www.techrepublic.com/article/how-to-set-up-a-gitlab-server-and-host-your-own-git-repositories/>

<https://about.gitlab.com/install/?version=ce#ubuntu>

https://docs.gitlab.com/ee/user/admin_area/settings/sign_up_restrictions.html

<https://www.digitalocean.com/community/tutorials/como-instalar-y-configurar-gitlab-en-ubuntu-18-04-es>

<https://docs.gitlab.com/ee/administration/raketasks/ldap.html>

<https://davidjguru.wordpress.com/2010/03/11/sistemas-de-control-de-versiones-iii-un-poco-de-historia/#:~:text=Son%20las%20siglas%20de%20%E2%80%9CSource,en%201972%20por%20Marc%20J.&text=Se%20considera%20el%20primer%20sistema,de%20versiones%20m%C3%A1s%20antigua%20conocida.>