**FORMATO PARA EL DESARROLLO DE COMPONENTE FORMATIVO**

|  |  |
| --- | --- |
| PROGRAMA DE FORMACIÓN | ADSO |

|  |  |  |  |
| --- | --- | --- | --- |
| COMPETENCIA | 220501096. Desarrollar la solución de software de acuerdo con el diseño y metodologías de desarrollo. | RESULTADOS DE APRENDIZAJE | 220501096-01. Planear actividades de construcción del *software* de acuerdo con  el diseño establecido. |

|  |  |
| --- | --- |
| NÚMERO DEL COMPONENTE FORMATIVO | 019 |
| NOMBRE DEL COMPONENTE FORMATIVO | Integración continua |
| BREVE DESCRIPCIÓN | En este componente formativo se especifican las características y elementos necesarios para la implementación de la práctica de integración continua en el contexto del desarrollo de *software*. Se detallan los procesos y herramientas esenciales para asegurar la integración y entrega continuas, permitiendo a los equipos de desarrollo automatizar pruebas, identificar y solucionar problemas de manera temprana, y mejorar la colaboración y eficiencia en la gestión de código y despliegue de aplicaciones. |
| PALABRAS CLAVE | Integración continua, control de versiones, repositorios, Git. |

|  |  |
| --- | --- |
| ÁREA OCUPACIONAL | 2 - CIENCIAS NATURALES, APLICADAS Y RELACIONADAS |
| IDIOMA | Español |

1. **TABLA DE CONTENIDOS:**
2. Sistemas de control de versiones
3. Sistema de control de versiones - Git
4. Comandos básicos de Git
5. Git en entornos remotos
6. Plataformas para implementar integración continua – Gitlab
7. **INTRODUCCIÓN**

La integración continua es una práctica esencial en *DevOps* que asegura un espacio compartido donde los equipos pueden gestionar su trabajo de manera unificada mediante sistemas de control de versiones. Estos espacios, conocidos como repositorios, no solo agrupan archivos, sino que también administran metadatos sobre los cambios y los usuarios que los realizan.

Los sistemas distribuidos, como Git, ofrecen una solución más robusta al permitir que cada cliente tenga una copia completa del repositorio, lo que facilita la restauración en caso de fallos del servidor central. Git, ampliamente utilizado, nació para la comunidad del *kernel* de Linux y permite trabajar sin conexión, garantizando la integridad de los datos mediante algoritmos criptográficos.

|  |  |
| --- | --- |
| GitHub vs. Bitbucket vs. GitLab: Which one is right for your dev team? -  Rewind | Plataformas comoGitHub, GitLab y BitBucket, basadas en Git, facilitan la integración continúa proporcionando repositorios accesibles por internet, donde se pueden gestionar proyectos de *software* de manera colaborativa y segura, optimizando así el desarrollo y mantenimiento de los proyectos. |

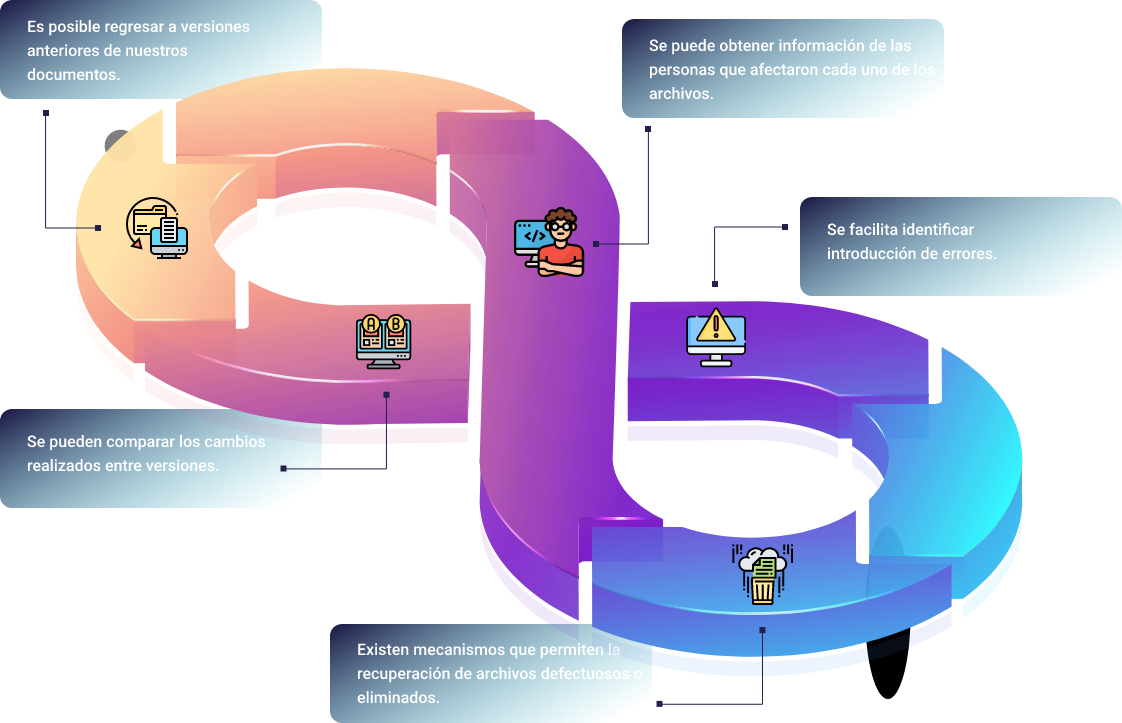
1. **DESARROLLO DE CONTENIDOS:**
2. **Sistemas de control de versiones**

Un sistema de control de versiones se encarga del registro de los cambios realizados a lo largo del tiempo en uno o un conjunto de archivos (versiones), de forma tal que estos puedan ser recuperados con precisión según las necesidades de los usuarios. A este tipo de procesos se le puede denominar **versionamiento** y, aunque estas actividades son comunes en archivos o código fuente en la industria del desarrollo de *software,* el proceso es aplicable a cualquier tipo de archivo.

El proceso de hacer copias de seguridad y crear versiones de los archivos considerados importantes es una actividad muy común.

Así, mediante el uso de un sistema de control de versiones, se obtienen múltiples posibilidades, entre las cuales se encuentran:

**Figura 1.** Beneficios del control de versiones



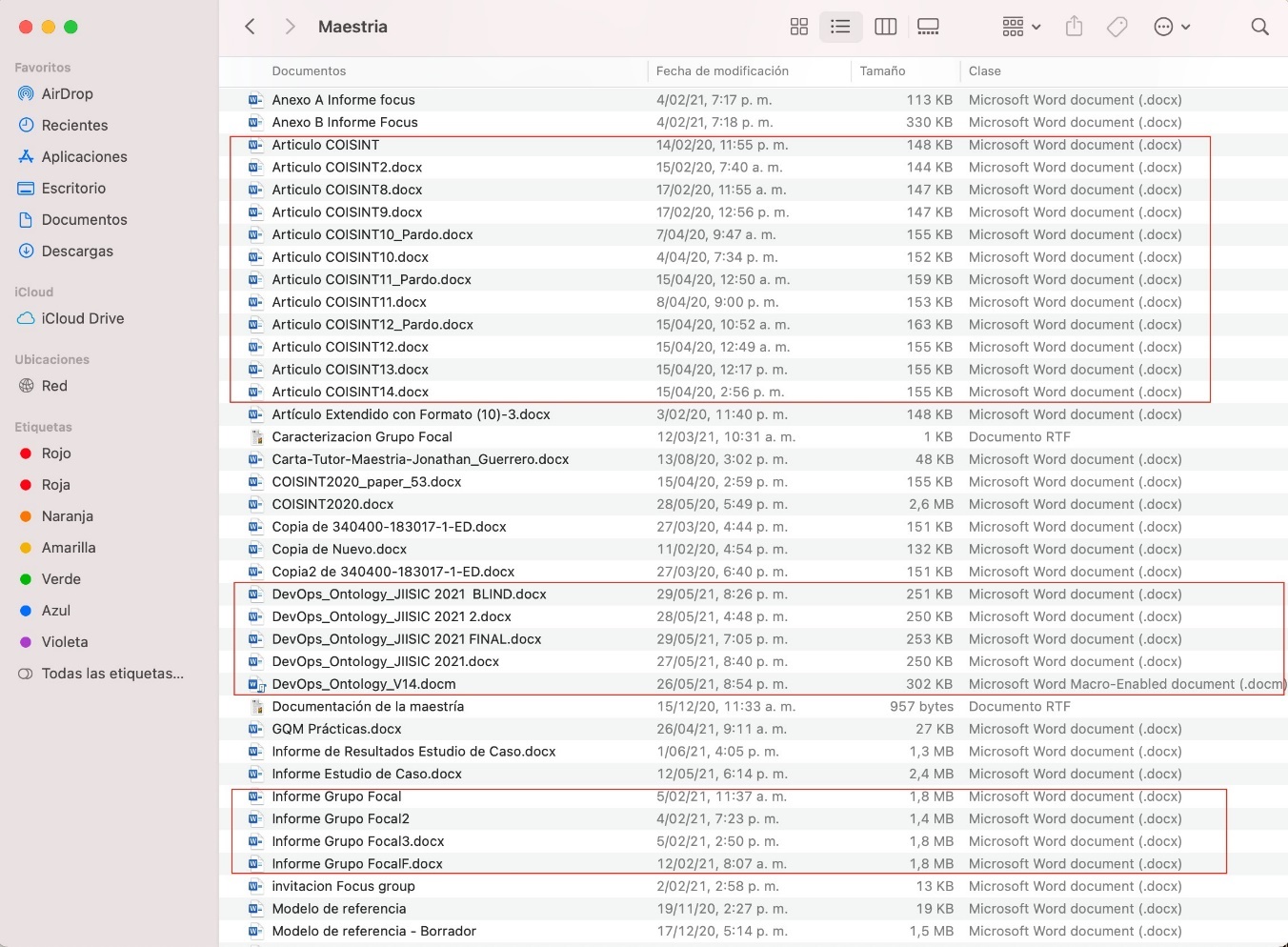
Respecto a los sistemas de control de versiones, estos pueden ser categorizados según sus características en tres grupos: sistemas de control de versiones locales, sistemas de control de versiones centralizados y sistemas de control de versiones distribuidos.

A continuación, se podrán conocer sobre los sistemas de control de versiones locales, centralizadas y distribuidas:

**Sistemas de control de versiones locales**

Una forma de llevar control de versiones locales es crear copias manuales en un directorio exclusivo o de fácil reconocimiento, donde se registre los cambios realizados sobre un documento particular, generalmente mediante una numeración o incluso llevando registro de la fecha de realización de la versión, como se aprecia en la siguiente figura.

**Figura 2.** Ejemplo de los sistemas de control de versiones locales

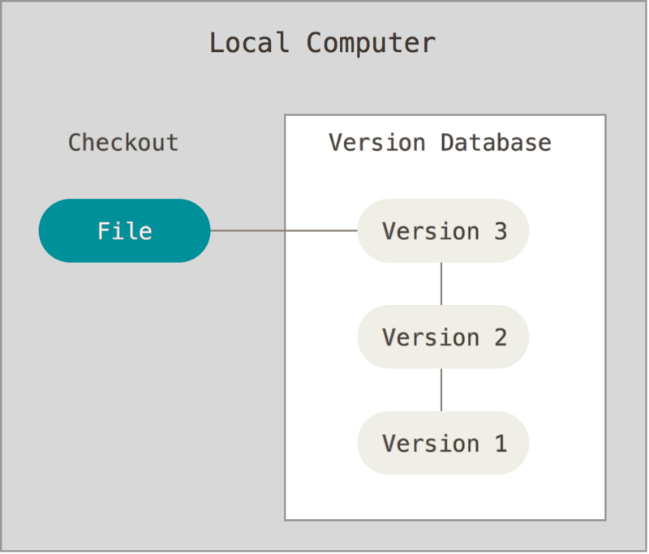
****

**Sistemas de control de versiones locales**

Lo anterior, aunque puede ser una práctica común y fácil de implementar, resulta muy riesgoso, ya que la posibilidad de introducir errores, olvidar la ubicación del directorio, sobrescribir por accidente un archivo o nombrar incorrectamente un archivo es muy alta.

Para solucionar este problema, surgieron hace un tiempo los sistemas de control de versiones locales, que contienen una base de datos simple donde se registra información de todos los cambios realizados sobre los archivos, como se ilustra en la siguiente figura.

**Figura 3.** Esquema de control de versiones en un computador local



**Sistemas de control de versiones centralizados**

Este tipo de sistemas surge para solucionar los problemas asociados a la colaboración en grupo. Bajo el sistema de control de versiones locales, la única forma de colaborar en equipo era mediante un proceso de sincronización manual. Los miembros de un equipo debían sacar copias de los archivos procesados localmente y enviarlas a sus compañeros, quienes, manualmente, debían integrar sus propios cambios y luego repetir el proceso de regreso.

En los sistemas de control de versiones centralizados, se dispone de un servidor donde se almacenan todos los archivos y a partir del cual varios clientes pueden realizar descargas, como se ilustra en la siguiente figura.

**Figura 4.** Ejemplo sistemas de control de versiones centralizados

Sistemas de control de versiones centralizados

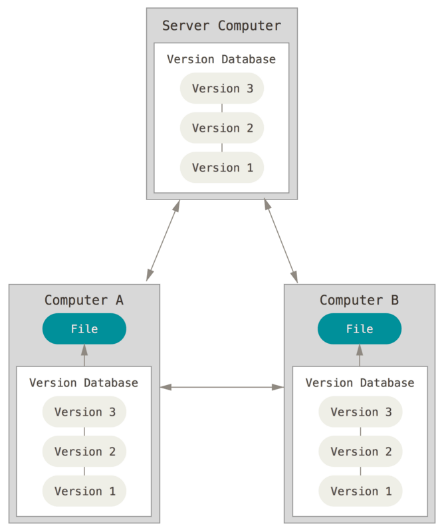

Este tipo de sistemas de control de versiones ha sido ampliamente utilizado en la industria del *software*, ya que facilitan el trabajo colaborativo, permiten un control detallado por parte de los administradores y, son mucho más fáciles de gestionar en comparación con los procesos de sincronización manual que debían realizarse en sistemas de control de versiones locales al trabajar en equipo. Sin embargo, al tratarse de sistemas centralizados, existe una dependencia muy peligrosa del equipo central. Si esta falla por alguna razón bloqueará automáticamente a todos los usuarios y, en casos más extremos, se puede perder todo si el equipo central sufre un fallo irreparable, como fallas en los discos.

**Sistemas de control de versiones distribuidas**

En este tipo de sistemas, los clientes, además de descargar una copia actual de los archivos presentes en el repositorio, crean una réplica completa del mismo. De esta forma, si por alguna razón el servidor falla, los repositorios presentes en los clientes sirven para realizar el proceso de restauración. Es decir, en estos sistemas, los clientes se convierten en clones del repositorio en el servidor.

Este tipo de sistemas permite administrar múltiples repositorios remotos con los que los usuarios pueden trabajar simultáneamente con diferentes grupos de personas en varios proyectos, e incluso dentro del mismo proyecto y de distintas formas. Esto posibilita el establecimiento de varios flujos de trabajo que no son posibles de implementar en sistemas centralizados y mucho menos en sistemas locales.

**Figura 5.** Ejemplo sistemas de control de versiones distribuidas



1. **Sistema de control de versiones - Git**

Git es un sistema de control de versiones que nació en el año 2005, producto de las necesidades que afrontaba la comunidad del *kernel* de Linux para encontrar un mecanismo que les permitiera soportar las actualizaciones y mantenimientos que requería este gran proyecto. Según Git (2021), algunos de los objetivos trazados que debía cumplir este nuevo mecanismo incluían:

Git, a diferencia de otros sistemas de control de versiones, posee un sistema de almacenamiento de información en el que cada versión tiene una copia de instantáneas en miniatura. Es decir, ofrece la posibilidad de trabajo incluso cuando no se tiene conectividad al servidor central, ya que localmente se posee toda la información requerida, incluso si se desea hacer procesos de recuperación a versiones anteriores.

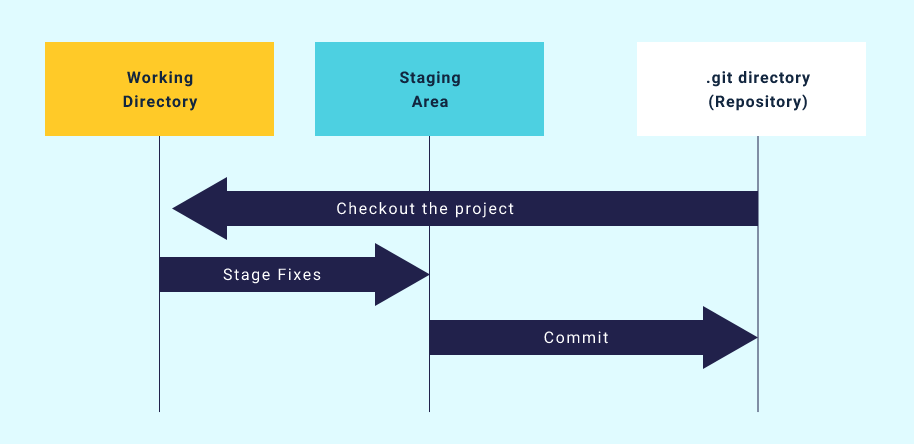
|  |  |
| --- | --- |
| Logotipo de git png imágenes | PNGWing | Git realiza un proceso de comprobación antes de almacenar datos mediante algoritmos con códigos criptográficos SHA-1, que se calculan a partir del contenido y la estructura del directorio del proyecto. Esto hace imposible cambiar el contenido de algún archivo o la estructura de directorios sin que Git lo detecte. |

Debido a que Git solo añade información a la base de datos con cualquier acción realizada, es difícil llegar a un estado en el que este sistema de control de versiones no pueda realizar un proceso de recuperación o que se elimine información de alguna manera una vez exista una confirmación en los cambios.

Según Git (2021), el sistema de control de versiones Git maneja tres estados principales para cada uno de los archivos de un proyecto, los cuales son:

En la siguiente figura se puede identificar la interacción que ocurre entre el directorio de trabajo, el área de preparación y el directorio Git. El directorio de trabajo es donde se desarrollan las labores de construcción del proyecto y corresponde a una copia de la versión del proyecto. El área de preparación es un archivo que almacena información de lo que va en la próxima confirmación. El directorio Git es donde se almacenan los metadatos y la base de datos de objetos para el proyecto. Este directorio Git es lo que se copia en el proceso de clonación del repositorio desde otro equipo de trabajo.

**Figura 6.** Flujo de archivos de acuerdo con el estado



Teniendo en cuenta lo anterior, un flujo normal de acciones para trabajar con Git sería el siguiente:

|  |
| --- |
| Git se puede instalar en cualquier tipo y distribución de sistema operativo e incluso se puede integrar con diferentes IDE para la conexión con varios repositorios locales y en la nube. Para un mejor entendimiento de los comandos más comúnmente utilizados, se optará por la instalación común y la manipulación de este por consola de comandos. |

**Instalación de Git en sistemas operativos**

La forma más sencilla de instalar Git en sistemas operativos Windows y Linux es por medio del sitio oficial disponible.

|  |  |
| --- | --- |
| Closed book with solid fill | **Instalación de Git en sistemas operativos Windows**  Ingrese al paso a paso del proceso de instalación para este sistema. |
| **Instalación de Git distribuciones de Linux**  Ingrese al paso a paso del proceso de instalación para este sistema. |

1. **Comandos básicos de Git**

Para trabajar con Git es importante conocer los diferentes comandos que posibilitan la creación de repositorios, a continuación, se podrá acceder a ellos a través del siguiente recurso:

|  |
| --- |
| **Acordeón**  **CF019\_3\_Comandos básicos de Git** |

1. **Git en entornos remotos**

Para poder colaborar con un equipo de trabajo e implementar el concepto de integración continua, es necesario hospedar los proyectos en repositorios remotos, los cuales son accesibles por medio de internet o por un entorno de red.

|  |  |
| --- | --- |
| Ilustración de concepto de archivos personales digitales | En estos repositorios remotos se definen una serie de permisos que indican si se tiene la posibilidad de solo lectura o de lectura y escritura. Colaborar implica la gestión de estos repositorios, enviando y obteniendo información desde estos, teniendo en cuenta los estados que ya se han mencionado anteriormente. |

A continuación, se relacionan los pasos a seguir para los entornos remotos.

|  |
| --- |
| **Slides**  **CF019\_** **4\_ Git en entornos remotos** |

1. **Plataformas para implementar integración continua – Gitlab**

En la actualidad, existe una gran cantidad de plataformas con servicios en la nube donde se pueden implementar procesos de integración continua. Estas plataformas generalmente ofrecen la posibilidad de crear repositorios públicos y privados, se integran con sistemas de control de versiones, permiten el registro de grupos de trabajo, y muchas otras funcionalidades. Entre las más utilizadas en la industria, basadas en sistemas de control de versiones en Git, se encuentran las descritas a continuación:

|  |  |
| --- | --- |
| Qué es GitHub y por qué es útil en la actualidad 💻 | HACK A BOSS | GitHub  Conoce GitHub un portal creado para alojar códigos de las aplicaciones. <https://Github.com> |
| Secure Code Warrior Socio | GitLab | **GitLab**  Conoce sobre GitLab, un servicio web que permite controlar las versiones y desarrollos de *software* colaborativo basado en Git.  <https://about.gitlab.com/> |
| Bitbucket: qué es, ventajas, quiénes lo usan, diferencia con GitHub y  Gitlab | Leninmhs | **BitBucket**  Conoce a Bitbucket un servicio de alojamiento basado en web para los proyectos que utilizan sistema de control de versiones Mercurial y Git.  <https://bitbucket.org/> |

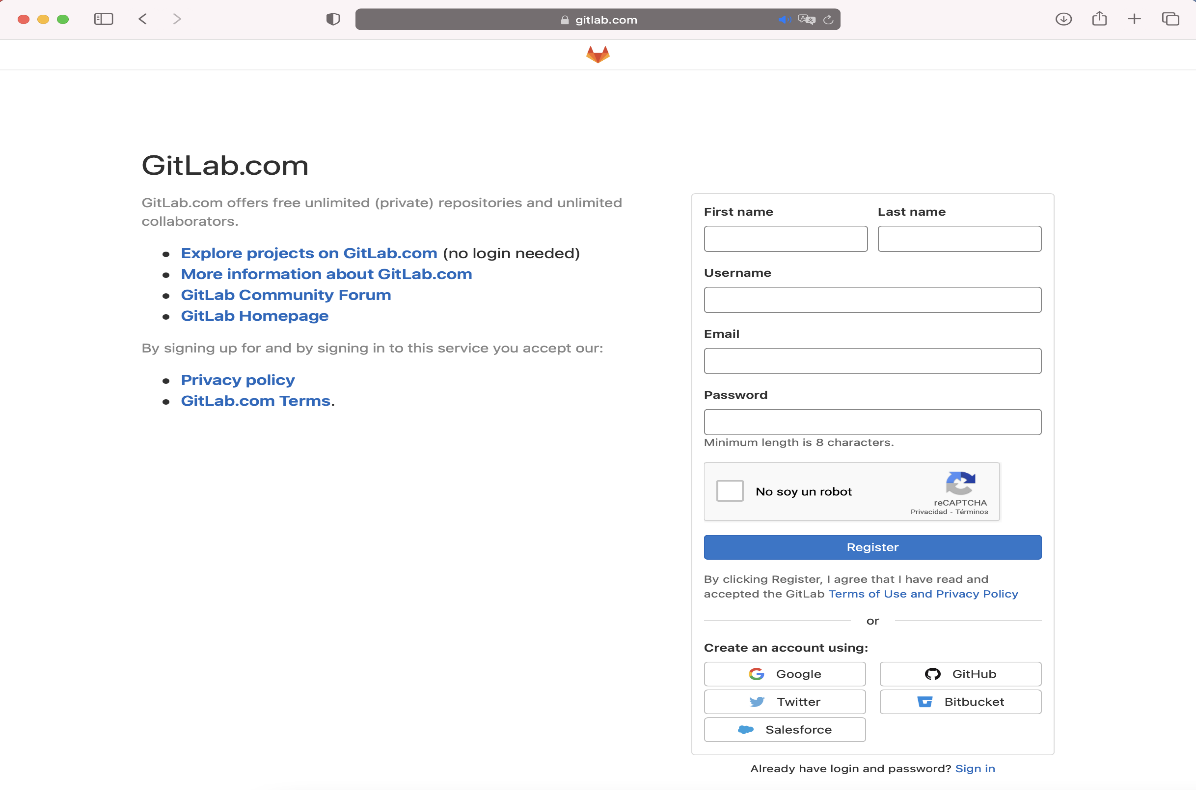
Para complementar la información sobre los anteriores sistemas, a continuación, se presentarán las diferentes versiones gratuitas con sus funcionalidades básicas, las cuales son accesibles simplemente con la creación de una cuenta de usuario.

**Creación de cuenta**

Primero, se debe crear una cuenta en Gitlab directamente desde el enlace: <https://gitlab.com/users/sign_up>

El proceso se puede realizar de forma tradicional o usando cuentas ya existentes de otras plataformas como Google, Github, Twitter, Bitbucket o Salesforce.

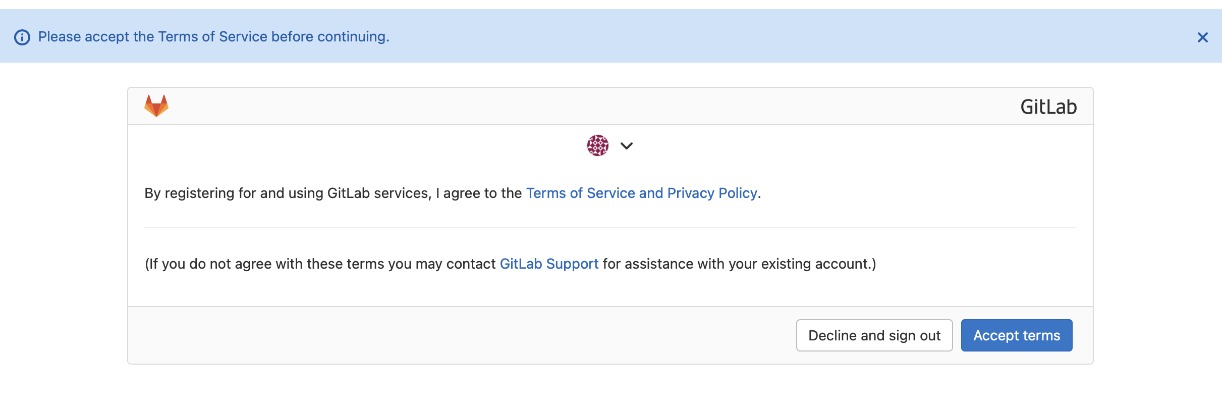
**Figura 7.** Página de registro de GitLab



**Aceptación de términos de servicios**

Una vez se realiza el registro, se deben aceptar los términos del servicio y las políticas de privacidad.

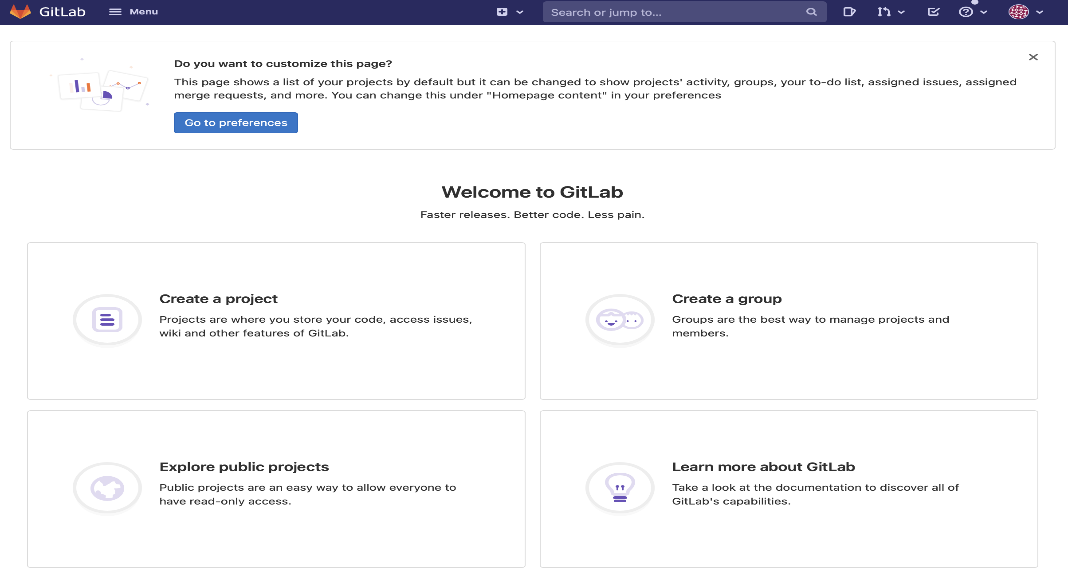
**Figura 8.** Aceptación de términos y condiciones



**Interfaz de inicio**

Después de esto, se muestra la interfaz de inicio donde se puede crear directamente el proyecto (repositorio).

**Figura 9.** Página de bienvenida de GitLab

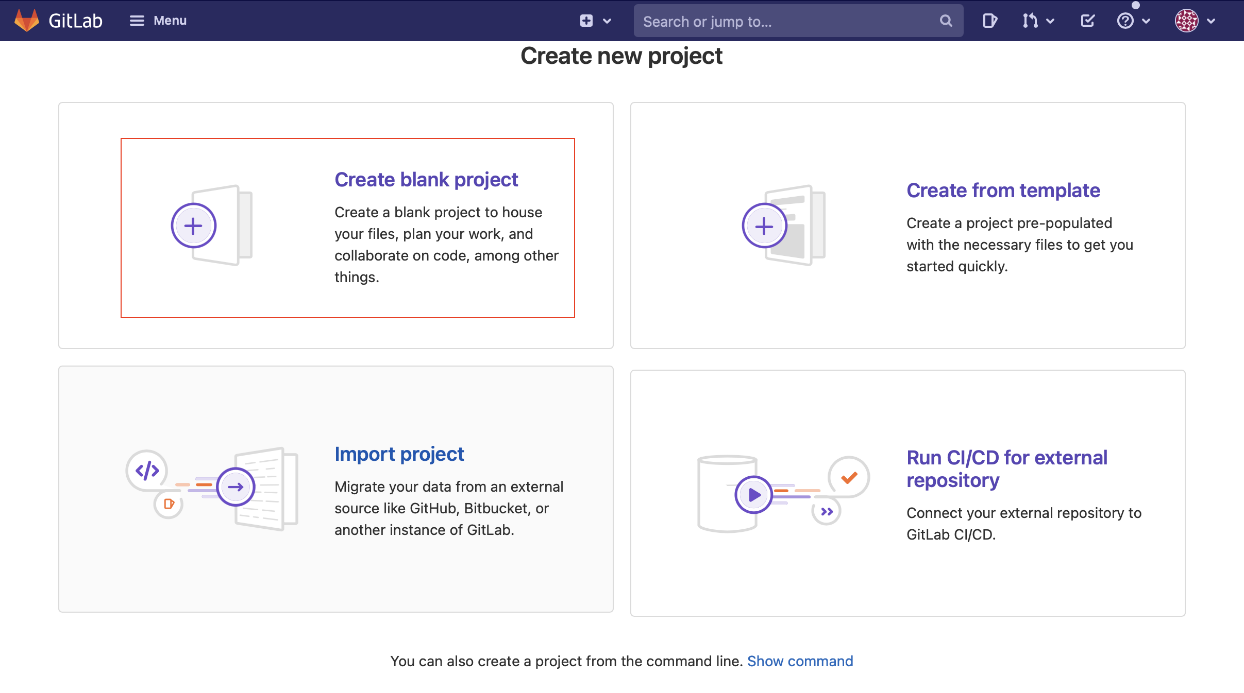


1. **Creación de proyecto**

En la parte de creación de proyectos, se ofrecen cuatro posibilidades:

Para los grupos de desarrollo que apenas inician con prácticas de integración continua, el proyecto en blanco es la mejor opción.

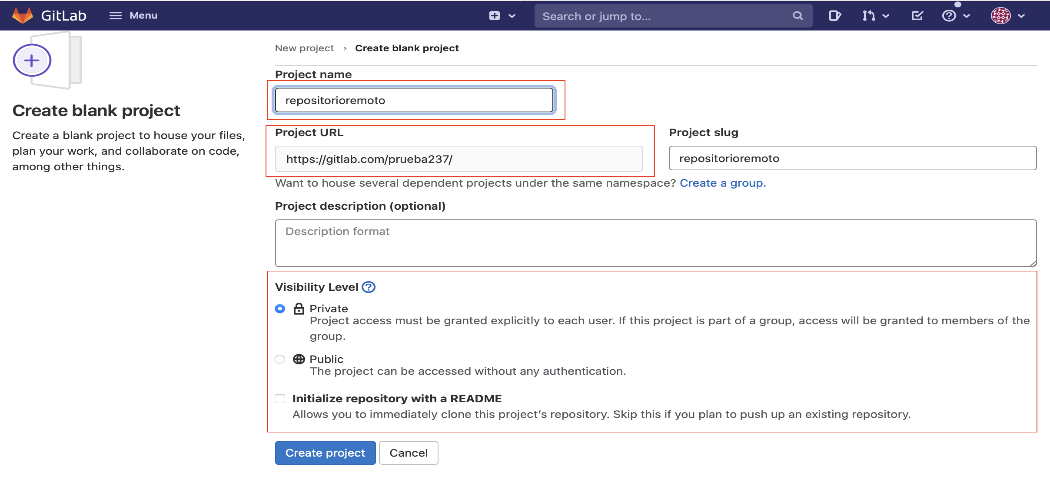
**Figura 10.** Creación de un nuevo proyecto



1. **Nombre del proyecto**

A continuación, se debe definir el nombre del proyecto, agregar una descripción y definir el tipo de visibilidad, que puede ser pública (cualquiera podría acceder sin autenticarse) o privada (se deben tener permisos asignados para ingresar). La URL asignada dependerá del nombre de usuario asignado a la cuenta de Gitlab creada y del nombre definido al repositorio.

**Figura 11**. Creación de un proyecto en blanco

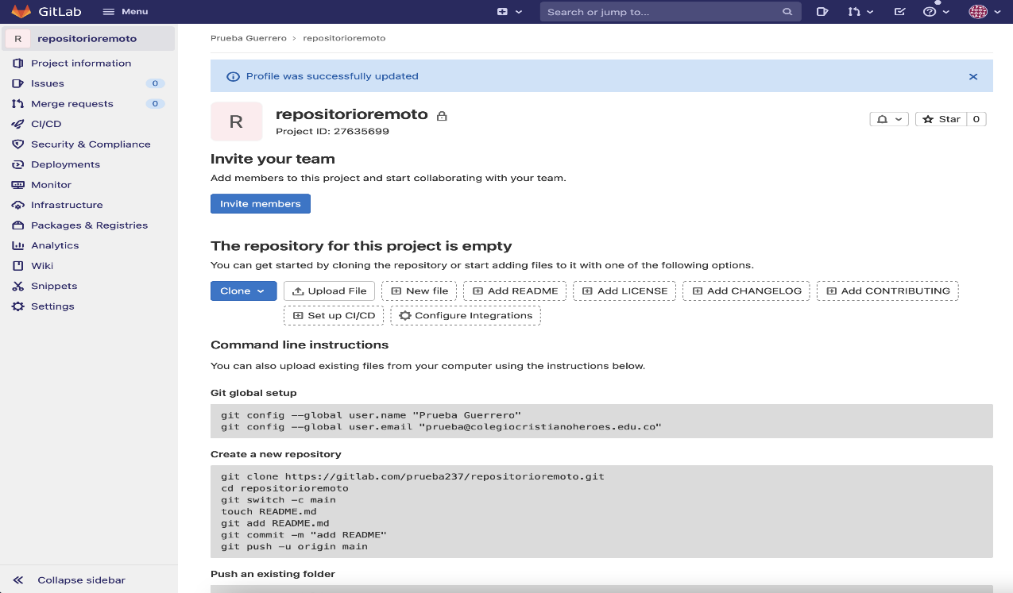


1. **Instrucción de uso**

Una vez finalizado el proceso, el sistema proporciona un conjunto de instrucciones para usar el repositorio, entre las cuales se encuentra la configuración del usuario y correo de acuerdo con las credenciales de Gitlab y los comandos Git para hacer su respectiva gestión.

A continuación, lo único que queda por hacer es conectar el equipo local con el repositorio remoto por medio de Git usando los comandos mencionados anteriormente. En este caso, se usará la ayuda ofrecida por Gitlab para crear un nuevo repositorio local realizando la clonación respectiva desde el repositorio remoto.

**Figura 12**. Repositorio vacío



1. **Modificación de usuario**

En primer lugar, se debe modificar el usuario y la cuenta de correo en la máquina local para que coincida con las credenciales autorizadas en el repositorio remoto.

|  |
| --- |
| root@ubuntu-VirtualBox:/home/ubuntuser# git config --global user.name "Prueba Guerrero"  root@ubuntu-VirtualBox:/home/ubuntuser# git config --global user.email "prueba@colegiocristianoheroes.edu.co"  root@ubuntu-VirtualBox:/home/ubuntuser# |

1. **Clonación de repositorio**

Luego, se clona el repositorio creado en Gitlab para copiarlo en la máquina local. En este punto, se debe tener claridad sobre el usuario y su contraseña en Gitlab, ya que esta información será requerida durante el proceso de clonación. Una vez se verifican las credenciales, se realiza el proceso de clonación. En este caso, es un repositorio vacío, pero se encontrará en el sistema local de archivos una carpeta con el mismo nombre definido en el repositorio remoto, con toda la configuración necesaria para la gestión de estados con Git. Este repositorio se convierte en el directorio de trabajo local.

|  |
| --- |
| root@ubuntu-VirtualBox:/home/ubuntuser# git clone https://gitlab.com/prueba237/repositorioremoto.git  Clonando en 'repositorioremoto'...  Username for 'https://gitlab.com': prueba237  Password for 'https://prueba237@gitlab.com':  warning: Pareces haber clonado un repositorio sin contenido.  root@ubuntu-VirtualBox:/home/ubuntuser# |

1. **Acceso al directorio**

Ahora, se accede al directorio de trabajo para empezar la construcción de archivos, ponerlos en rastreo, prepararlos, confirmarlos localmente y luego enviar la versión confirmada localmente al repositorio remoto. Se debe utilizar el comando **git switch -c nombre** para crear una nueva rama de trabajo local, donde "nombre" corresponde al nombre de la rama.

|  |
| --- |
| root@ubuntu-VirtualBox:/home/ubuntuser# cd repositorioremoto/  root@ubuntu-VirtualBox:/home/ubuntuser/repositorioremoto# git switch -c main  Cambiado a nueva rama 'main' |

1. **Creación de archivo local**

Después de esto, se probará crear un archivo local vacío llamado **README.md**, el cual se preparará, confirmará y enviará al repositorio remoto.

|  |
| --- |
| root@ubuntu-VirtualBox:/home/ubuntuser/repositorioremoto# touch README.md |

1. **Archivo de rastreo**

Luego, se agrega este archivo para que sea rastreado y agregado al área de preparación.

|  |
| --- |
| root@ubuntu-VirtualBox:/home/ubuntuser/repositorioremoto# git add README.md |

1. **Archivo de repositorio local**

Se confirma el archivo en el repositorio local, lo cual genera una nueva versión en el repositorio local.

|  |
| --- |
| root@ubuntu-VirtualBox:/home/ubuntuser/repositorioremoto# git commit -m "Agregado readme"  En la rama main  Tu rama está actualizada con 'origin/main'. |

1. **Envío al repositorio remoto**

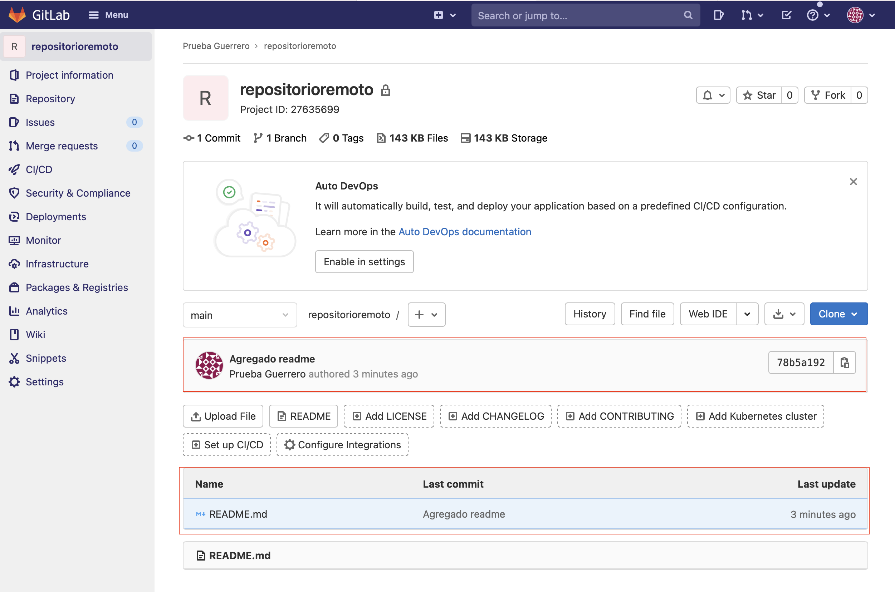
Luego de tener una versión confirmada, se envía al repositorio remoto. Se deberán ingresar las credenciales de usuario en Gitlab.

|  |
| --- |
| root@ubuntu-VirtualBox:/home/ubuntuser/repositorioremoto# git push -u origin main  Username for 'https://gitlab.com': prueba237  Password for 'https://prueba237@gitlab.com':  Enumerando objetos: 3, listo.  Contando objetos: 100% (3/3), listo.  Escribiendo objetos: 100% (3/3), 230 bytes | 57.00 KiB/s, listo.  Total 3 (delta 0), reusado 0 (delta 0)  To https://gitlab.com/prueba237/repositorioremoto.git  \* [new branch] main -> main  Rama 'main' configurada para hacer seguimiento a la rama remota 'main' de 'origin'. |

1. **Repositorio Gitlab**

En este punto, los cambios aplicados ya se pueden visualizar directamente desde la plataforma Gitlab.

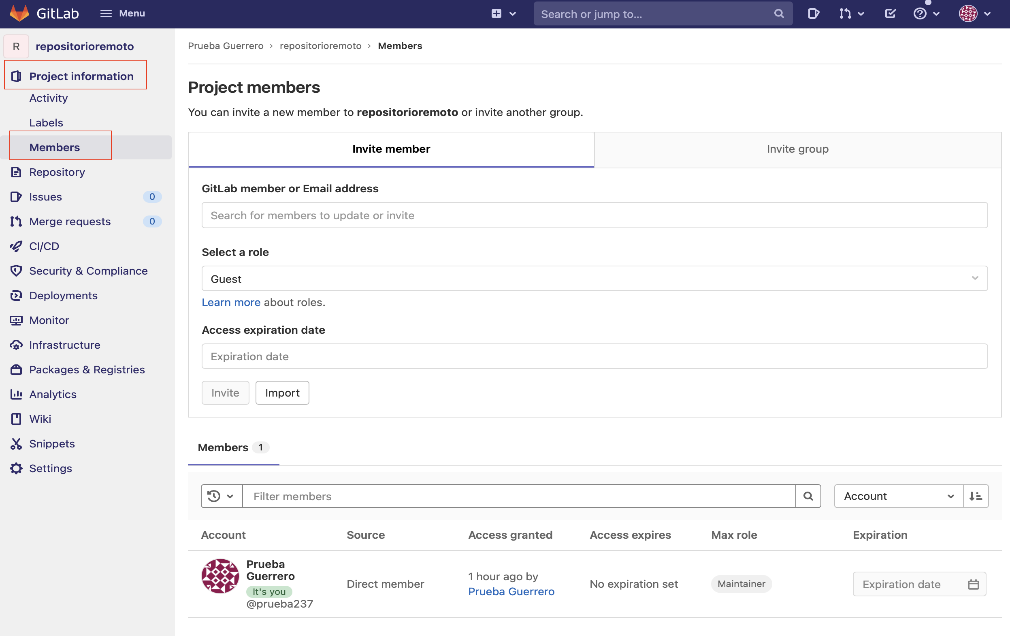
**Figura 13.** Vista del repositorio con archivo README



1. **Agregar colaboradores**

Para agregar colaboradores que también puedan clonar, preparar, confirmar y subir cambios al repositorio, es necesario agregarlos como colaboradores y asignarles los permisos requeridos, ya que el repositorio fue definido como privado. Para esto, se debe regresar a Gitlab y, en el menú del panel del lado izquierdo, seleccionar la opción de información del proyecto y luego miembros.

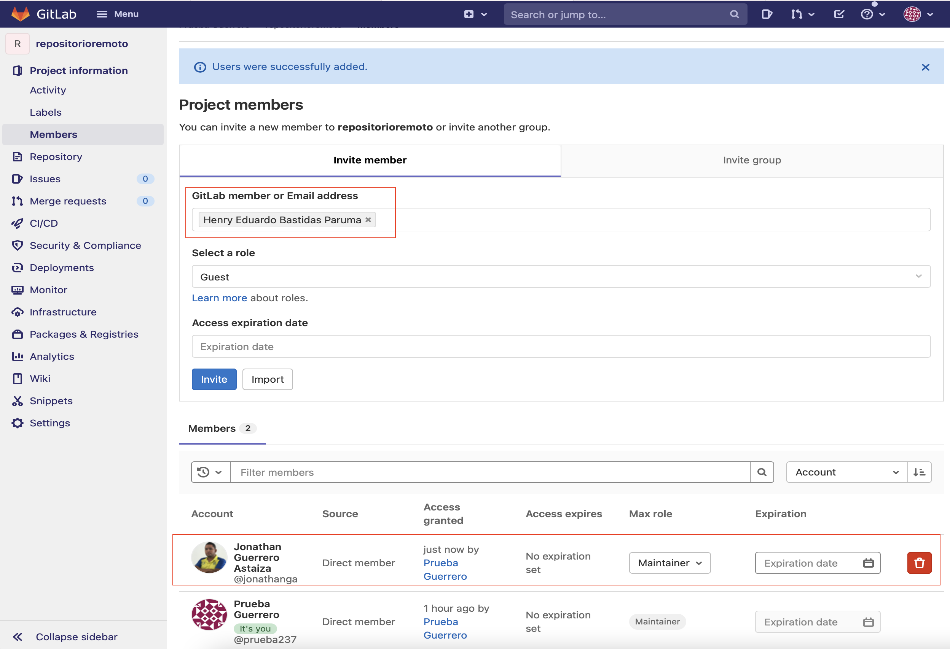
**Figura 14.** Gestión de miembros del proyecto



1. **Configuración de roles**

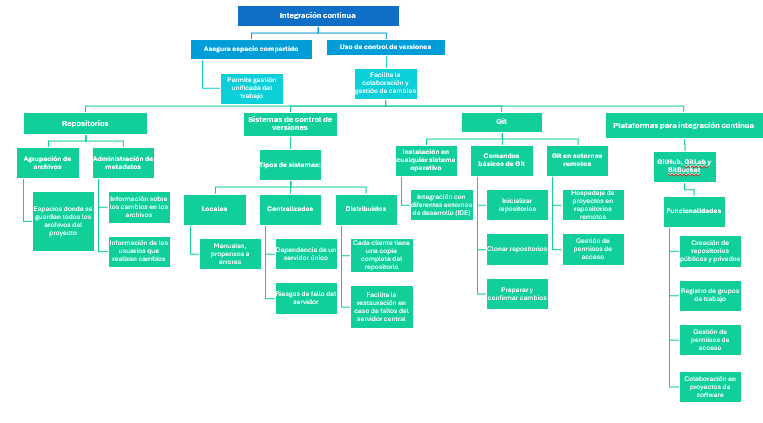
Por defecto, el usuario creador del repositorio aparece como miembro de este. Mediante esta interfaz, se pueden invitar a nuevos miembros por correo electrónico o por nombre de usuario, si ya están registrados en Gitlab. Adicionalmente, se puede configurar el rol asociado a este usuario e incluso definir la fecha en la que expira la autorización asignada. Si se desea asignar permisos de lectura y escritura, se recomienda asignar el rol *Maintainer*. Usando este mismo espacio, se pueden revocar permisos y eliminar colaboradores del proyecto. Todos los usuarios autorizados podrán realizar la clonación del repositorio e interactuar con él por medio de los comandos Git vistos a lo largo de este módulo.

**Figura 15.** Adición de un nuevo miembro al proyecto



1. **SÍNTESIS**

A continuación, se presenta una síntesis de la temática estudiada en el componente formativo.



1. **ACTIVIDADES DIDÁCTICAS (Se debe incorporar mínimo 1, máximo 2)**

|  |  |
| --- | --- |
| DESCRIPCIÓN DE ACTIVIDAD DIDÁCTICA | |
| Nombre de la Actividad | Implementación de integración continua |
| Objetivo de la actividad | Fortalecer las prácticas de integración continua y el uso de sistemas de control de versiones, específicamente Git y GitLab |
| Tipo de actividad sugerida | Relacionar conceptos |
| Archivo de la actividad  (Anexo donde se describe la actividad propuesta) | CF019\_Actividad didactica |

1. **MATERIAL COMPLEMENTARIO:**

|  |  |  |  |
| --- | --- | --- | --- |
| Tema | Referencia APA del Material | Tipo de material  (Video, capítulo de libro, artículo, otro) | Enlace del Recurso o  Archivo del documento o material |
| Comandos básicos de Git | Fazt. (2014). Git and GitHub for Poets [Video]. YouTube. | Video | <https://youtu.be/HiXLkL42tMU> |
| Comandos básicos de Git | GIT. (s.f.). Git - Download for Windows. | Software | [http://Git-scm.com/download/win](http://git-scm.com/download/win) |

1. **GLOSARIO:**

|  |  |
| --- | --- |
| TÉRMINO | SIGNIFICADO |
| BitBucket: | servicio de alojamiento basado en web para proyectos que utilizan Mercurial y Git. |
| Comandos básicos de Git: | comandos como git init, git clone, git add, git commit, git push para gestionar repositorios. |
| *DevOps:* | metodología de desarrollo de *software* que integra el desarrollo (Dev) y las operaciones (Ops) para mejorar la colaboración y productividad mediante la automatización de procesos y la entrega continua de *software.* |
| Git: | sistema de control de versiones distribuido que permite trabajar sin conexión y garantiza la integridad de datos. |
| GitHub: | plataforma para alojar código de aplicaciones y colaborar en proyectos. |
| GitLab: | plataforma web que permite el control de versiones y desarrollo colaborativo basado en Git. |
| Hospedaje de proyectos: | proceso de alojar proyectos en repositorios remotos accesibles por internet. |
| Integración continua: | práctica de *DevOps* que asegura la disponibilidad de un espacio compartido y gestión unificada del trabajo. |
| Repositorios: | espacios donde se agrupan archivos y se administran metadatos relacionados con los cambios realizados. |
| Sistemas de control de versiones: | sistemas que registran los cambios en archivos a lo largo del tiempo, permitiendo su recuperación. |

1. **REFERENCIAS BIBLIOGRÁFICAS:**

Atlassian. (s.f.). *Qué es el control de versiones | Atlassian Git Tutorial*. [https://www.atlassian.com/es/Git/tutorials/what-is-version-control](https://www.atlassian.com/es/git/tutorials/what-is-version-control)

Git. (2021) *Git - Acerca del control de versiones*. Git --Local-Branching-on-the-Cheap.[https://Git-scm.com/book/es/v2/Inicio---Sobre-el-Control-de-Versiones-Acerca-del-Control-de-Versiones](https://git-scm.com/book/es/v2/Inicio---Sobre-el-Control-de-Versiones-Acerca-del-Control-de-Versiones)

Younes. (2021). *Gitlab VS Github VS BitBucket. Which one deserve your time?* DEV Community. <https://dev.to/yafkari/gitlab-vs-github-vs-bitbucket-which-one-deserve-your-time-2npm>

1. **CONTROL DEL DOCUMENTO**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Nombre | Cargo | Dependencia  *(Para el SENA indicar Regional y Centro de Formación)* | Fecha |
| Autor (es) | Jonathan Guerrero Astaiza | Experto Temático | Regional Cauca - Centro de teleinformática y producción industrial | Noviembre 2022 |
| Paola Alexandra Moya | Evaluadora instruccional | Regional Antioquia - Centro de Servicios de Salud | Mayo 2024 |
|  | Olga Constanza Bermúdez Jaimes | Responsable Línea de Producción Antioquia | Regional Antioquia - Centro de Servicios de Salud | Mayo 2024 |

1. **CONTROL DE CAMBIOS**

**(Diligenciar únicamente si realiza ajustes a la Unidad Temática)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Nombre | Cargo | Dependencia | Fecha | Razón del Cambio |
| Autor (es) |  |  |  |  |  |