Procedimiento para la implementacion de software para el control de versiones en ambientes colaborativos y distribuidos

Andrés Eduardo Meneses Cáceres

Universidad de Pamplona

Facultad de Ingenierías y Arquitectura

Programa de Ingeniería de Sistemas

Pamplona, Norte de Santander

2017

Procedimiento para la implementación de software para el control de versiones en ambientes colaborativos y distribuidos

Andrés Eduardo Meneses Cáceres

Trabajo de grado presentado como requisito para optar al título de

INGENIERO DE SISTEMAS

Director: Luis Alberto Esteban Villamizar

Licenciado en Matemáticas y Computación

Mg. en Informática

lesteban@unipamplona.edu.co

Universidad de Pamplona

Pamplona, Norte de Santander

2017

Contenido

1 Introducción 1

1.1 Planteamiento del problema 2

1.2 Objetivos 3

1.2.1 Objetivo general 3

1.2.2 Objetivos específicos 3

1.3 Justificación 3

1.4 Metodologia 4

2 Marco teórico 6

2.1 Antecedentes 6

2.2 Software Libre 7

2.3 Open source 9

2.4 Diferencia entre Software Libre y Open Source 12

2.5 Comunidades open source 12

2.6 Control de versiones 15

2.6.1 Terminologia 17

2.6.2 Funcionamiento de los VCS 20

2.6.3 Clasificación de los VCS 20

2.7 Herramientas para el Control de Versiones (VCS) 23

2.7.1 Concurrent Versions System (CVS) 23
2.7.2 Subversion (SVN) 24
2.7.3 Bazaar 26
2.7.4 Git 28
2.7.5 Mercurial 36
2.8 Comparacion de algunos VCS 36
2.8.1 Ventajas y Desventajas de algunos VCS 36
2.8.2 Información general 39
2.8.3 Información Técnica 40
2.8.4 Comandos Basicos 40
3 Procedimiento 42
3.1 Descripcion del procedimiento 42
3.1.1 Planificacion del proyecto 43
3.1.2 Instalación y Configuración de VCS 44
3.1.3 Inicializacion de Repositorio 45
3.1.4 Gestion de Ramas 47
3.1.5 Gestion de Tags 48
3.1.6 Integracion con Merge 48
3.1.7 Gestion de IES 49
3.2 Validacion del Procedimiento 50

3.2.1 Planificacion del proyecto 50
3.2.2 Instalacion y Configuracion de VCS 51
3.2.3 Inicializacion de Repositorio 51
3.2.4 Gestion de Ramas 55
3.2.5 Gestion de Tags 56
3.2.6 Integracion con Merge 58
3.2.7 Gestion de IES 60
4 Conclusiones, recomendaciones y trabajos futuros 61
4.1 Conclusiones 61
4.2 Recomendaciones 63
4.3 Trabajos futuros 63
5 Referencias Bibliografía 64
5.1 Bibliografia 64
5.2 Infografia 66

Tabla de Ilustraciones

Ilustración 1: Rama 18
Ilustración 2: Fusion de Ramas 19
Ilustración 3: VCS centralizados 21
Ilustración 4: VCS Distribuidos 22
Ilustración 5: Funcionamiento CVS 24
Ilustración 6:Funcionamiento SVN 25
Ilustración 7: Bazaar, funcionamiento en pareja 27
Ilustración 8: Bazaar, funcionamiento centralizado 27
Ilustración 9:Bazaar, funcionamiento descentralizado con supervisor 28
Ilustración 10: Funcionamiento Git 29
Ilustración 11:HEAD en Git 31
Ilustración 12: Procedimiento propuesto 42
Ilustración 13:Planificacion del proyecto 43
Ilustración 14:Crear proyecto Gitlab 46
Ilustración 15:Merge request 49
Ilustración 16:Gestion IES 50
Ilustración 17:Planificacion Proyecto CursosLibres 51
Ilustración 18:Creacion de repositorio 52
Ilustración 19:Inicializacion de repositorio 52
Ilustración 20:Archivos controlados 53
Ilustración 21:Roles gitlab 53

Ilustración 22: Protegiendo una rama 55

Ilustración 23:Ramas protegidas en ProyectoCursosLibres 56

Ilustración 24:Accediendo a un tag 57

Ilustración 25:Lista de merge request 58

Ilustración 26: Ventana de fusion 59

Índice de tablas

Tabla 1: Roles en una Comunidad Open Source 14

Tabla 2:Permisos Gitlab 36

Tabla 3: Ventajas y Desventajas de algunos VCS 38

Tabla 4: Informacion General VCS 39

Tabla 5: Informacion Tecnica VCS 40

Tabla 6: Comandos basicos 41

Tabla 7: Permisos a ramas 47

1 Introducción

En el desarrollo de software se presentan gran numero de procesos tanto técnicos como de gestión que son necesarios para lograr el objetivo de un proyecto de software. Frecuntemente se le ha dado más importancia a los procesos técnicos como la escritura de codigo fuente, el diseño, las pruebas de software, frente a los procesos de gestion como los costos, los riesgos, el manejo de personal, la gestion de configuración, entre otros, sin embargo es importante reconocer que todos ellos hacen son indispensables para garantizar el éxito de un proyecto.

En la actualidad las compañías de desarrollo de software cuentan con herramientas que facilitan la gestión de un proyecto durante la evolución de este y entre esta herramientas estan los sistemas de control de versiones (VCS) los cuales permiten controlar las modificaciones que se realizan a artefactos documentados y código fuente dentro de un proyecto, facilitando la coordinación entre distintos desarrolladores y evitando conflictos al momento de lanzar un producto final.

En este proyecto se estudió el funcionamiento de algunas comunidades bajo concepto de open source, precisamente se observó como los usuarios pueden contribuir a un proyecto mediante el uso de un sistema de control de versiones propio de la comunidad. Ademas se tuvo en cuenta la experiencia obtenida en el centro de investigación y desarrollo de la Universidad de Pamplona, en donde se colaboro con el desarrollo de una nueva version de Academusoft.

Como producto final se obtuvo un procedimiento para implementar un sistema de control de versiones utilizando herramientas open Source como gitlab en la Universidad de Pamplona.

Este documento esta organizado en cuatro capitulos

En el primer capitulo se describe el alcance del trabajo realizado como pasantia en el centro de investigación y desarrollo de la Universidad de Pamplona.

En el segundo capitulo se describen los temas que fueron fundamentales para la construccion del procedimiento para el control de versiones en ambientes colaborativos y distribuidos.

En el tercero se describe el procedimiento y su validacion mediante el uso del sistema de control de versiones git y la interfaz web gitlab. El procedimiento estara estructurado con los siguientes pasos: Planificacion del proyecto, Instalacion y configuracion de VCS, Inicializacion de repositorio, Gestion de ramas, Gestion de tags, Integracion con Merge, Gestion de IES.

En el cuarto se presentan conclusiones y trabajos futuros.

1.1 Planteamiento del problema

Anteriormente los profesionales en desarrollo de software se encontraban con algunos inconvenientes a la hora de trabajar en un proyecto en el que participan muchas personas, principalmente se enfrentaban a proyectos difíciles de gestionar y liderar, imposibilidad de trabajar de forma remota, y problemas al modificar el código formal del equipo de desarrollo.

Hoy en día en los proyectos de desarrollo de software de calidad, se necesitan muchos perfiles o roles en la construcción de este: hay personal encargado del frontend, backend, administración de base de datos. Etc. Por esto la necesidad de implementar herramientas que permitan el control de versiones está presente en todos los sectores donde se desarrolle software de calidad, estas herramientas permiten controlar las modificaciones hechas por los demás integrantes del equipo de desarrollo para que no exista ninguna interferencia al momento de lanzar un producto final.

En el CIADTI de la Universidad de Pamplona no se cuenta con un procedimiento debidamente documentado para gestionar y controlar las modificaciones en el desarrollo de

software, esto motivo el desarrollo del presente trabajo esperando que a futuro permita organizar los equipos de desarrollo y trabajar de forma más eficiente.

1.2 Objetivos

1.2.1 Objetivo general

- Diseñar un procedimiento que permita administrar el control de versiones en ambientes de desarrollos de software colaborativos y distribuidos

1.2.2 Objetivos específicos

- Estudiar el funcionamiento de comunidades open source.
- Proponer un procedimiento para utilizar estas herramientas enfocadas en el control de versiones.
- Validar el procedimiento en un ambiente simulado en la Universidad de Pamplona

1.3 Justificación

La guia del cuerpo del conocimiento en ingenieria del software SWEBOK (Bourque & Fairley, 2014) define unas areas del conocimiento en el desarrollo de software de calidad: requerimientos, diseño, construccion, pruebas y mantenimiento, gestion de configuracion, gestion de ingenieria, proceso de ingenieria, herramientas y metodos, calidad.

Este proyecto se adentra dentro de una de las areas del conocimiento que define el SWEBOK: la gestion de configuracion del software, que durante el ciclo de desarrollo de un software se encarga de identificar la configuracion del sistema en distintos momentos en el tiempo, con el fin de controlar sistematicamente los cambios y mantener la integridad.

La gestion de configuracion del software abarca muchos temas desde la identificacion de elementos a controlar durante el ciclo de vida del software, pasando por el control, autorizacion e implementacion de los cambios, para terminar con la liberacion del software.

Teniendo en cuenta la guia SWEBOK es necesario el uso de una herramienta que permita controlar los diferentes estados por los que pasa un proyecto de desarrollo de software durante su ciclo de vida, permitiendo gestionar los cambios que se realizan ademas de aprobar dichos cambios para posteriormente ser implementados en la linea base del proyecto.

Este proyecto propone el uso de sistemas de control de versiones para un trabajo en equipo más eficiente y organizado, por tanto se observaran algunas de las herramientas más comunes en cuanto a control de versiones se refiere, analizando algunas características, ventajas y desventajas de cada una, haciendo énfasis claro, en las herramientas de software libre.

Ademas se tiene en cuenta la experiencia del trabajo como pasantia en el centro de investigacion y desarrollo de la Universidad de Pamplona, el cual se toma como base para proponer un procedimiento que mejore la gestion en el control de estados por los que pasa un software durante su ciclo de vida, ademas de la autorizacion de cambios a la linea base del proyecto.

1.4 Metodologia

El primer objetivo de este proyecto es entender cómo funcionan las comunidades open source, con un enfoque principal en el uso y funcionamiento de herramientas destinadas al control de versiones. Una vez comprendido el funcionamiento de estas herramientas y en como usuarios acceden a estas comunidades para visualizar, descargar y realizar aportes sobre un tema

específico, es posible proponer un procedimiento para el propio control de versiones en la Universidad de Pamplona.

Este procedimiento consiste en crear un proyecto generico que permita explorar algunas de las muchas opciones que ofrecen las herramientas de control de versiones, en este caso Git.

Para cumplir con el primer objetivo se analizaron algunas comunidades tales como GNOME, Alliance for Open Media, Django Software Foundation, Mozilla, y se observaron algunas forjas[1] tales como github, o sourceforge; estas comunidades utilizan Git como el sistema de control de versiones para sus proyectos open source, o utilizan forjas como github en el caso de Mozilla para poder contribuir en sus proyectos. Por este motivo se analizó la forma en la que funciona la colaboracion en un proyecto open source realizada por un usuario. En estas comunidades es muy importante el aporte de cualquier usuario para sus proyectos por esta razon los proyectos son publicos para que el interesado en algun proyecto en concreto pueda revisar tanto el codigo como la documentacion de esta, pudiendo ademas, realizar su respectivo aporte a este proyecto, no obstante, para algunos proyectos en github es necesario realizar un *push request*, el cual notificara al creador del proyecto que existen *commits* que pueden ser añadidos a su proyecto, y este decidira si desea incorporar los cambios realizados a su proyecto.

Para la respectiva validacion del procedimiento se realizó la instalacion y configuracion de git en los servidores de la Universidad de Pamplona, añadiendo la instalacion de gitlab, que permite gestionar de forma sencilla los usuarios y los permisos que tendran estos en algunos repositorios.

[1] Una forja es una plataforma de desarrollo colaborativo de software. Se enfoca hacia la cooperación entre desarrolladores para la difusión de software y el soporte al usuario, en una forja se encuentran diversos proyecto en los cuales los interesados pueden contribuir en su desarrollo.

2 Marco teórico

En este capitulo se encuentra la documentacion sobre los movimientos de Software Libre y Open Source, ademas se aborda el concepto de control de versiones, su funcionamiento, clasificacion y terminologia, por ultimo se describen algunas herramientas para el control de versiones y se realiza una comparacion entre ellas.

2.1 Antecedentes

Un primer trabajo de investigación revisado corresponde al de (Bartolomé, 2014) quien realizó el "Desarrollo de Software Open Source Analizado desde Dentro", En este trabajo se estudiaron los diferentes procesos en el desarrollo de software en comunidades colaborativas y distribuidas para realizar una comparación con los procesos en el desarrollo tradicional de software. Esto significa estudiar los medios de comunicación existentes entre los miembros de las comunidades, sus motivaciones, los métodos de trabajo, qué documentos se desarrollan y qué herramientas se utilizan.

Esta investigacion se relaciona con el trabajo presente ya que analizan cual es la mejor manera para contribuir a la comunidad, existen dos opciones, la primera es descargar el proyecto directamente desde la pagina principal y la segunda es utilizar un software de control de versiones. Utilizar un sistema de control de versiones permite descargar y subir las modificaciones de forma más rápida y mejor documentada, lo que es de vital importancia para el correcto funcionamiento de la comunidad.

Un segundo trabajo de investigacion que se reviso fue el de (Carrasco, 2016) quien realizo el "Analisis de la comunicacion en comunidades open source", en esta investigacion se hace enfasis en el proceso de comunicación de los usuarios en las comunidades open source, en como los

usuarios tienen su propia importancia asi no sean desarrolladores, ya que se espera de estos una variedad de tareas que permiten enriquecer el proyecto en conjunto como son la documentación (guías de inicio, tutoriales, artículos), el reporte de errores (a través de listas de correos o de gestores de defectos) o simplemente la retroalimentación a través de opiniones.

Esta investigacion se relaciona con el trabajo presente debido a que se ha presentado la situación de los proyectos de software libre y open source en cuanto a estructura comunicativa y se ha mostrado la influencia de GitHub como herramienta colaborativa de desarrollo y debate.

Por ultimo un tercer proyecto revisado fue el de (Tania E, 2012) quien realizo “Software Libre y abierto: comunidades y redes de producción digital de bienes comunes”, donde se analizo el movimiento en crecimiento de open source, asi como el propio desconocimiento de este movimiento en algnunos paises, observando la cooperacion voluntaria de usuarios y desarrolladores en los proyectos, ademas aportando varios datos estadisticos sobre el uso de software privativo que se utiliza por el desconocimiento de herramientas libres por parte de los usuarios.

Esta investigacion se relaciona con el trabajo presente ya que ademas de aportar datos estadisticos sobre el uso de software libre en diferentes paises, tambien habla sobre los procesos de contribucion en las comunidades open source, presentando ademas algunas herramientas de control de versiones primordiales en estas comunidades.

2.2 Software Libre

En 1983, Richard Stallman, siendo programador del Laboratorio de Inteligencia Artificial del MIT, inició el Proyecto GNU para escribir un sistema operativo completo y libre de las restricciones sobre el uso de su código fuente. Esto fue motivado debido a que el desarrollo del

software ya no era una construcción colaborativa realizado por las universidades, sino que era desarrollado por las nacientes empresas de software, las cuáles no permitían ni el acceso, ni la modificación del código fuente.

El suceso que colmó la paciencia de Stallman, fue el fallo de los drivers de una impresora, los cuales él podía arreglar porque tenía el conocimiento, pero no se lo permitieron porque los términos de la licencia lo impedía. Otra probable inspiración para desarrollar el proyecto, fue un desacuerdo con Symbolics Inc sobre el acceso a las actualizaciones de un software que esta empresa desarrolló a partir de un código fuente desarrollado por el MIT. (Stallman, 2004)

Software Libre es aquel que respeta la libertad de todos los usuarios que adquirieron el producto para ser usado, copiado, estudiado, modificado, y redistribuido libremente de varias formas. Es muy importante aclarar que el Software Libre establece muchas libertades pero no es necesariamente gratuito.

- La libertad de usar el programa, con cualquier propósito.
- La libertad de estudiar cómo funciona el programa y modificarlo, adaptándolo a tus necesidades.
- La libertad de distribuir copias del programa, con lo cual puedes ayudar a tu prójimo.
- La libertad de mejorar el programa y hacer públicas esas mejoras a los demás, de modo que toda la comunidad se beneficie.

Existen diferentes licencias de software libre. Una licencia es aquella autorización formal con carácter contractual que un autor de un software da a un interesado para ejercer actos de explotación legales.

- Licencias GNU General Public Licence (GNU GPL): La adopción de esta licencia garantiza a los usuarios finales la libertad de usar, estudiar, compartir (copiar) y modificar el software. Su propósito es declarar que el software cubierto por esta licencia es software libre y protegerlo de intentos de apropiación que restrinjan esas libertades a los usuarios.
- Licencias AGPL (Affero General Public License): Es integramente una GNU GPL con una clausula nueva que añade la obligacion de distribuir el software si este se ejecuta para ofrecer servicios a traves de una red de ordenadores.
- Licencias BSD (Berkeley Software Distribution): Mantiene la protección de copyright únicamente para la renuncia de garantía y para requerir la adecuada atribución de la autoría en trabajos derivados, pero permite la libre redistribución y modificación, incluso si dichos trabajos tienen propietario. Puede argumentarse que esta licencia asegura "verdadero" software libre, en el sentido que el usuario tiene libertad ilimitada con respecto al software, y que puede decidir incluso redistribuirlo como no libre.
- Licencias MPL (Mozilla Public License): La MPL es Software Libre y promueve eficazmente la colaboración evitando el efecto "viral" de la GPL (si usas código licenciado GPL, tu desarrollo final tiene que estar licenciado GPL). No obstante la MPL no es tan excesivamente permisiva como las licencias tipo BSD. Estas licencias son denominadas de copyleft débil.

2.3 Open source

En 1997, Eric Ratmond publicó La Catedral y el Bazar, un análisis reflexivo de la comunidad hacker y los principios del software libre. El documento recibido una atención significativa por parte de la comunidad de programadores de todo el mundo y fue un factor clave para que

Netscape Communications Corporation adoptará al modelo de desarrollo comunitario del software libre. Este código fue la base del navegador Mozilla Firefox y del cliente de correo Thunderbird.

Sin embargo, Raymond y otros llegaron a la conclusión de que el activismo social de la FSF no era atractivo para las compañías como Netscape y buscaron una manera de adecuar el movimiento del software libre al mundo de los negocios. Fue así como, en una sesión de trabajo para diseñar la estrategia de desarrollo de Netscape en Palo Alto, California, Christine Peterson sugirió el uso de la expresión "Código Abierto" para liberarse de las connotaciones ideológicas y de confrontación con el término "Software Libre". En 1998 se crea la Open Source Initiative para promover y respaldar las iniciativas empresariales que quisieran utilizar este nuevo modelo de desarrollo. (Raymond, 2005)

Open Source no se centran tanto en el hecho que los programas derivados mantengan las características, sino en fomentar la apertura del código que utilizan los programas para que todos puedan colaborar y beneficiarse.

Open Source tiene 10 requisitos para que cualquier proyecto pueda considerarse codigo abierto:

- Codigo fuente: A la hora de publicar un programa tiene que incluirse su codigo fuente íntegro o permitir acceder libremente a el.
- Libre redistribución: La licencia del software no debe impedir que este sea regalado o vendido libremente como parte de una distribución mayor que contenga programas de diferentes fuentes. Tampoco debe exigir un pago por hacerlo.

- Trabajos derivados: Las licencias deben permitir modificaciones y trabajos derivados, y debe permitir que estos se distribuyan bajo los mismos términos que el software original.
- Integridad del código fuente del autor: Se puede impedir la distribución de modificaciones únicamente si se permite la distribución de tales como parches. También se puede requerir que trabajos derivados cambien de nombre o número de versión.
- Sin discriminación de personas o grupos: No se puede discriminar a ninguna persona o grupo a la hora de acceder a un programa o su código.
- Sin discriminación de áreas de iniciativa: Tampoco le se puede restringir su acceso a ninguna iniciativa. Las empresas o grupos de investigación tienen tanto derecho como el resto a utilizar el software.
- Distribución de la licencia: Los derechos asociados en las licencias de los programas deben aplicarse a todos a los que lo redistribuyan sin necesidad de pedir una licencia adicional.
- La licencia no debe ser específica de un producto: Un programa no puede licenciarse únicamente como parte de un software mayor. Podrá ser extraído y utilizado libremente y con todos los derechos en otras soluciones.
- La licencia no debe restringir otro software: El hecho de que un proyecto sea de código abierto no puede obligar a que los programas en los que se incluye sean también de código abierto.
- La licencia debe ser tecnológicamente neutral: Ninguna disposición de la licencia puede basarse en la tecnología o un estilo de interfaz, con lo que, por ejemplo, no se debe requerir su aceptación mediante gestos explícitos como clicks de ratón.

2.4 Diferencia entre Software Libre y Open Source

En 1998, algunos dentro de la comunidad del software libre empezaron a usar el término software open source en lugar de software libre para describir lo que hacían. El término open source se asoció rápidamente con un enfoque distinto, una filosofía distinta, e incluso diferentes criterios para decidir que licencias son aceptables. El movimiento de software libre y el movimiento open source son hoy en día movimientos separados con diferentes puntos de vista y objetivos.

La diferencia fundamental entre los dos movimientos está en sus valores, en su visión del mundo. Para el movimiento open source, la cuestión de si el software debe ser de fuente abierta es una cuestión práctica, no ética. El open source es un método de desarrollo; el software libre es un movimiento social. Para el movimiento open source, el software no libre es una solución ineficiente. Para el movimiento de software libre, el software no libre es un problema social y el software libre es la solución. (Stallman, 2004)

2.5 Comunidades open source

Las comunidades open source son un ejemplo de ambientes de desarrollo colaborativo y distribuido, un ambiente colaborativo es aquel que soporta un grupo de usuarios que participan en actividades u objetivos comunes y que proveen una interfaz a un entorno compartido por ellos, mientras que un ambiente distribuido permite la existencia de varios usuarios, usualmente asociados mediante un sistema de conexión de red, estos sistemas de trabajo distribuido soportan el desarrollo de aplicaciones y servicios teniendo en cuenta la autonomia que tiene cada integrante y que se comunican por mensajes sobre una red de comunicaciones. (Paletta, 2009)

Partiendo de que una comunidad es un grupo de personas que interactuan entre si para lograr un fin comun, llevado al ambito de la informatica, los grupos de personas buscan conocimiento mutuo, satisfacer necesidades de informacion y de solucion de problemas, desempeñando roles especificos.

Las comunidades Open Source se caracterizan por ser de libre acceso a las personas, para que estas puedan observar, descargar contenido y realizar aportes a esta misma con el fin de que todos aquellos interesados en un tema contribuyan al desarrollo de este.

Ademas las comunidades de código abierto pueden innovar de forma mucho más eficaz que los laboratorios tradicionales debido a que todas los interesados pueden aportar en la resolucion de problemas comunes ademas de compartir conocimientos para dar lugar a mejores y mas rapidas innovaciones.

Apartir de esta definicion este proyecto se enfoca en el estudio de las comunidades open source, especificamente en la investigacion de como los usuarios contribuyen en sus proyectos, esto sin necesidad de ser desarrolladores como habla (Carrasco, 2016), cada usuario en un proyecto open source tiene diferentes roles, y en este proyecto son importantes tanto los desarrolladores, como los demas usuarios que aportan a la documentacion, al testing, traducciones, etc. Algunos de los roles dentro de estas comunidades estan detallados acontinuacion. (German, 2006).

Administrador del OSS: Es el individuo o grupo que inicia el proyecto del para el desarrollo del producto software.
Desarrollador: Es el encargado de la codificación de las funcionalidades del desarrollo del producto software.
Escritores de documentación: Este rol tiene a cargo de escribir la documentación del producto software, es decir, este rol documenta cómo instalar, configurar y usar el producto software.
Traductor: Las comunidades Open Source son comunidades distribuidas esto quiere decir que existe la internalización en la participación del proyecto, luego existen diferentes lenguajes.
Testing: Son los participantes que verifican que el software haga lo que debe hacer en diferentes ambientes de hardware y software además son los que hacen reportes de bugs.
Soporte a usuarios: Esta tarea consiste en que usuarios responden inquietudes acerca del software de otros usuarios
Usuarios: Son los individuos que usan el producto software, esta definición incluye a empresas o cualquier otra organización.

Tabla 1: Roles en una Comunidad Open Source

Este proyecto analiza el uso de sistemas de control de versiones como herramienta para la contribucion en las comunidades, GNOME, Django Software Foundation, Alliance for Open

Media, Mozilla Foundation, Apache Software Foundation, FreeBSD Foundation, son algunas de las comunidades que ya tienen sus proyectos libres para el acceso de cualquier usuario que quiera contribuir en la continua evolucion de sus proyectos. Muchas de estas comunidades utilizan Git como el sistema de control de versiones para que los usuarios descarguen el codigo fuente, documentacion y demas secciones de un proyecto, realicen sus respectivos cambios y suban los mismos al proyecto principal, comunidades como Mozilla utilizan forjas como Github debido a sus multiples proyectos en los que es posible participar, y eligen esta forja porque permite una mejor gestion para usuarios y administradores en un proyecto.

2.6 Control de versiones

Para comprender el papel del control de versiones en la ingeniería de software es importante comenzar por recordar que el objetivo principal de esta ingeniería es entregar un producto que satisfaga las necesidades del usuario, de forma eficiente y predecible, es por esto que basados en conceptos del SWEBOK (Bourque & Fairley, 2014), el cual como guia al cuerpo de la ingenieria del software define las areas del conocimiento para que se desarrolle un producto de calidad desde la captura de requerimientos, pasando por la gestion de configuracion, para terminar por el despliegue y mantenimiento a traves de la gestion de la ingenieria del software, se observa que el proceso de control de versiones es importante durante la evolucion de un producto de software, ademas de hablar del control de cambios, este hace referencia tambien a la debida autorizacion de cambios presente en este proyecto.

Aunque en los procesos de ingenieria del software no se tiene una guia definitiva para contruir software de calidad, es posible observar que muchos autores como (Wanumen, 2014) tambien definen unas fases principales: iniciacion, elaboracion, construccion y transicion; y en cada una de estas fases se deben realizar unos procesos: modelado del negocio, requisitos, analisis y

diseño, implementacion, pruebas, despliegue, soporte, gestion de configuracion, gestion del proyecto, entorno. Lo interesante es que la gestion de configuracion esta presente en cada guia para la construccion de software de calidad, y es por eso que es tan importante llevar un control de los estados por los que pasa el software durante su ciclo de vida.

El propósito de la gestión de configuración es establecer y mantener la integridad de cada uno de los productos que se van obteniendo del ciclo de desarrollo de software (Paredes, 2011). La gestion de configuracion representa un elemento clave en el proceso de desarrollo de software ya que proporciona estabilidad a la producción de software, controla el cambio continuo y concurrente que viene con la evolución del producto de software y obliga a implementar estrategias de versionamiento.

Los sistemas de control de versiones permiten la gestión de diversos cambios que se realizan sobre elementos de un producto o configuración del mismo, es utilizado principalmente para controlar el código fuente en un proyecto de desarrollo de software, aunque también funciona para otro tipo de archivos que requieran un seguimiento durante sus respectivos cambios.

Este control puede realizarse manualmente aunque en el proceso podrían generarse errores además de que sería muy tedioso por lo que se han creado algunas herramientas para automatizar esta gestión de archivos y código fuente en un proyecto de desarrollo. Estas herramientas son llamadas Sistemas de Control de Versiones (VCS) las cuales se explicaran más adelante.

Con este tipo de herramientas un usuario o administrador puede regresar un archivo o conjunto de archivos a un estado anterior, esto en el caso de algun fallo, ademas brinda mas funcionalidades como el control de acceso al codigo, ver las actividades de los integrantes del proyecto, en algunos casos permite la autorizacion de cambios. Etc.

2.6.1 Terminologia

Para comprender los VCS y su funcionamiento es necesario entender algunos de los terminos que se manejaran mas adelante en este proyecto. (González, 2010). Tambien se abordan terminos de (Collado, 2004).

- Repositorio: Un repositorio es un lugar donde se almacenan los datos actualizados e historicos de cambios de uno o varios archivos, normalmente este es almacenado en un servidor remoto.
- Modulo: Un modulo es un conjunto de archivos o directorios que forman parte de un proyecto.
- Revision: Una revision es una version o un estado determinado de la informacion que se esta controlando, hay sistemas que identifican estas versiones con un contador, aunque otras generan un codigo sha1.
- Concepto de control de configuracion: un sistema software comprende distintos componentes, que evolucionan individualmente, una configuracion es una combinacion de versiones particulares de los componentes que forman un sistema consistente. Por lo tanto el control de esta configuracion significa administrar las diferentes versiones por las que pasan los componentes de un sistema durante su evolucion.
- Linea base: es la configuracion operativa del sistema software, puede ser vista como la version aprobada.
- Tag: Un tag es una “etiqueta” que se le da a una revision determinada, es utilizada con frecuencia para marcar etapas clave en el desarrollo de un proyecto, por ejemplo, la primera version liberada del mismo proyecto, de esta forma se podrá encontrar esta

version de manera mas simple. En algunos sistemas un tag es considerado una rama en la que los ficheros no evolucionan, estos estan congelados.

- Rama: Una rama es un apuntador movil a cualquier revision que se esté trabajando, por lo que en este momento se tienen dos copias (ramas) que evolucionan de forma independiente siguiendo su propia linea de desarrollo. Con esto se asegura que si se desea crear alguna funcionalidad de forma independiente o si se desea implementar codigo de evaluacion en una rama de prueba, es posible hacerlo sin peligro a dañar la rama principal del proyecto. Al finalizar el desarrollo en la rama se procede a realizar un merge en la rama principal.

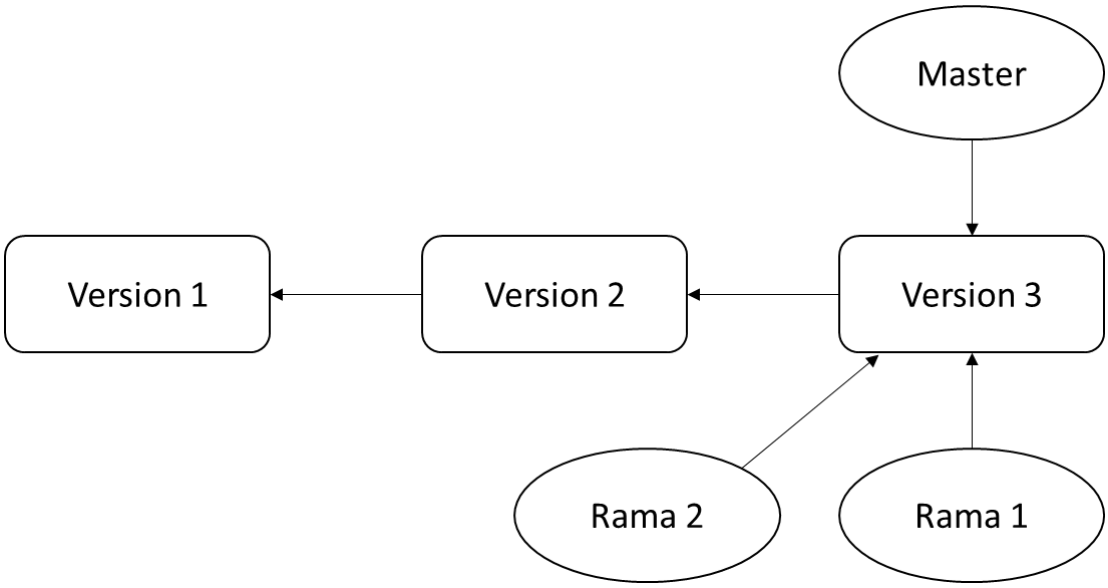


Ilustración 1: Rama

- Merge: Merge es la integracion o fusion para unir dos conjuntos de cambios en un fichero o un conjunto de ficheros. Esto es necesario cuando dos usuarios trabajan análogamente en los ficheros, ademas trabajan en la misma linea, por lo que esto genera un conflicto y

el desarrollador que esta confirmando los cambios debe escoger si combinar las versiones o dejar una version especifica de uno de los dos que trabajan sobre el mismo fichero. Tambien es necesario cuando se ha creado una rama durante el desarrollo, en el momento en el que la rama termine su ciclo de desarrollo, esta debe integrarse o fusionarse con la rama principal (master) y asi unificar el proyecto.

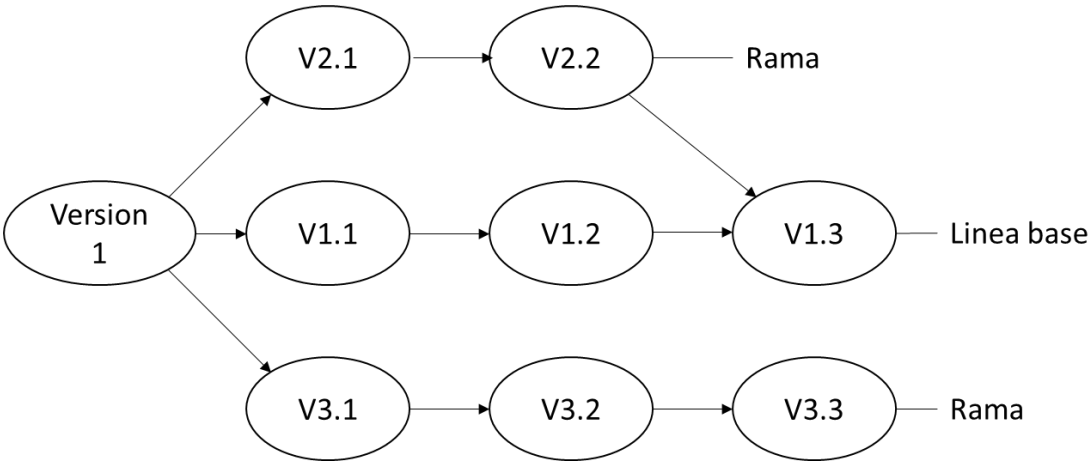


Ilustración 2: Fusion de Ramas

- Checkout: Checkout crea una copia local del repositorio, aunque se puede realizar un checkout de una version especifica, predeterminadamente se realiza de la ultima revision.
- Diff: Diff representa un cambio o una modificacion especifica a un archivo bajo control de versiones.
- Commit: Un commit confirma los cambios realizados hechos en la copia local y la integra en los repositorios, en algunos sistemas este se realiza completamente de forma local y para confirmar los cambios a un servidor remoto es necesario realizar otra operación mas.

- Update: Un update integra los cambios que han sido hechos en el repositorio en la copia local de trabajo.

2.6.2 Funcionamiento de los VCS

Los VCS son aplicaciones que guardan en repositorios las versiones de un software generadas en el transcurso de su desarrollo, evolución y fin del proyecto. En (Gutierrez, 2011) se describen estos sistemas como los encargados de administrar los distintos estados por los que pasa una aplicación durante su etapa de desarrollo, permitiendo guardar un historial con los cambios realizados entre las versiones de esta. Esto es de gran ayuda para los desarrolladores porque todas las versiones por la que pasa un archivo está disponible para todos los participantes del proyecto, añadiendo además, fecha, hora y el nombre del desarrollador responsable del cambio para así los administradores del proyecto tener una visión completa de cómo va su proyecto.

2.6.3 Clasificación de los VCS

Los VCS se clasifican en centralizados y distribuidos, en la actualidad existen exponentes importantes de cada tipo. CVS, Subversion (en el caso de los VCS centralizados); y Git, Mercurial (en el caso de los VCS distribuidos).

Los VCS centralizados son aquellos en los que se cuenta con un servidor central donde se almacenan las diferentes versiones de un archivo, los desarrolladores las toman, las modifican y las guardan de nuevo en este servidor central, a menudo con autorización requerida.

Los VCS distribuidos no necesitan un servidor central para almacenar la información, se puede almacenar la información de forma local, accediendo a ella, modificándola y guardando diferentes versiones sin necesidad de un servidor central.

2.6.3.1 Sistemas de control de versiones centralizados

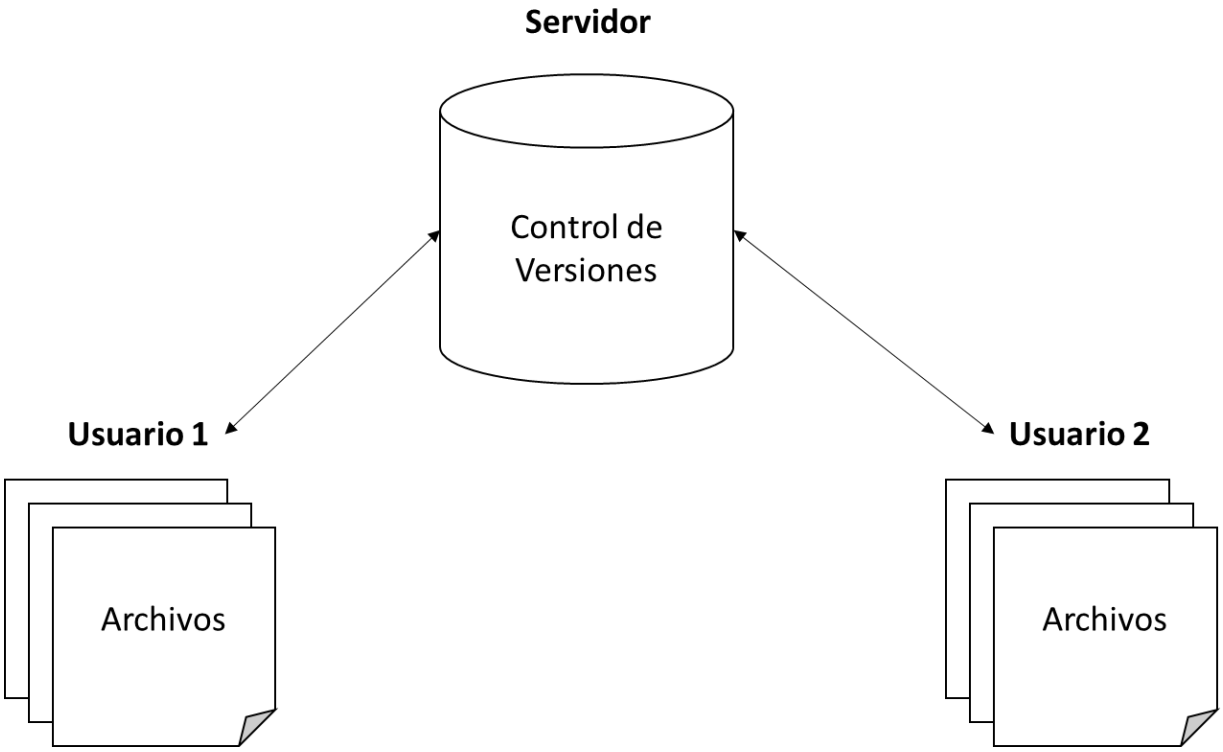


Ilustración 3: VCS centralizados

En los VCS centralizados las versiones de los archivos de un proyecto están almacenadas en un servidor central, esto hace que los desarrolladores deban solicitar una copia local de los archivos y que cuando realicen los cambios respectivos actualicen nuevamente el repositorio par que este guarde esta nueva versión. (González, 2010)

Este tipo de VCS tienen algunas ventajas:

- Es posible conocer (hasta cierto punto) en que están trabajando los demás desarrolladores.
- Los administradores tienen control total de que puede hacer cada miembro del equipo
- Ofrece más facilidad a la hora de administrar.

2.6.3.2 Sistemas de control de versiones distribuidos

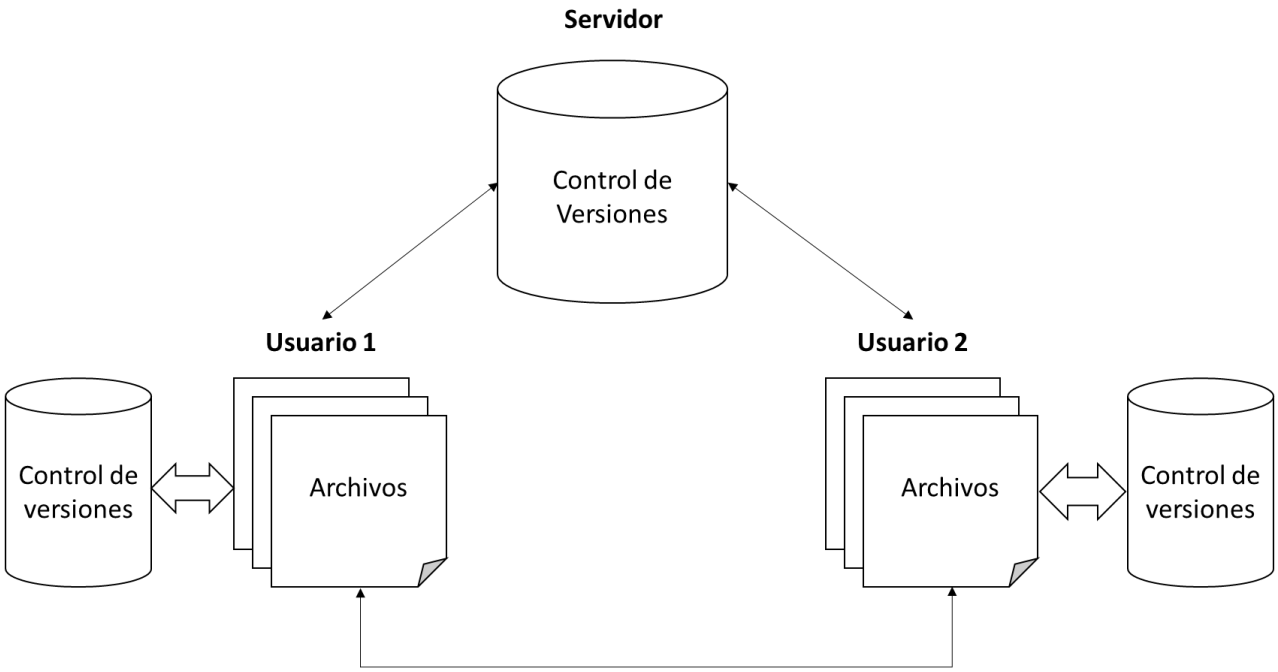


Ilustración 4: VCS Distribuidos

En los VCS distribuidos cada desarrollador tiene una copia del repositorio completo en su computadora, generando así un repositorio local del proyecto. De esta forma cada desarrollador puede trabajar paralela e independientemente debido a que cada cambio generado en el proyecto se guardara de manera local. Luego en la etapa en la que se requiera unir los repositorios locales de cada desarrollador, los VCS realizaran una sincronización de copias generando una nueva versión del proyecto conteniendo todos los cambios de cada desarrollador. (González, 2010)

Este tipo de VCS tienen algunas ventajas:

- Depende menos de una conexión a internet por lo que no se debe estar conectado en todo momento.

- Aunque se caiga el repositorio remoto los desarrolladores pueden continuar trabajando localmente
- Los desarrolladores pueden decidir si alguna información es irrelevante para el proyecto por lo que pueden evitar que esta sea accesible de forma publica
- El servidor remoto requiere menos recursos que uno centralizado.

2.7 Herramientas para el Control de Versiones (VCS)

El control de versiones en un proyecto consiste en guardar copias de seguridad de archivos modificados para poder restaurar esta versión si algo no sale como debería; este procedimiento para gestionar las versiones de un archivo se puede realizar de forma manual localmente, aunque este sería un proceso muy tedioso y propenso a errores.

Para evitar este inconveniente existen los llamados VCS o sistema de control de versiones que permiten de forma automática realizar las actividades ya descritas y que permiten tener un control total de cada proyecto durante el comienzo de su desarrollo hasta el lanzamiento del mismo.

Algunas de estas herramientas son.(Tello-leal, 2012)

2.7.1 Concurrent Versions System (CVS)

Esta herramienta fue la más utilizada por un tiempo en ambientes open source, cuenta con la arquitectura cliente-servidor para que los usuarios accedan al repositorio y desde allí hagan una copia del proyecto, favoreciendo el trabajo colaborativo.

Este VCS requiere de conexión permanente a internet para que los desarrolladores hagan una copia del proyecto (checkout), realicen sus respectivos cambios y suban estos al repositorio (commit), y si en alguna ocasión se genera un conflicto entre las lineas modificadas por dos

desarrolladores, el CVS dara un mensaje de error (conflict) para que los desarrolladores involucrados resuelvan este inconveniente. Una vez confirmados los cambios se generara un (log-message) con el numero de la version de los archivos del repositorio, asi como la fecha y los nombres de los desarrolladores que han realizado cambios.

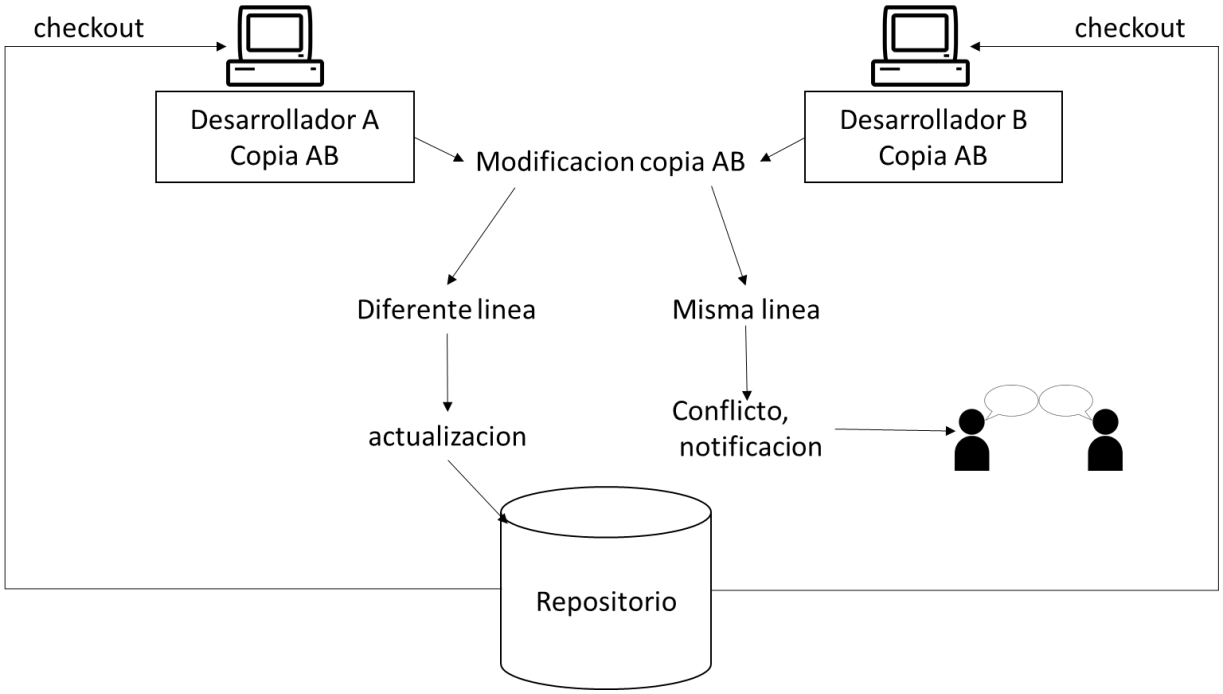


Ilustración 5: Funcionamiento CVS

2.7.2 Subversion (SVN)

Subversion es otra herramienta para el control de versiones dentro de la categoría de VCS centralizados, a diferencia de CVS este permite administrar los cambios tanto en archivos como en directorios, facilitando la recuperación de archivos o brindando la posibilidad de observar el historial de cambios de un archivo o directorio.

Al ser centralizado subversión funciona con copias locales que los desarrolladores hacen desde el repositorio hasta el ordenador local donde trabajan, los conocidos (checkout) y (commit)

forman parte de esta herramienta así como de la anterior, incorporando una nueva función los (merge) que permite fusionar las diferentes copias privadas de los desarrolladores, si ocurriera algún error, son los mismos desarrolladores quienes observan que cambios dejar con respecto a demás para solucionar el problema.

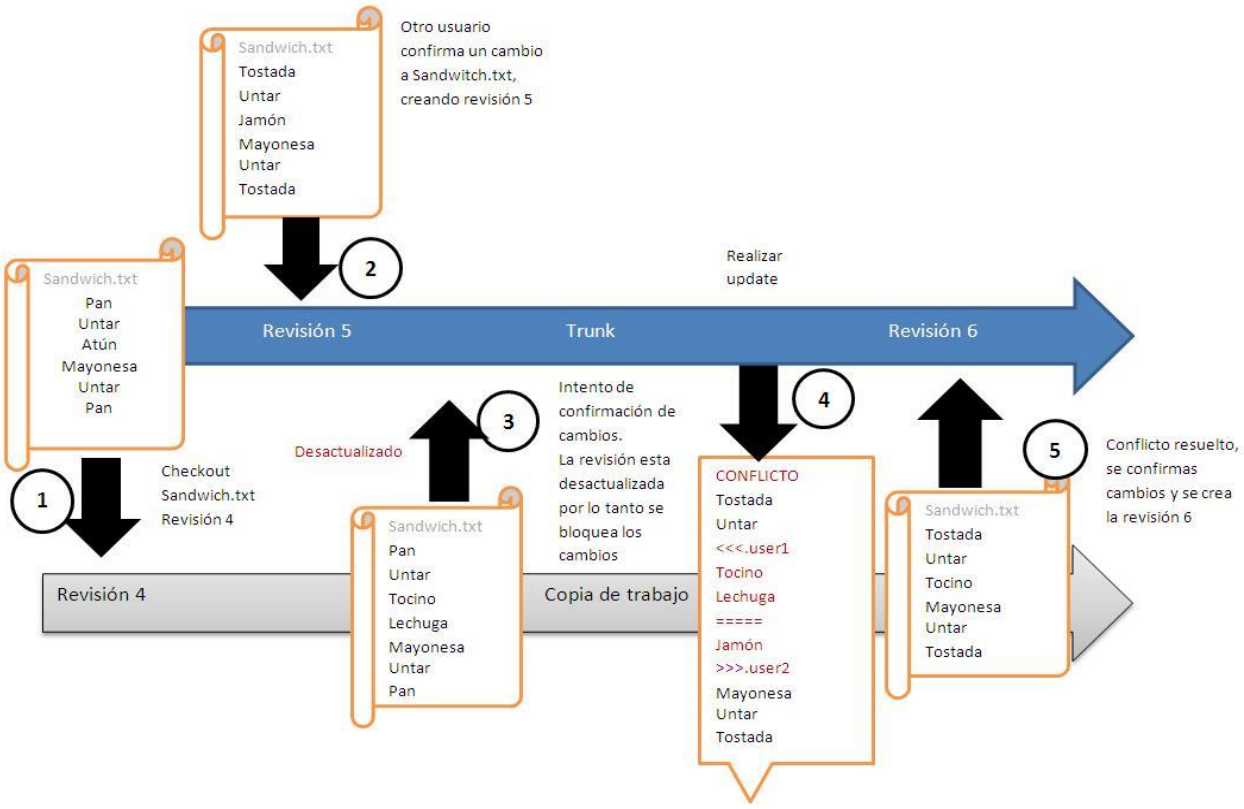


Ilustración 6:Funcionamiento SVN

Subversion mejoro muchos aspectos en los que su predecesor fallaba:

- En subversión no solo los archivos tienen un control de cambios sino también los directorios cuentan con este versionado que permite observar cuando archivos son agregados o eliminados de un directorio.

- Subversión permite también renombrar archivos y directorios, cosa que su predecesor no tenía.
- A diferencia de su predecesor, subversión permite agregar, eliminar, copiar y renombrar archivos y directorios; cada archivo nuevo inicia con un historial de versiones limpio.
- Incorpora un mejor sistema de resolución de conflictos.

2.7.3 Bazaar

Bazaar es un VCS un poco especial debido a que puede combinar alguna caracteristica de un VCS centralizado y de un VCS distribuido, bazaar soporta distintas formas de trabajar debido a que esta depende de diversos factores, como el numero de integrantes en el proyecto, por esto, bazaar cuenta con algunas formas de trabajo.

- Solitaria: Bazaar permite gestionar nuestros archivos sin la necesidad de levantar ningun servidor, esta forma de trabajo apenas necesita administracion, asi que una sola persona puede comenzar a crear un proyecto, controlar los cambios y ver el historial de estos archivos.
- En Pareja: Este tipo de trabajo se puede dar cuando dos personas necesitan trabajar en un proyecto juntas y compartir sus cambios, por lo que una segunda persona puede crear una rama (con historial incluido) de lo que la primera persona tiene y pueden fusionar sus ramas cuando lo deseen.

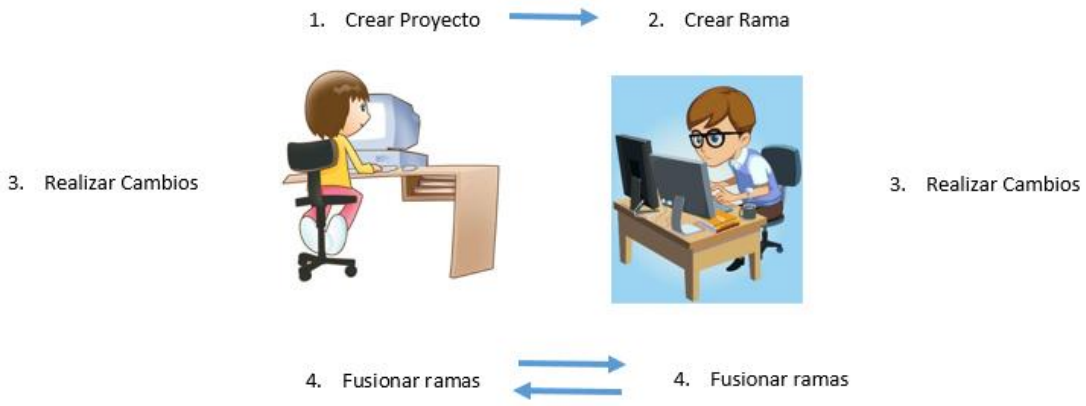


Ilustración 7: Bazaar, funcionamiento en pareja

- Centralizado: En esta forma de trabajo es necesario un servidor centralizado en el cual alojar todo el proyecto en un repositorio. Esta es la misma forma como se trabaja en CVS o Subversion, todo el equipo de desarrollo trabajan en las mismas ramas y realizan sus updates para mantener sus archivos actualizados y commits para registrar los cambios.

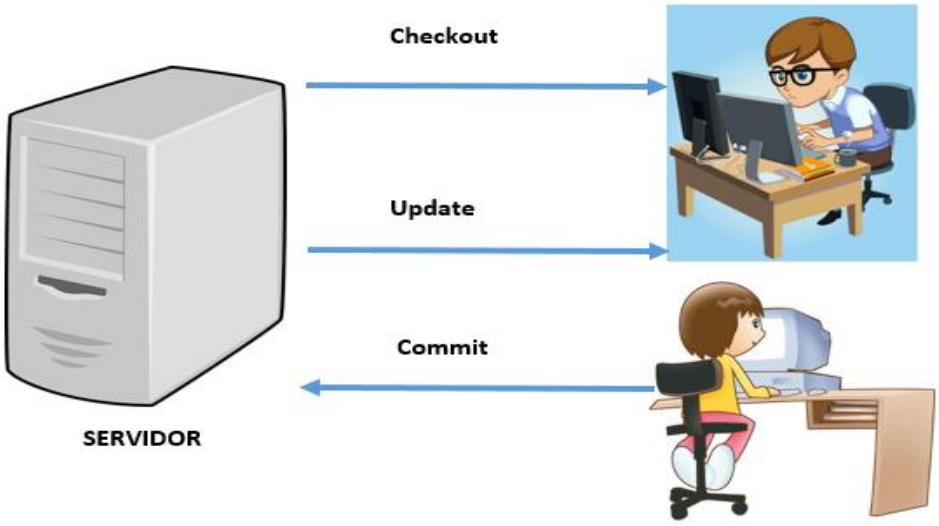


Ilustración 8: Bazaar, funcionamiento centralizado

- Descentralizado con supervisor humano: En esta modalidad de trabajo, los desarrolladores realizan sus respectivos cambios en sus propias ramas de trabajo, pero estos no pueden confirmar ningun cambio a la rama principal del proyecto debido a que solo tienen permisos de lectura en esta. Para confirmar cambios en la linea principal del proyecto es necesario que el supervisor realice dicha confirmacion de los cambios.

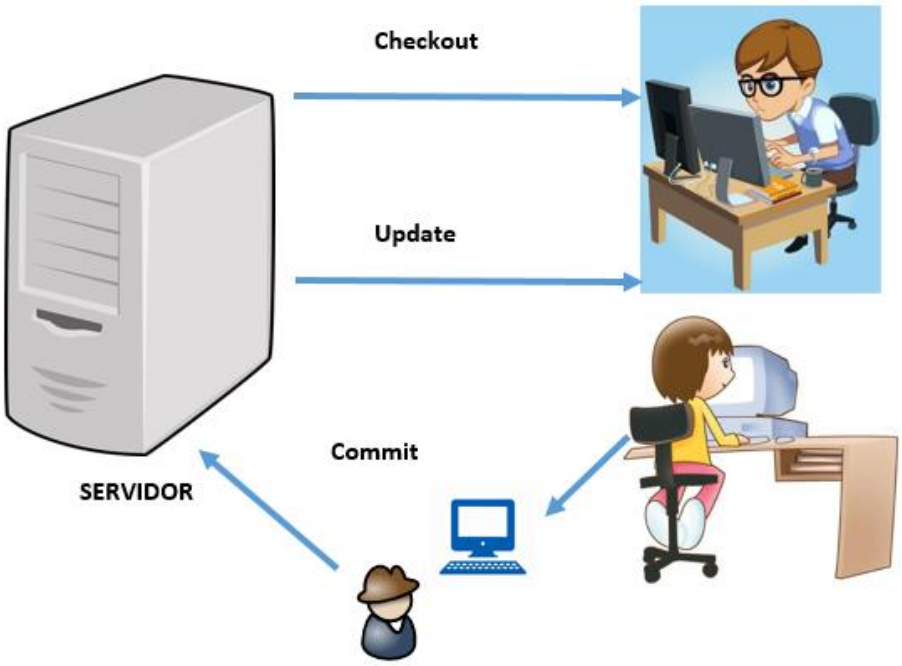


Ilustración 9:Bazaar, funcionamiento descentralizado con supervisor

2.7.4 Git

Git es uno de los VCS mas usados en la actualidad, es el preferido debido a su gran utilidad y su gran usabilidad, siendo utilizado en proyectos muy grandes, Git al ser un VCS distribuido cuenta con muchos beneficios con respecto a los VCS centralizados.

Algunas de las ventajas con las que cuenta Git al ser distribuido es que en cada proyecto por grande que sea, y la cantidad de desarrolladores que tenga, cada uno del personal involucrado en

el proyecto puede descargar su copia local desde el repositorio y una vez hecho esto no es necesaria la conexión a internet para poder trabajar, cosa que con los VCS centralizados no era posible.

Git esta compuesto de una estructura de tres secciones, el area de trabajo, el area de preparacion, el directorio de Git. Ademas es posible agregar un cuarto si se quiere trabajar de forma remota.

- Area de trabajo: Esta seccion contiene los archivos de la version actual sobre los que se realizaran cambios.
- Area de preparacion: En esta seccion se almacena la informacion de los cambios que se van a enviar en la proxima confirmacion.
- Directorio de Git: Esta seccion guarda los objetos que mantienen el historial con los cambios que se han producido en el proyecto.

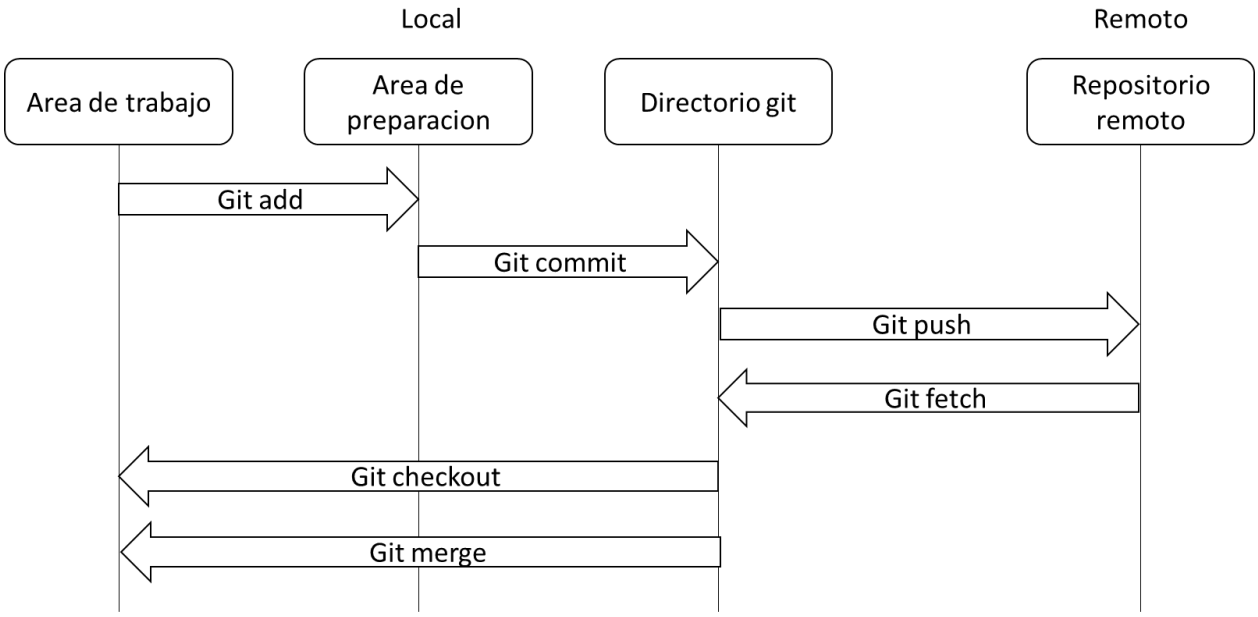


Ilustración 10: Funcionamiento Git

Como se observa en la Ilustración 10: Funcionamiento Git; git utiliza los comandos “git add” y “git commit” para trabajar de forma local y asi confirmar los cambios en nuestro repositorio local, sin embargo, para actualizar y confirmar cambios en un servidor remoto es necesario explicar algunos comandos que utiliza Git.

- Git push: git push confirma los cambios realizados a nivel remoto, es decir, envia los cambios realizados en tu copia local a un repositorio remoto.
- Git fetch: git fetch actualiza los cambios desde el repositorio remoto pero los añade a una rama diferente a la principal (master) para que cuando quieras integres (merge) los cambios con tu rama de trabajo.
- Git pull: git pull permite actualizar los cambios desde el repositorio remoto añadiendo todo a tu rama de trabajo, es decir, combina el uso de git fetch y git merge.

Git tambien permite crear las ramas que cada desarrollador necesite, y cada rama creada solamente afecta o esta almacenada en nuestra propia copia local, por lo que los cambios solamente afectaran al desarrollador que realice los cambios, tambien cuenta con una excelente fusion de ramas.

Es importante destacar que a diferencia con otros VCS el HEAD de Git funciona diferente, este funciona como un apuntador a una rama en la que se esta trabajando, por ejemplo, si se esta trabajando sobre una rama llamada pruebas, el HEAD apuntara a esta.

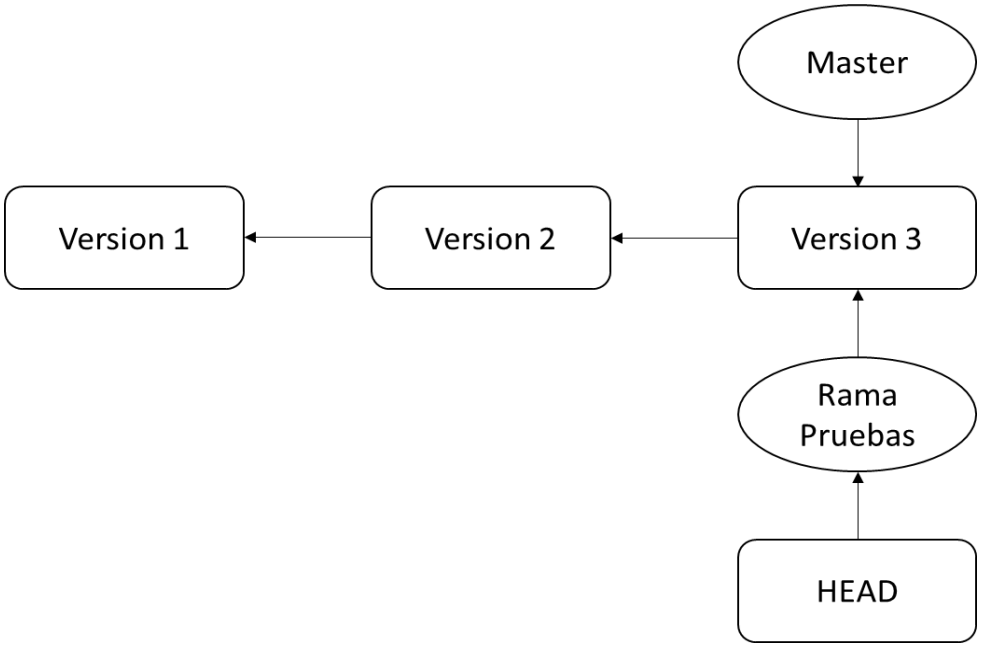


Ilustración 11:HEAD en Git

Todo esto hace de Git el VCS preferido por los desarrolladores, ademas de ser muy veloz en cuanto a sus actividades, aporta cosas que facilitan el desarrollo y el seguimiento de un proyecto grande con muchas personas que infieren en su evolucion.

Para administrar los diferentes usuarios que intervienen en un proyecto es necesario hablar de que gitlab que es un servicio web de control de versiones y desarrollo de software colaborativo basado en Git, ademas de ser un gestor de repositorios, gitlab tambien ofrece alojamiento de wikis, un sistema de seguimiento de usuarios, seguimiento de incidencias o tareas, revision de codigo, diferentes tipos de accesos o permisos para los usuarios de un proyecto, gestion de equipos por grupos, capacidad para importar repositorios entre otras.

Los diferentes roles que maneja gitlab seran detallados en la siguiente tabla, estos permisos son en cuanto al proyecto.

Acción	Invitado	Reportero	Desarrollador	Maestro	Propietario
Crear nuevo asunto o problema (issue)	✓	✓	✓	✓	✓
Crear asunto, problema confidencial	✓	✓	✓	✓	✓
Ver asuntos, problemas confidenciales	✓	✓	✓	✓	✓
Dejar comentarios en proyecto	✓	✓	✓	✓	✓
Ver lista de actividades del proyecto	✓	✓	✓	✓	✓
Ver registro de actividades	✓	✓	✓	✓	✓
Explorar y descargar artefactos[1] (Artifacts) de trabajo	✓	✓	✓	✓	✓
Ver paginas wiki del proyecto	✓	✓	✓	✓	✓
Descargar el proyecto desde servidor remoto (Pull)		✓	✓	✓	✓
Descargar proyecto		✓	✓	✓	✓

[1] Artefactos: los artefactos (artifacts) se utilizan para especificar una lista de archivos y directorios que se deberian añadir al trabajo despues de su finalizacion.

Crear fragmentos[1] (Snippets) de codigo		✓	✓	✓	✓
Administrar seguimiento de asuntos o problemas		✓	✓	✓	✓
Administrar etiquetas (Tags)		✓	✓	✓	✓
Ver un estado de commit		✓	✓	✓	✓
Ver registro de contenedores[2] Dockers[3].		✓	✓	✓	✓
Ver entornos[4]		✓	✓	✓	✓
Crear nuevos entornos			✓	✓	✓
Usar entorno en terminal				✓	✓
Detener entornos			✓	✓	✓
Ver lista de solicitudes de fusion (merge request).		✓	✓	✓	✓
Administrar solicitudes de fusion			✓	✓	✓
Crear solicitud de fusion			✓	✓	✓

[1] Fragmentos: los fragmentos (snippets) son pequeñas partes de codigo o texto, que se usan semi regularmente dentro del proyecto, pero que no forman parte del control de codigo fuente, por ejemplo, un archivo de configuracion.

[2] Registro de contenedores: El registro de contenedores (Container registry) hace referencia a la integracion continua de gitlab, es un registro de imágenes docker, que contienen lo necesario para ejecutar una aplicación.

[3] Dockers: Docker es una herramienta open-source que permite realizar una 'virtualización ligera', con la que poder empaquetar entornos y aplicaciones que posteriormente podremos desplegar en cualquier sistema que disponga de esta tecnología.

[4] Entornos: un entorno en gitlab permite probar y desplegar el codigo en un ambiente de pruebas, los entornos son como etiquetas (tags) en la integracion continua de gitlab.

Crear ramas (branch)			✓	✓	✓
Realizar push a ramas no protegidas			✓	✓	✓
Forzar push a ramas no protegidas			✓	✓	✓
Remover ramas no protegidas			✓	✓	✓
Añadir etiquetas (tags)			✓	✓	✓
Redactar una wiki			✓	✓	✓
Cancelar y reintentar actividades			✓	✓	✓
Crear o actualizar estados de confimacion(commits)			✓	✓	✓
Actualizar registro de contenedores			✓	✓	✓
Eliminar una imagen docker del registro de un contenedor			✓	✓	✓
Crear hitos[1]				✓	✓
Añadir nuevos miembros de equipo				✓	✓
Realizar push a ramas				✓	✓

[1] Hito: un hito es un punto clave durante el desarrollo de un proyecto, no es un entregable, es un logro importante en el proyecto.

protegidas					
Habilitar/deshabilitar proteccion de ramas				✓	✓
Habilitar/deshabilitar push en ramas protegidas para desarrolladores				✓	✓
Habilitar/deshabilitar proteccion de tags				✓	✓
Reescribir/remover tags				✓	✓
Editar proyecto				✓	✓
Añadir claves de implementacion al proyecto				✓	✓
Configurar hooks[1] en proyecto				✓	✓
Administrar disparadores de actividades				✓	✓
Administrar variables				✓	✓
Administra paginas				✓	✓
Administrar dominios y certificados de paginas				✓	✓
Cambiar el nivel de					✓

[1] Los hooks de git son scripts que se pueden definir en nuestro repositorio para que se disparen cuando git realiza una acción.

visibilidad					
Transferir proyecto					✓
Remover proyecto					✓
Remover paginas					✓

Tabla 2:Permisos Gitlab

2.7.5 Mercurial

Mercurial es otro VCS distribuido, por lo tanto tambien permite copiar un repositorio, conteniendo toda la historia del desarrollo y evolucion del proyecto, esta copia del repositorio se realiza de forma local para que esta funcione sin necesidad de requerir un acceso a la red o a un servidor remoto.

Mercurial tiene bastantes similitudes con Git, ya que es otro VCS muy completo para controlar la evolucion de un proyecto en desarrollo, la gestion de ramas y la fusion de las mismas tambien estan disponibles con Mercurial.

Las diferencias que tiene Git con Mercurial se basan en que la curva de aprendizaje es mas grande en el caso de Git, Mercurial es mucho mas sencillo de usar, simplifica muchas opciones de Git por lo que es una buena opcion para los desarrolladores que vienen de SVN Subversion.

2.8 Comparacion de algunos VCS

2.8.1 Ventajas y Desventajas de algunos VCS

A continuación se observaran algunas ventajas y desventajas de algunos de los sistemas más conocidos para el control de versiones en la actualidad. (M¥rka, 2006)

VCS	Ventajas	Desventajas
CVS	• Utiliza arquitectura cliente-servidor • Los usuarios realizan una copia completa del proyecto • Funciona en cualquier SO • Capacidad de mantener distintas ramas en un proyecto	• No soporta refactorizacion • Limitada a UTF-8 Unicode • No soporta eliminar ni renombrar directorios
SVN (Subversion)	• Se continua con la historia de archivos siendo renombrados • Modificaciones atomicas • Las operaciones de creacion de ramas y tags • Permite el bloqueo de archivos	• Al momento de renombrar un archivo este lo realiza como una operación de borrado y agregado de un archivo con diferente nombre • Dificultad al aplicar parches a las ramas
Git	• Versatil y configurable • Distribuido, por lo que no se necesita de conexión a internet • Es muy veloz debido a que gran parte de las actividades se realizan de forma local	• Curva de aprendizaje para los que vienen de SVN • Muchos comandos

Bazaar	• Capacidad de adaptacion para cualquier tipo de proyecto • Facil de utilizar • Tiene una comunidad que realiza aportes para continuar mejorando	• Se trabaja con acceso remoto permanentemente • Consume ancho de banda en comparacion con los demas • Estadisticamente es lento
Mercurial	• Muy veloz • Mas simple que Git • No hay tantas funciones que aprender • Tiene una excelente documentacion	• Solo se utiliza en proyectos grandes

Tabla 3: Ventajas y Desventajas de algunos VCS

Despues de revisar algunas ventajas y desventajas que decantan por el uso de algun VCS especifico se revisaran algunas otras caracteristicas de estos. (Scv, 2014)

2.8.2 Información general

VCS	Estado Desarrollo	Modelo repositorio	Modelo de concurrencia	Licencia	costo
CVS	Se sostiene pero no se han añadido nuevas funciones	Cliente-servidor	Fusion	GNU	Gratis
Subversion	Activo	Cliente-servidor	Fusion y bloqueo	Apache	Gratis
Bazaar	Activo	Cliente-servidor y distribuido	Fusion	GNU	Gratis
Git	Activo	Distribuido	Fusion	GNU	Gratis
Mercurial	Activo	distribuido	Fusion	GNU	Gratis

Tabla 4: Informacion General VCS

2.8.3 Información Técnica

VCS	Lenguaje de programacion	Metodo de almacenamiento	Alcance de cambios	Id de revision
CVS	C	Changeset	Fichero	Numeros
Subversion	C	Changeset – Snapshot	Arbol	Numeros
Bazaar	Python 2, C	Snapshot	Arbol	Pseudorandom
Git	C, Perl	Snapshot	Arbol	SHA-1 hashes
Mercurial	Python, C	Changeset	Arbol	Numeros, SHA-1 hashes

Tabla 5: Informacion Tecnica VCS

2.8.4 Comandos Basicos

VCS / Comandos	CVS	SVN Subversion	Bazaar	Git	Mercurial
Init	✓	✓	✓	✓	✓
Clone		✓	✓	✓	✓
Pull		✓	✓	✓	✓
Push		✓	✓	✓	✓
Branch			✓	✓	✓
Checkout	✓	✓	✓	✓	✓
Update	✓	✓	✓	✓	✓

Lock	✓	✓		✓	✓
Add	✓	✓	✓	✓	✓
Remove	✓	✓	✓	✓	✓
Move		✓	✓	✓	✓
Copy		✓		✓	✓
Merge	✓	✓	✓	✓	✓
Commit	✓	✓	✓	✓	✓
Revert	✓		✓	✓	✓

Tabla 6: Comandos basicos

3 Procedimiento

Acontinuacion se describira el procedimiento a seguir y posteriormente se describira la validacion correspondiente.

3.1 Descripcion del procedimiento

Para realizar el procedimiento se observaron y analizaron diferentes sistemas para el control de versiones (VCS), eligiendo Git como la herramienta a utilizar en este proyecto.

Este procedimiento se ilustrara por medio de un diagrama de flujo que clarificara los pasos a seguir para la correcta configuracion de un sistema de control de versiones en nuestro proyecto.

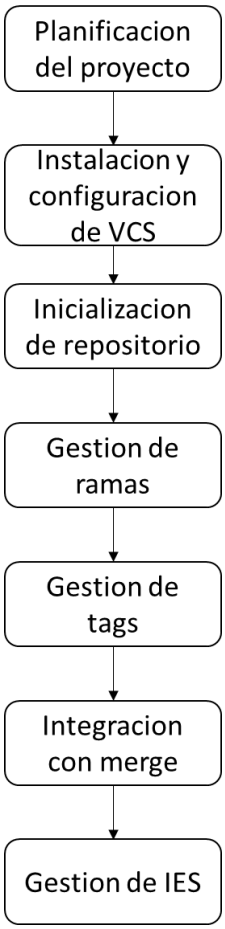


Ilustración 12: Procedimiento propuesto

3.1.1 Planificacion del proyecto

En esta etapa del proceso se deben determinar aspectos importantes como es la estructura propia del proyecto, definicion de roles en el mismo, los modulos y submodulos que tendra el proyecto.

Se recomienda la estructura del proyecto de la siguiente forma.

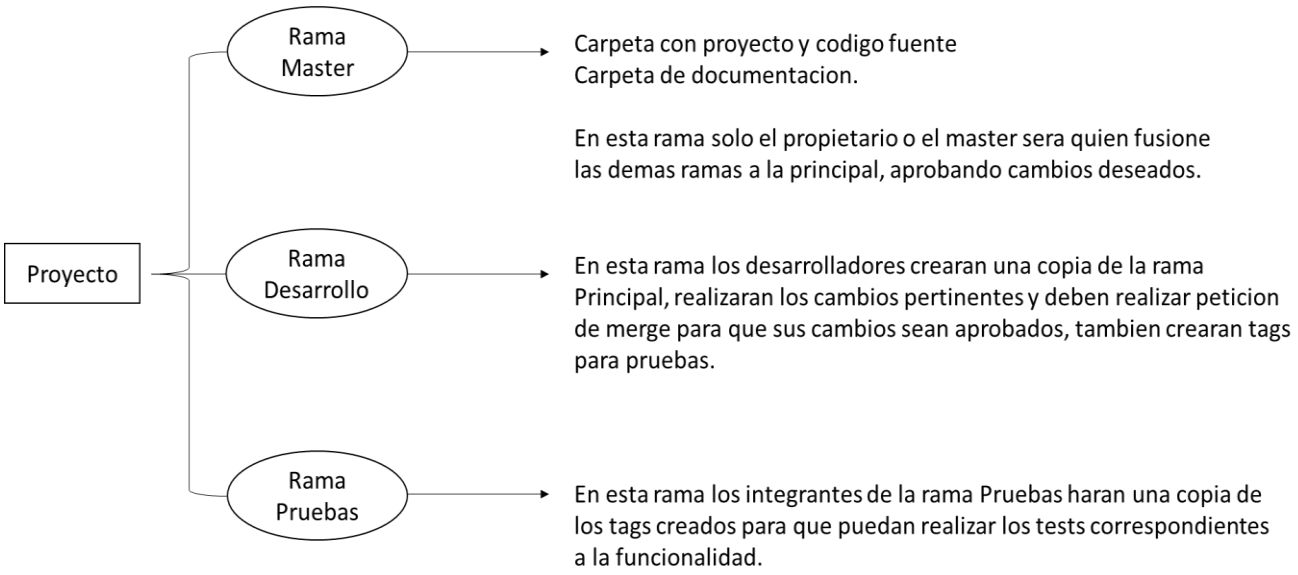


Ilustración 13:Planificacion del proyecto

Por lo tanto se recomiendan los siguientes roles: el propietario del proyecto sera el responsable de fusionar a la rama Master los cambios realizados en la rama Desarrollo, el rol de desarrollador para los integrantes de la rama Desarrollo, y aunque los integrantes de la rama Pruebas no deberian tener los privilegios de acceso que los desarrolladores, es necesario que lo tengan para que se unan a los tags generados en Desarrollo, realicen las pruebas pertinentes y dejen las notas de la version que se ha probado.

El numero de modulos a definir depende del tipo de aplicación, sin embargo se debe usar como principal criterio para la definicion de los modulos el bajo nivel de acoplamiento para garantizar que se pueda ejecutar casi de manera independiente..

Se sugiere que cada molulo sea registrado en git como proyectos independientes.

3.1.2 Instalación y Configuración de VCS

Acontinuacion se describira la instalacion y configuracion de herramientas necesarias para realizar el procedimiento propuesto, es necesario aclarar que la primera configuracion detallada acontinuacion sera a traves de comandos en la terminal, sin embargo, recomendamos la segunda configuracion donde se utiliza la interfaz web gitlab para facilitar la inicializacion del repositorio.

- Instalar Git y configurar usuario y correo inicial, este usuario sera el responsable de los commits realizados a nivel local y de los push en un repositorio remoto. (https://www.digitalocean.com/community/tutorials/how-to-install-git-on-debian-8)
- Añadir usuarios a grupo Git y dar permisos sobre la carpeta del repositorio, esto permite que solo usuarios deseados accedan al repositorio, sin embargo, esto se facilita mas adelante con el uso de la interfaz web gitlab, el cual permitira agregar usuarios a grupos determinados y que dichos grupos accedan a las ramas que esten autorizadas. (http://claves-de-linux.blogspot.com.co/2014/03/git-repositorio-compartido-crear-debian.html)
- Configurar repositorio de Git, esta configuracion debe realizarse para acceder al repositorio ya sea mediante ssh o http/s, inicializando el repositorio y añadiendo las llaves ssh para una autenticacion de usuarios mas simple.

(http://codehero.co/git-desde-cero-instalando-git-en-un-servidor/)

- Configurar repositorio remoto de Git, esta configuracion debe realizarse si se cuenta con un servidor remoto donde se va a almacenar el repositorio, incluyendo sitios como github, bitbucket, gitlab. (https://git-scm.com/book/es/v2/Fundamentos-de-Git-Trabajar-con-Remotos)

Con el procedimiento anterior un equipo de desarrollo ya puede comenzar a controlar la evolución de un proyecto a través de la herramienta Git, aunque de esta forma resulta un poco tedioso la gestión de usuarios, debido a que cada proyecto debe ser asignado a un grupo de usuarios con permisos de lectura y escritura siendo estos permisos asignados sobre la carpeta especifica del repositorio, a continuación se describirá el proceso para instalar gitlab, el cual permitirá administrar de forma fácil: usuarios, permisos y proyectos en nuestro servidor.

- Instalar servidor web, apache o utilizar nginx por defecto de gitlab
- Instalar gitlab (https://about.gitlab.com/downloads/#debian8)
- Configurar gitlab en servidor web (http://blog.hunabsys.com/apache-gitlab/)
- Crear usuarios y grupos de usuarios (https://git-scm.com/book/es/v2/Git-en-el-Servidor-GitLab)

3.1.3 Inicializacion de Repositorio

En este punto ya debe estar definida la estructura del proyecto por lo tanto ya es posible podra crear un repositorio remoto, esto es recomendable ya que se almacenara toda la informacion en un servidor, es posible que sea un servidor propio, o utilizar los sitios como github, bitbucket o en este caso gitlab.

Ahora bien si se desea inicializar un repositorio remoto por medio de comandos es necesario acceder al servidor donde se desea almacenar el repositorio y utilizar el comando ***git init - -bare*** , ahora solo faltara referenciar el servidor remoto en nuestro repositorio local; se referencia con el siguiente comando ***git remote add [nombre-servidor] [url]*** , de esta forma ya esta configurado el servidor remoto y es posible hacer uso de los comandos ***git push [nombre-servidor][rama]*** para confirmar cambios en el servidor, y ***git pull*** para bajar la ultima version que se encuentre en el servidor.

En este caso se realizara este procedimiento mediante el servicio web gitlab, permitira realizar este proceso de forma rapida.

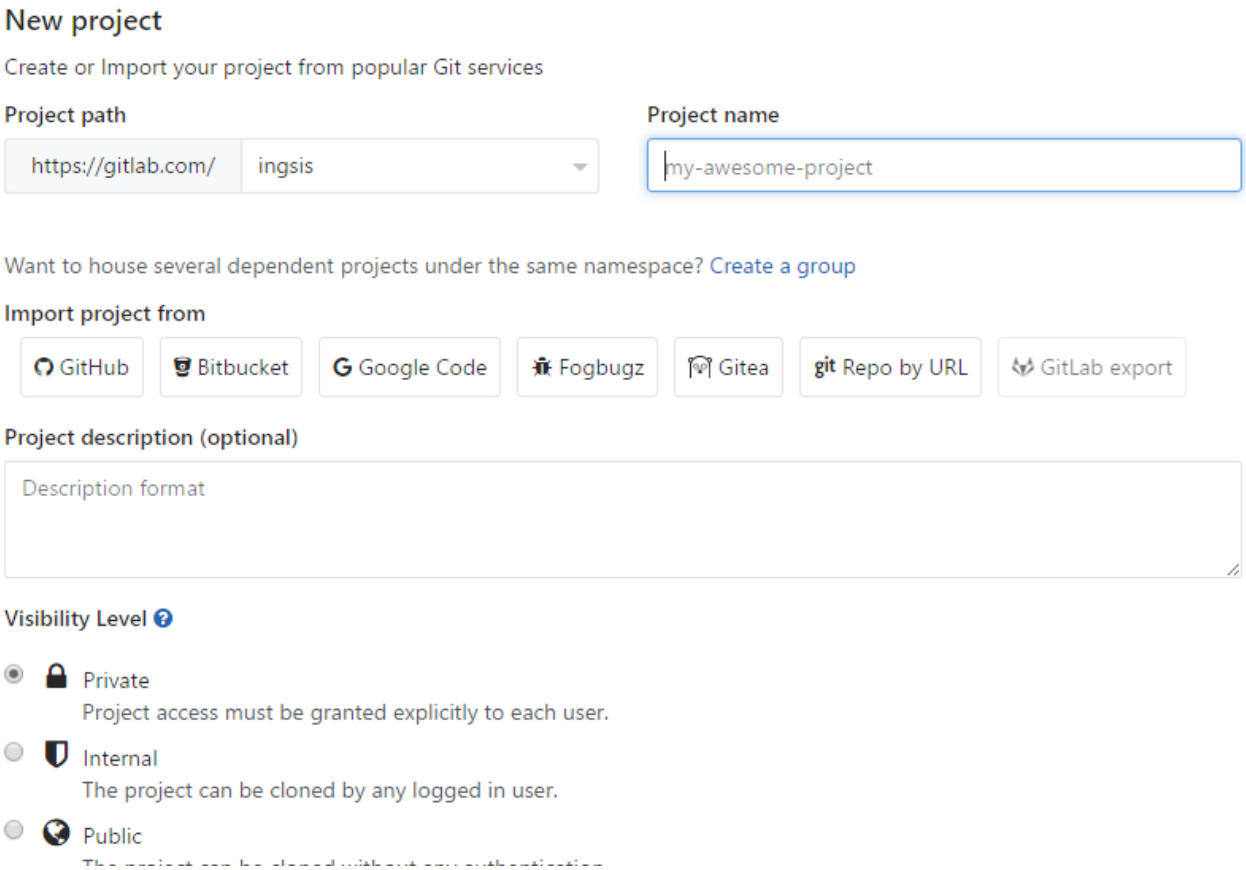


Ilustración 14:Crear proyecto Gitlab

Gitlab permite crear el repositorio o importarlo desde diversos sitios como Github, Bitbucket o una Url de otro servidor. Una vez creado ya se podra realizar el primer commit y push al repositorio.

El paso siguiente es añadir usuarios, cada usuario debe pertenecer a un grupo y cada grupo tendra permisos sobre una rama, siendo el propietario el responsable de fusionar los cambios en las demas ramas.

En este caso ya se definieron los roles y las ramas en el proyecto.

Permisos de Ramas-Usuarios	**Rama Dev**		**Rama Test**		**Rama Master**	
	Push	Merge	Push	Merge	Push	Merge
Integrantes grupo Owner (propietario)		✓		✓	✓	✓
Integrantes grupo groupDesarrollo	✓	✓				
Integrantes grupo groupPruebas			✓	✓		

Tabla 7: Permisos a ramas

3.1.4 Gestion de Ramas

En la Ilustración 13:Planificacion del proyecto se detalla la organización del proyecto a seguir, se crearan dos ramas ya que la rama **Master** es la principal por defecto, las ramas **Desarrollo** y **Pruebas** serán ramas protegidas, es decir, cada rama tendra asignados unos usuarios autorizados para realizar un push al repositorio remoto. El responsable de fusionar los

cambios sera el propietario del proyecto quien tendra autorizacion de realizar los merge correspondientes para integrar los cambios aprobados.

La rama master tendrá las carpetas principales del proyecto, los usuarios de la rama desarrollo, realizaran una copia de los archivos principales y realizaran los cambios pertinentes hasta cumplir cierto hito en el desarrollo, una vez superado se creara un tag para que los usuarios de la rama pruebas realicen los test pertinentes de esta version. Una vez revisado los integrantes de la rama pruebas aprobaran la version y subiran las notas de la version tag al repositorio remoto, si no es asi, comunicaran a los integrantes de la rama desarrollo cualquier fallo encontrado.

3.1.5 Gestion de Tags

Se utilizaran los Tags para etiquetar o marcar versiones clave del proyecto durante su desarrollo, por ejemplo, la primera version del proyecto que cumpla con un hito especifico.

La rama de desarrollo durante la evolucion del proyecto creara tags para que los integrantes de la rama de pruebas realicen los tests pertinentes a esta version, la cual se puede subir al repositorio remoto pero sin realizar cambios en este tag, esta version especifica con la etiqueta estara congelada durante el desarrollo y no puede evolucionar.

3.1.6 Integracion con Merge

Las fusiones de los cambios con merge seran responsabilidad del propietario del proyecto, por lo tanto él será el unico autorizado para realizar merge con las demas ramas.

Los integrantes de las ramas de desarrollo y pruebas pueden hacer una peticion de merge, una vez se hayan subido los cambios al servidor en sus respectivas ramas, despues el propietario o master decide si fusiona estos cambios a la rama principal del proyecto.

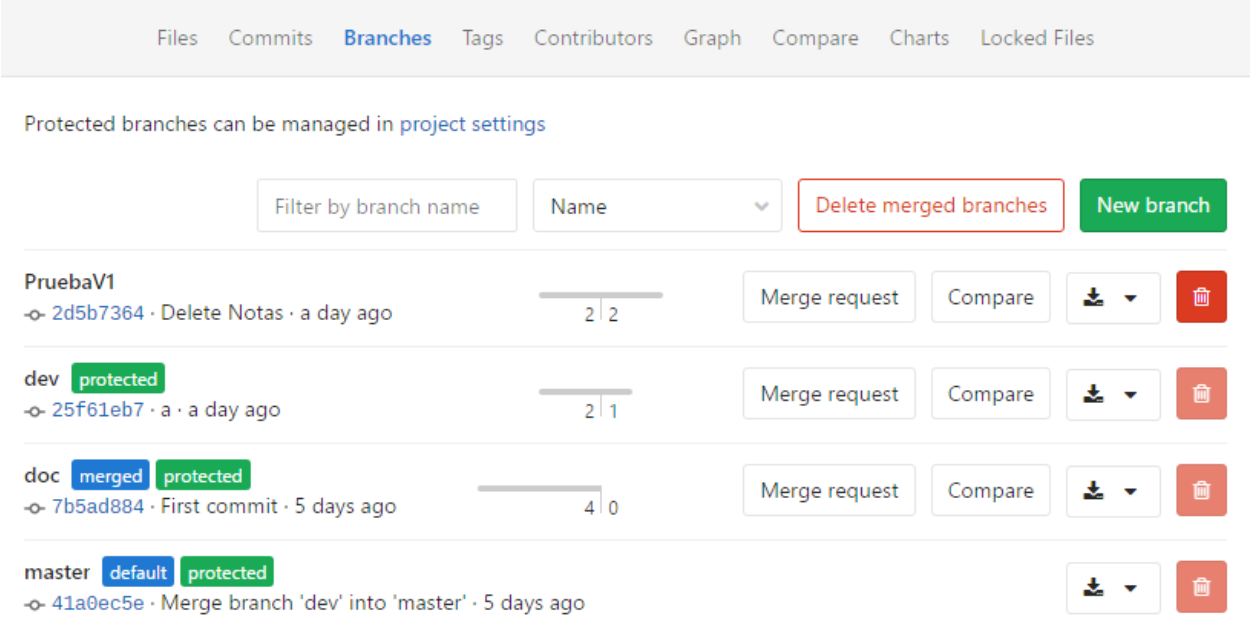


Ilustración 15:Merge request

3.1.7 Gestion de IES

Para controlar cada version de IES teniendo en cuenta la participacion en CIADTI, se utilizaran las propias ramas para cada IES vinculada a la plataforma de la Universidad, es decir, el desarrollo de cada IES va por separado, pero es posible fusionar caracteristicas a la rama principal o viceversa.

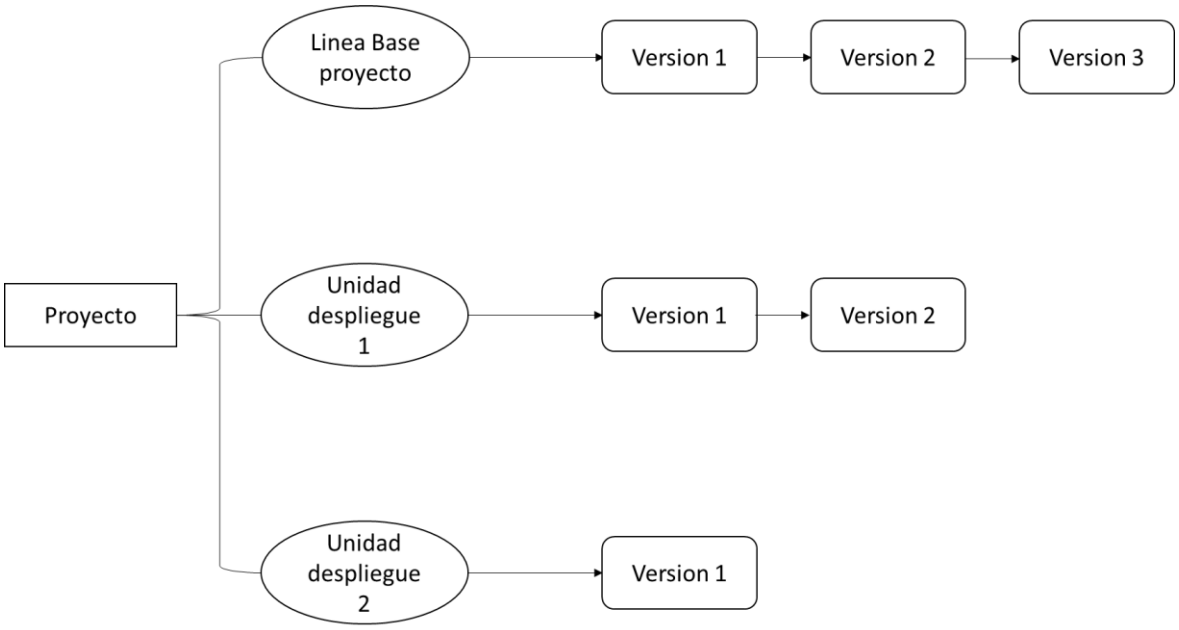


Ilustración 16:Gestion IES

3.2 Validacion del Procedimiento

Para la validacion del proyecto se realizó la instalacion del sistema Git en el servidor de la Universidad de Pamplona, junto a Gitlab que permitira gestionar los usuarios y sus respectivos roles.

3.2.1 Planificacion del proyecto

La estructura principal del proyecto se basa en crear dos carpetas, una para el proyecto web y otra para la documentacion, posteriormente se crearan las ramas de Desarrollo (dev) y de pruebas (test) y se darán permisos a aquellos usuarios que puedan realizar push a una determinada rama, el propietario será el unico con capacidad para fusionar lo que considere conveniente.

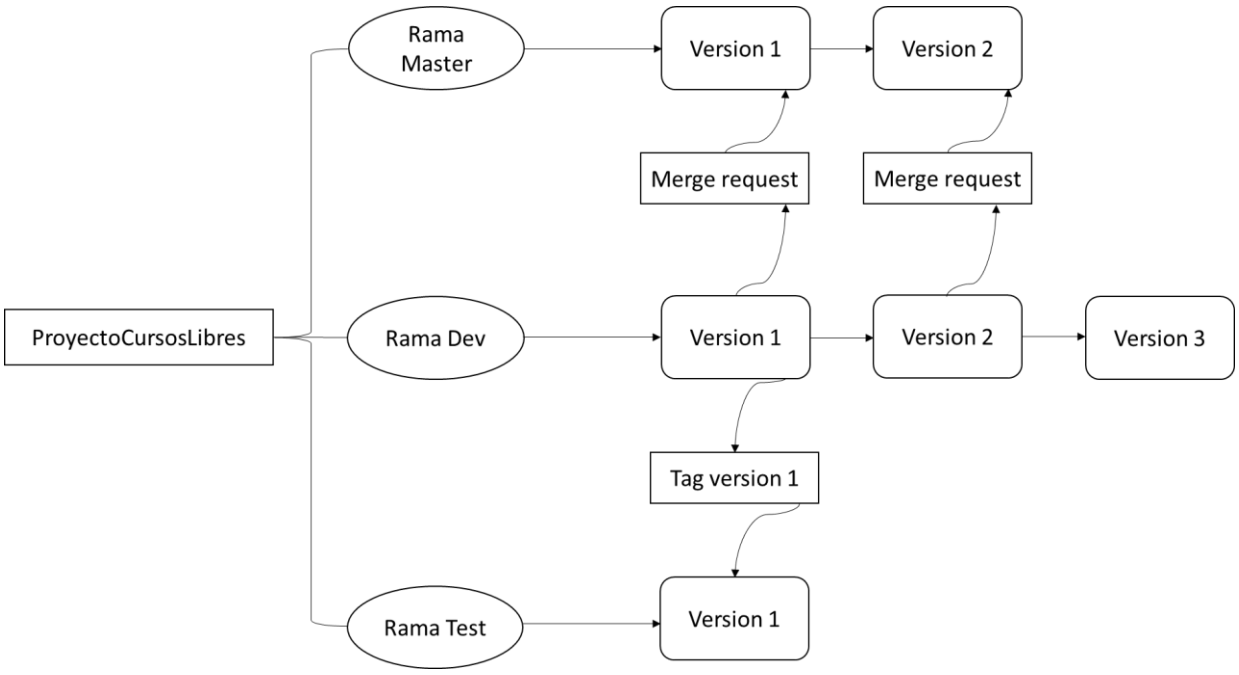


Ilustración 17:Planificacion Proyecto CursosLibres

3.2.2 Instalacion y Configuracion de VCS

Se realizó la instalacion y configuracion descritas en la seccion 3.1.2 en el servidor de software libre de la Universidad de Pamplona, instalando Git y Gitlab para gestionar repositorios y grupos de usuarios mediante roles.

3.2.3 Inicializacion de Repositorio

Para la inicializacion del repositorio se utilizara la interfaz de gitlab, una vez creado el proyecto se debe realizar un primer commit que puede ser subiendo un archivo readme, gitlab facilita esto en su misma area del proyecto.

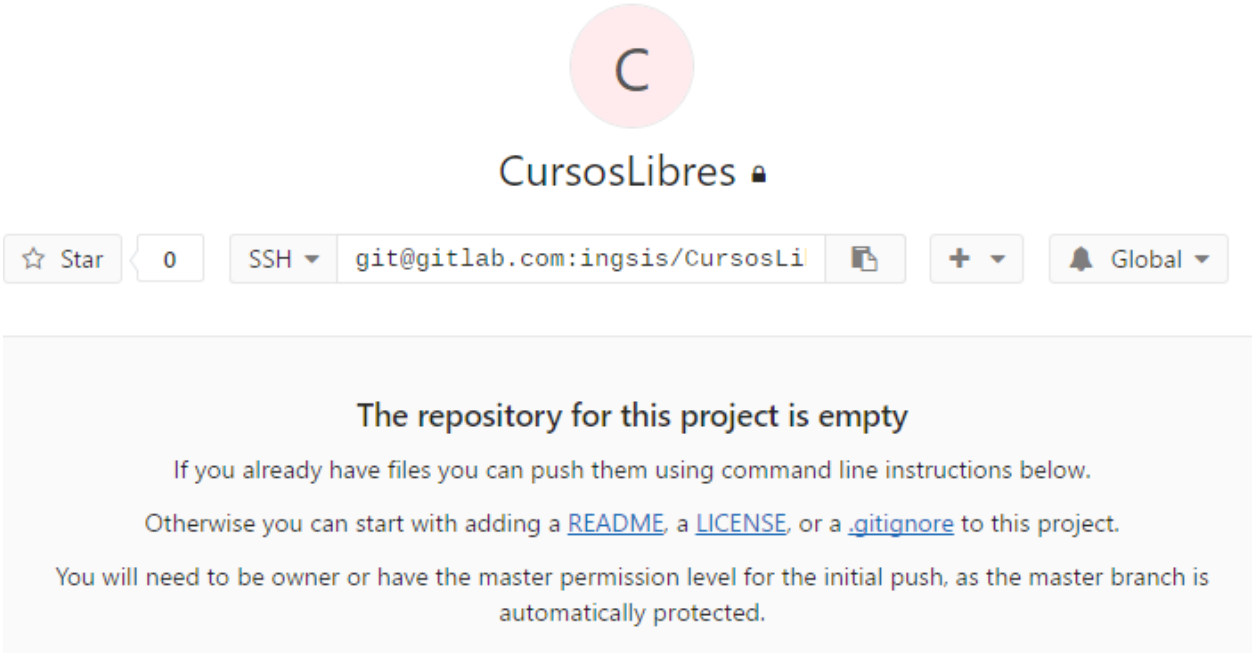


Ilustración 18:Creacion de repositorio

Es posible añadir cualquier archivo al repositorio, gitlab permite añadir facilmente un archivo Readme, la otra forma de hacerlo es clonar el repositorio y posteriormente realizar un push a este con los archivos deseados.

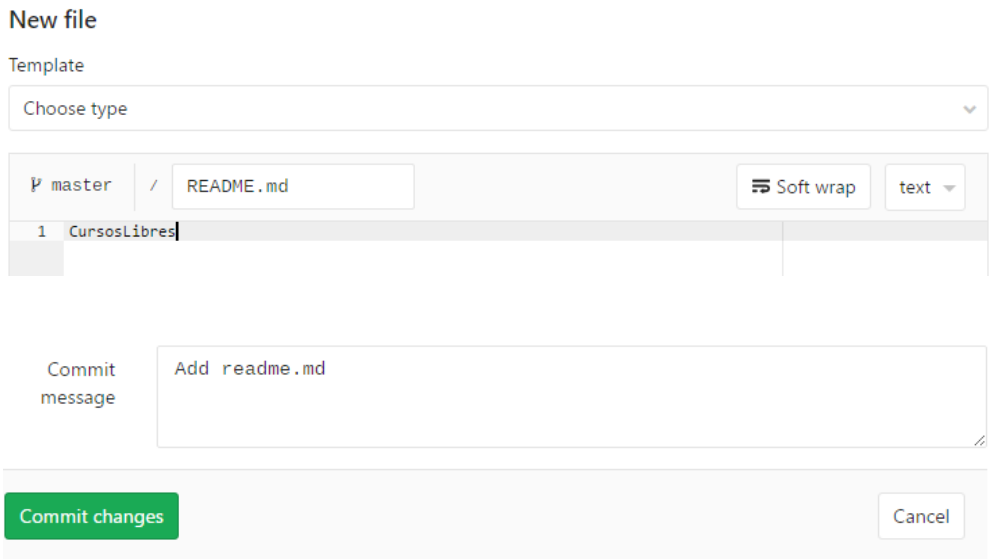


Ilustración 19:Inicializacion de repositorio

Apartir de ahora ya se esta controlando nuestro proyecto con Gitlab.

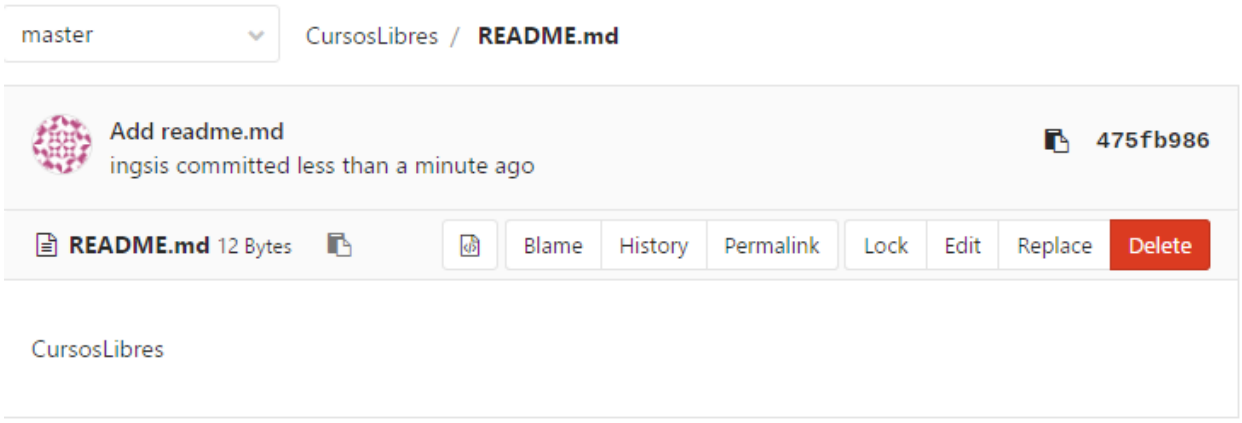


Ilustración 20:Archivos controlados

Ahora se procede a agregar los usuarios con sus respectivos permisos.

Es posible agregar tambien un grupo de usuarios a un proyecto, y asignar un rol determinado a cada grupo, los roles de gitlab son los siguientes y estan detallados en la Tabla 2.

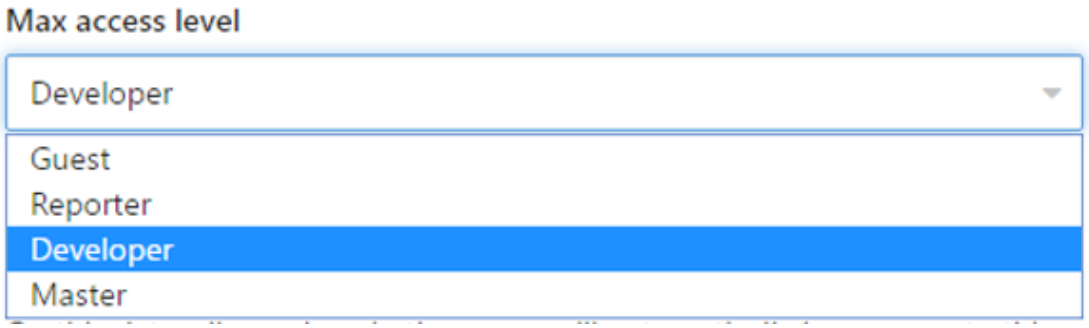
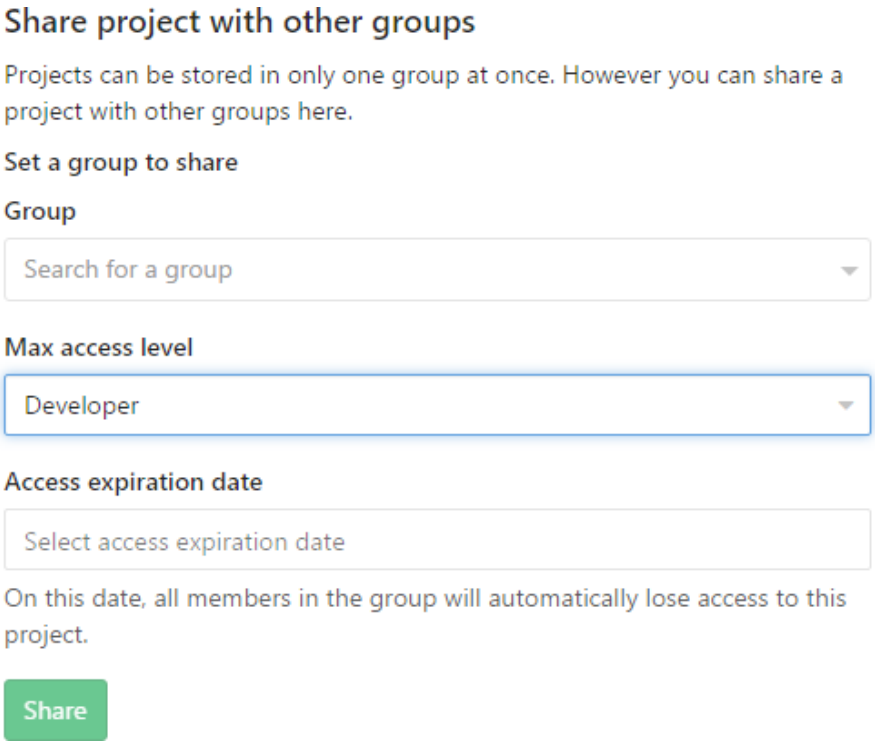


Ilustración 21:Roles gitlab



Para este proyecto se crearon dos grupos para poder acceder al repositorio, los dos grupos tendran el rol de desarrollador, esto debido a que el grupo groupDesarrollo sera el encargado de realizar avances en el proyecto añadiendo funcionalidades periodicamente, mientras que el grupo groupPruebas sera el responsable de unirse a los tags generados y realizar tests correspondientes a las funcionalidades.

3.2.4 Gestion de Ramas

Como se menciono en la seccion 3.1.4, se crearan dos ramas dev y test protegidas, en la seccion Configuracion/Repositorio del proyecto se pueden proteger estas ramas para que solo usuarios, grupos o roles puedan realizar push al repositorio, asi como quien es el responsable de fusionar las ramas.

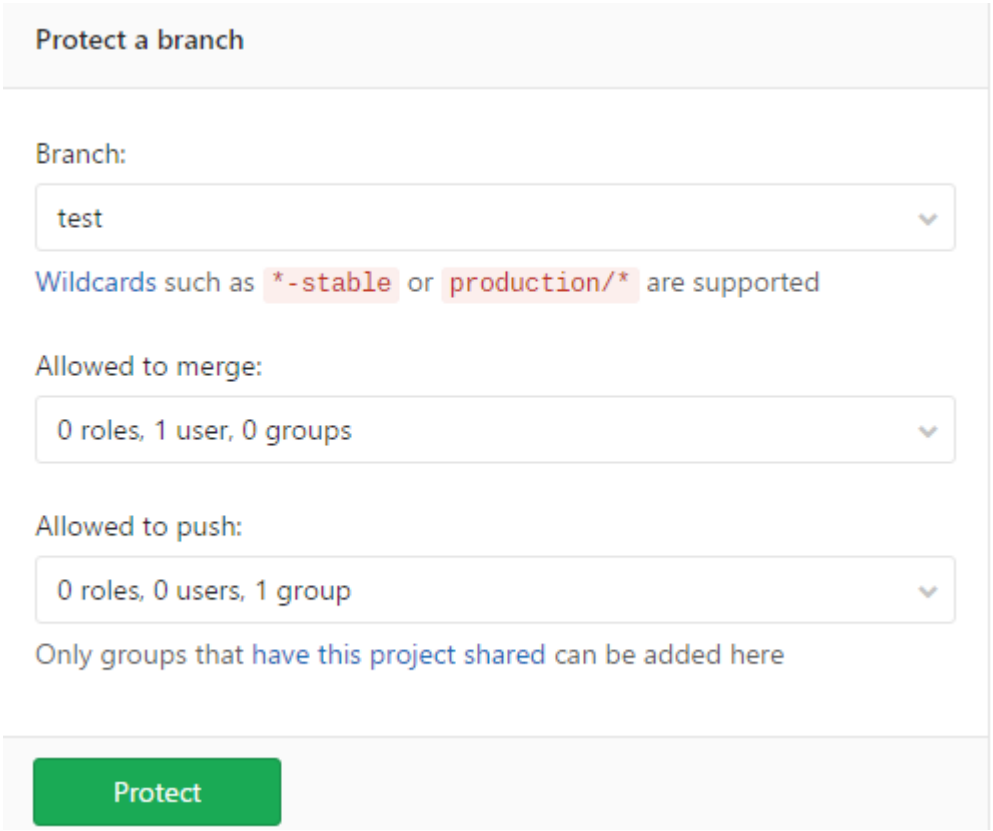


Ilustración 22: Protegiendo una rama

De esta forma la rama podra ser accedida por cualquiera que forme parte del proyecto con un nivel de acceso basico, sin embargo, solo podran realizar push aquellos que se indica en la proteccion de ramas, ya sea un usuario o un grupo de usuarios.

Para este proyecto quedan protegidas las siguientes ramas.

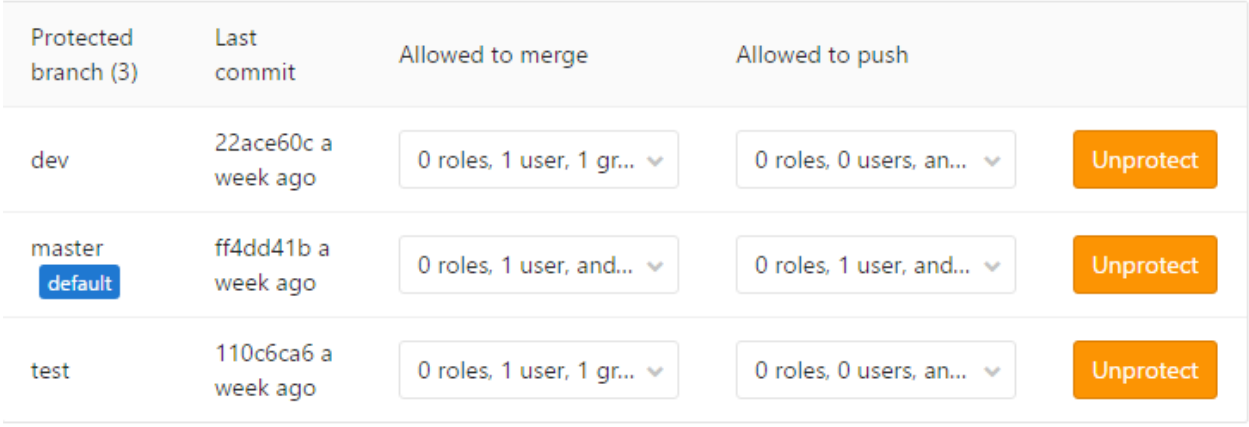


Ilustración 23:Ramas protegidas en ProyectoCursosLibres

3.2.5 Gestion de Tags

Para la creacion de tags se utilizara la herramienta Netbeans IDE que permitira realizar muchas operaciones con git, desde clonar el proyecto, realizar cambios, confirmar cambios y subirlos al repositorio. Esta herramienta tambien permitira que durante la evolucion de un proyecto se generen tags para algunas versiones, los integrantes de la rama pruebas podran realizar los tests a estas versiones para aprobar o no una version estable.

Los integrantes de la rama dev generaran los tags y los de la rama test accederan a ellos y realizaran pruebas a este.

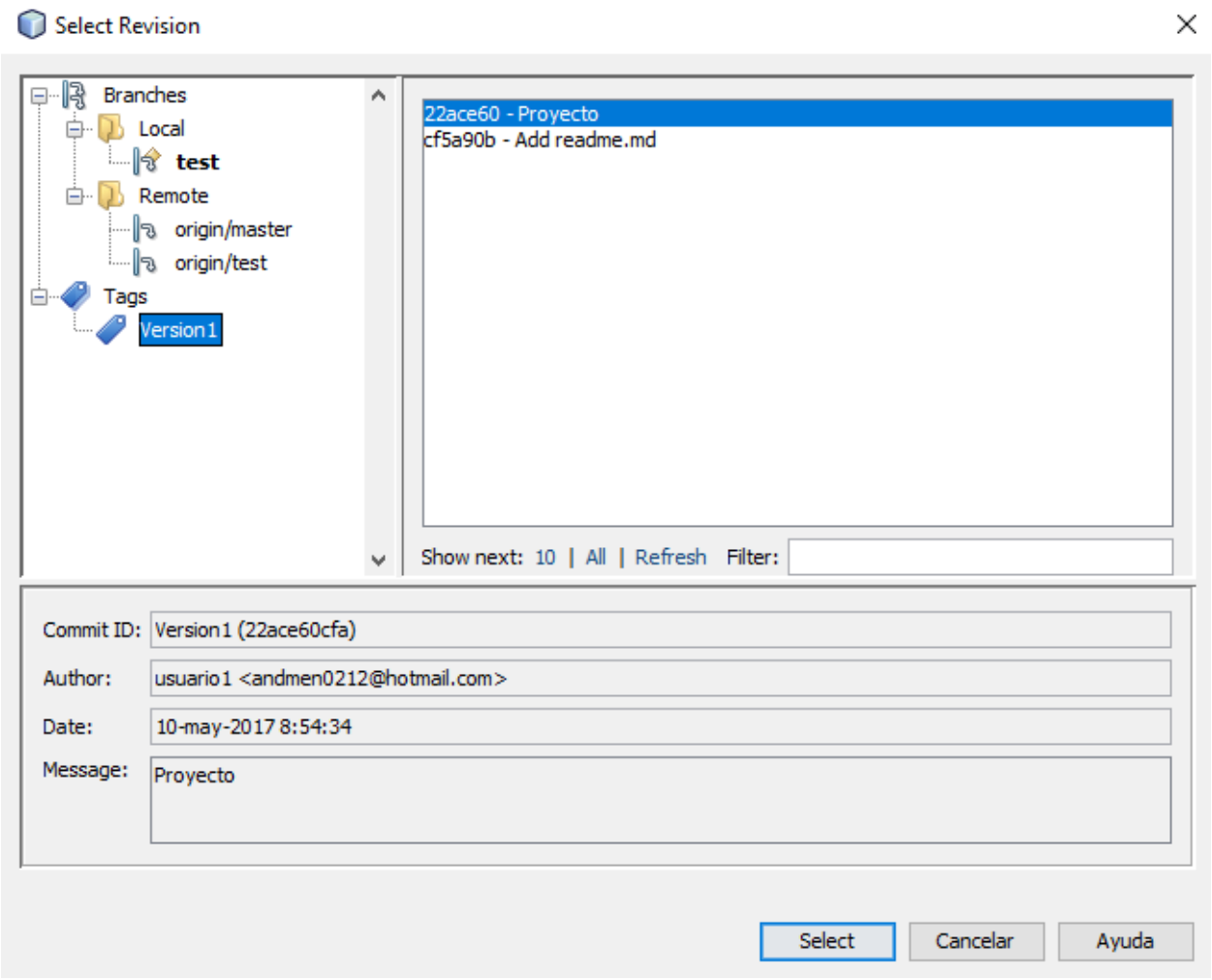
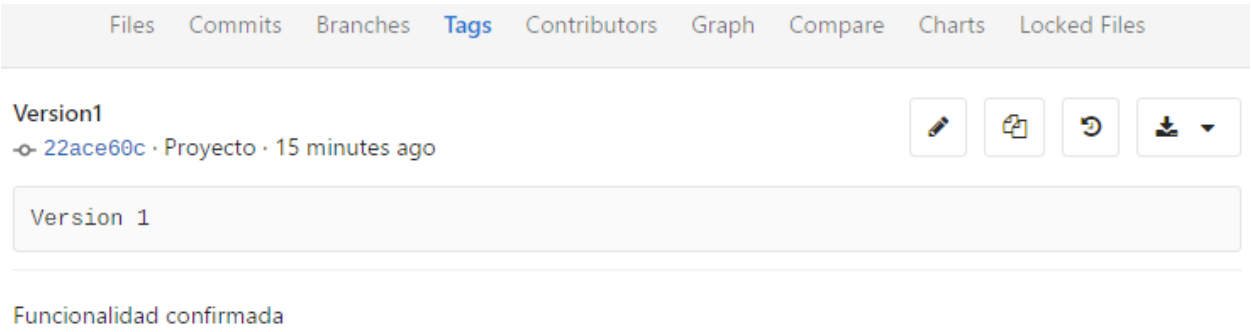


Ilustración 24:Accediendo a un tag

Este tag generado ya estara en el repositorio y podra ser aceptado por los integrantes de la rama test.



3.2.6 Integracion con Merge

Como se estructuraron las ramas protegidas en la seccion 3.2.4, las fusiones de las ramas a la principal sera responsabilidad del propietario del proyecto, al realizar los respectivos cambios en sus ramas por parte de los grupos asignados al proyecto, y posteriormente confirmar estos cambios, los usuarios y el propietario del proyecto podran realizar un merge request desde la plataforma web de gitlab.

A partir de este momento se informara al propietario del proyecto que tiene peticiones de fusion y decidira si fusionar los cambios realizados.

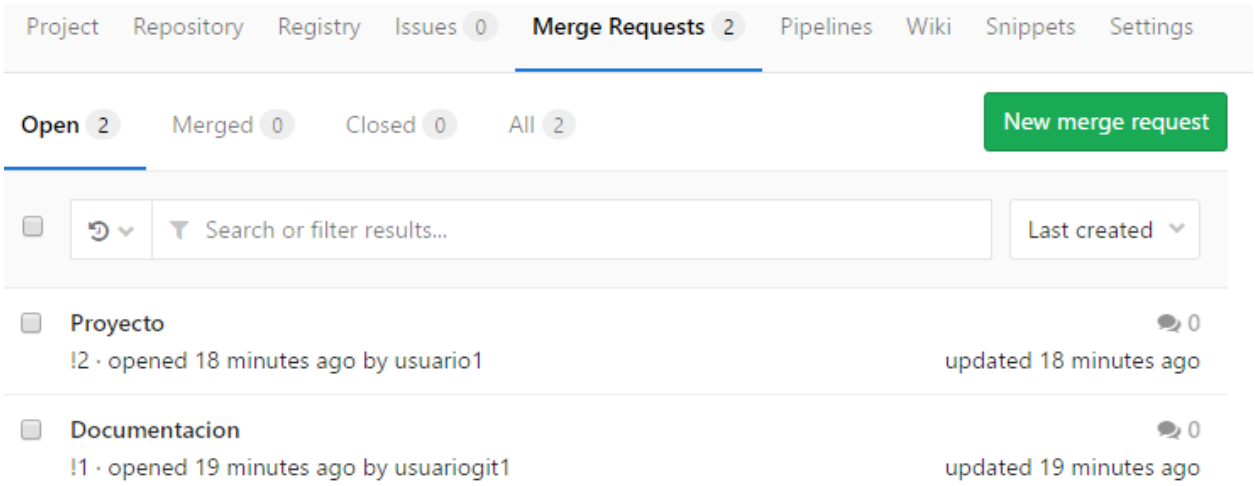


Ilustración 25:Lista de merge request

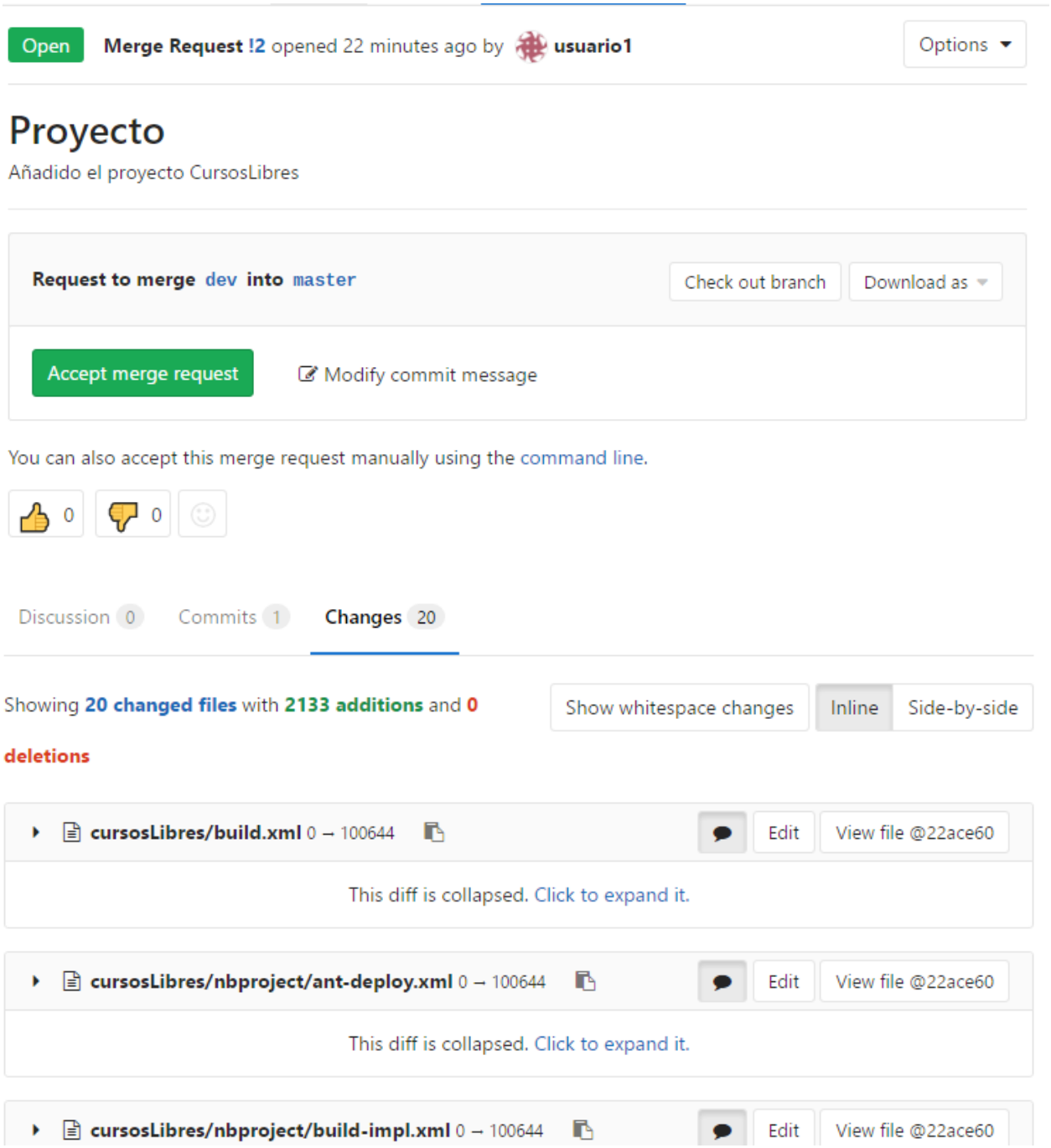


Ilustración 26: Ventana de fusion

Una vez el propietario ingrese a su plataforma podra ver los datos de la peticion, como el usuario que genero el merge request, la fecha, si este cumplio algun hito en el desarrollo. Y posteriormente podra observar los cambios realizados.

Una vez aprobado el contenido de las ramas, la rama master contendra todos los archivos aprobados por el propietario, es necesario especificar que los cambios seran totales, aunque se permiten editar los archivos, no es posible eliminar archivos debido a que se aprueba un commit completo.

3.2.7 Gestion de IES

La validacion de esta seccion no es posible debido a que el proyecto utilizado no representa la estructura en Academusoft, por lo tanto mediante nuestra experiencia en CIADTI, se recomienda el uso de ramas para que cada IES tenga un desarrollo independiente, y que sea posible integrar las ramas de IES con la Principal de Academusoft en cualquier momento y viceversa.

4 Conclusiones, recomendaciones y trabajos futuros

4.1 Conclusiones

- Los sistemas de control de versiones ofrecen un soporte importante para el proceso de desarrollo de software, impulsando el trabajo colaborativo y distribuido al permitir que multiples desarrolladores accedan paralelamente a un proyecto y realicen sus propios cambios de forma autonoma e independiente pero supervisada. Ademas de permitir la visualizacion y modificacion de versiones anteriores de un proyecto, permite realizar comparaciones entre distintas versiones durante el ciclo de vida del proyecto.
- Mediante la documentacion realizada se encuentran diferentes sistemas para el control de versiones, en este documento se analizan y se comparan algunas como son Concurrent Versions System(CVS), Subversion (SVN), Bazaar, Git y Mercurial. A partir de esta informacion se observa que Git es uno de los sistemas mas completos preferidos por las comunidades observadas como gnome, github, sourceforge.
- Teniendo en cuenta el procedimiento propuesto es posible concluir que es conveniente dividir el proyecto mediante ramas ya que no solo facilita el trabajo distribuido sino que tambien permite la correcta gestion de roles dentro de una rama especifica.
- La validacion del proyecto fue posible mediante la estructura de ramas que se utilizo en este procedimiento, fue posible controlar quienes tienen acceso, a realizar push en el repositorio y quienes pueden fusionar las ramas para tener una version principal.
- Git es el elegido por nosotros para controlar el desarrollo y evolucion del proyecto. Este sistema ofrece la posibilidad de crear ramificaciones para que cada desarrollador trabaje por su cuenta, en un entorno aislado de la linea principal de desarrollo por lo que cada modificacion no afecta a los demas integrantes ni al proyecto principal. Al finalizar es

posible fusionar cada rama creada para unir todos las modificaciones realizadas por los distintos integrantes del proyecto.

- Git es un sistema muy veloz y versatil debido a que muchas de sus operaciones se realizan de forma local, por lo que no es necesaria la conexión a internet para realizar modificaciones al proyecto. Sin embargo, no existe ningun sistema de control de versiones ideal, git tiene algunas desventajas, algunas de ellas son: la curva de aprendizaje para aquellos que vienen de un sistema como Subversion, la imposibilidad de administrar permisos a un directorio en especifico, y la falta de interfaces graficas que faciliten su uso es evidente con respecto a Subversion.
- Gitlab al ser una interfaz web facilita la gestion de usuarios y grupos de usuarios mediante roles, esto permite que cada integrante del proyecto acceda y modifique solo la seccion del proyecto a la que esta autorizada, ademas el propietario del proyecto puede aprobar estas modificaciones si el proyecto esta configurado de esa forma.
- Gitlab ofrece muchos beneficios a la hora de gestionar repositorios durante la evolucion y desarrollo de un proyecto, la integracion continua y el despliegue continuo, sin embargo, tambien carece de algunas caracteristicas que lo harian mucho mas completo, por ejemplo, la posibilidad de eliminar algunos archivos cuando se realiza un merge request, para que se apruebe solo lo que realmente se desea a la linea base del proyecto, los permisos a una carpeta del repositorio serian fundamentales, y tambien el hecho de que esta interfaz no nos permite utilizar los submodulos de git, para aprovechar esta caracteristica es necesario recurrir a otras herramientas como gitkraken.

4.2 Recomendaciones

Normalmente un proyecto grande debe estar dividido en modulos y submodulos en los cuales estaran dispuestas las funcionalidades, este tipo de division permite organizar los archivos y el codigo fuente de una manera que el usuario pueda navegar por la estructura del proyecto de forma fluida y sin complicaciones. Ademas ayuda en dos conceptos muy relevantes en ingenieria del software acoplamiento[1] y cohesion[2]. Por lo tanto si se logra que el acoplamiento de un sistema sea bajo se puede decir que tiene una alta cohesion y que el sistema esta bien estructurado y posee un buen diseño de software.

Basado en lo anterior, es recomendable que la estructura del proyecto permita dividirlo en modulos y submodulos, para que de esta forma sea posible utilizar y aprovechar los submodulos de Git y asi crear una referencia a un repositorio remoto para que el codigo no se duplique, de esta forma se asegura que los desarrolladores autorizados para cada modulo sean los unicos que puedan realizar los cambios en estos.

4.3 Trabajos futuros

Gitlab permite el uso conjunto con algunas aplicaciones, por lo tanto, posteriormente se realizara una integracion con aplicaciones como Bugzilla, que permite realizar un seguimiento de los errores que se producen durante el desarrollo de un proyecto. Y la integracion con Jenkins, que facilita la integracion de cambios en un proyecto, ademas de liberaciones de versiones en el mismo.

[1] Acoplamiento: El acoplamiento se refiere a la interdependencia entre módulos, si es necesario realizar algun cambio a un modulo, lo mas seguro es que aquellos con los que se tenga una interdependencia tambien se deban ajustar a dichos cambios.

[2] Cohesion: la cohesión describe cómo se relacionan las funciones dentro de un módulo independiente.

5 Referencias Bibliografía

5.1 Bibliografia

Bartolomé, C. (2014). TRABAJO FIN DE GRADO El Desarrollo de Software Open Source Analizado desde Dentro.

Bourque, P., & Fairley, R. E. (2014). *Guide to the Software Engineering - Body of Knowledge. IEEE Computer Society*. http://doi.org/10.1234/12345678

Carrasco, A. V. (2016). Análisis de la comunicación en las comunidades de software libre.

Collado, M. (2004). Control de versiones , configuración y cambios.

German, D. M. (2006). The Flow of Knowledge in Free and Open Source Communities. *2nd. International Workshop in Supporting Knowledge Collaboration in Software Development (KCSD'06)*.

González, L. P. (2010). Sistemas para el Control de Versiones.

Gutierrez, D. O. (2011). Desarrollo de una aplicacion Web para control de versiones de software.

M¥rka, M. (2006). Distributed Source Code Version Control Systems : Comparison and Usage Evaluation.

Paletta, M. (2009). MODELO DE APRENDIZAJE PARA ENTORNOS DISTRIBUIDOS COLABORATIVOS.

Paredes, L. M. (2011). Gestión de configuración: Validación De Un Modelo Liviano Para Pequeñas Empresas De Desarrollo De Software. *Entramado*, *7*(1), 190–201.

Raymond, E. (2005). The cathedral and the bazaar. *First Monday*, *2*(SPEC), 1–40. http://doi.org/10.1007/s12130-999-1026-0

Scv, C. L. (2014). Comparativa Sistemas de control de versiones.

Stallman, R. M. (2004). *Software libre para una sociedad libre*.

Tania E, T. Sen. (2012). Software Libre y abierto: comunidades y redes de producción digital de bienes comunes.

Tello-leal. (2012). Revision de los sistemas de control de versiones utilizados en el desarrollo de software, *3*(1), 74–81.

Wanumen, L. F. (2014). Los sistemas de control de versiones, (November 2008).

5.2 Infografia

"gnome," (https://www.gnome.org/get-involved/,2017)

"aomedia," (http://aomedia.org/contributor-guide/,2015)

"django," (https://code.djangoproject.com/,2008)

"mozilla," (https://www.mozilla.org/en-US/contribute/signup/,2003)

"github," (https://github.com/open-source,2017)

"sourceforge," (https://sourceforge.net/,2017)

"Software libre," (https://hipertextual.com/archivo/2014/05/diferencias-software-libre-y-open-source/,2014)

"Open source," (https://www.genbeta.com/a-fondo/cual-es-la-diferencia-entre-el-software-libre-y-el-open-source,2016)

"git," (https://git-scm.com/book/es/v1/Empezando-Acerca-del-control-de-versiones,2017)

"bazaar," (http://chuwiki.chuidiang.org/index.php?title=Formas_de_trabajo_con_Bazaar,2009)