



UNIVERSIDAD AUTÓNOMA DE GUERRERO

FACULTAD DE INGENIERÍA



TRABAJO DE INVESTIGACIÓN

REPOSITORIO EN GITHUB

QUE PRESENTA

ROSALINDA MOSSO MOSSO

**PARA OBTENER EL TÍTULO DE
INGENIERO EN COMPUTACION**

DIRECTORA DE TESIS

DRA. ALMA VILLASEÑOR FRANCO

CHILPANCINGO, GUERRERO, FEBRERO DE 2021

CONTENIDO

JUSTIFICACIÓN	5
ALCANCES.....	6
OBJETIVOS:.....	6
OBJETIVO GENERAL	6
MARCO TEÓRICO E HIPÓTESIS	7
CAPITULO 1.....	7
REPOSITORIO DENTRO DE LA INGENIERÍA.....	7
1.1 Acceso abierto	8
1.2 Las ventajas del acceso abierto al conocimiento	9
1.3 El acceso abierto como nuevo modelo de comunicación científica	10
1.4 Las políticas que apoyan el acceso abierto	11
1.5 Un elemento clave en el acceso abierto: los repositorios.....	11
1.6 La interoperabilidad y visibilidad de los repositorios	12
1.7 Introducción a Git	12
1.8 Clasificación	13
1.9 Github	16
1.10 Importancia GitHub	17
1.11 Los tres estados de Git	17
1.12 Flujos de trabajo distribuidos con git	18
1.13 Flujo de trabajo centralizado.....	18
1.14 Flujo de trabajo del Gestor de Integraciones	19
1.15 Flujo de trabajo con Dictador y tenientes	20
CAPÍTULO 2	21
NECESIDADES DE UN REPOSITORIO	21
2.1 GitHub aplicado a la docencia	22
2.2 Aprendizaje del alumno	22
2.3 Las herramientas y funcionalidades exclusivas de GitHub son:.....	24
CAPITULO 3	25
METODOLOGÍA.....	25
3.1 Uso básico para trabajar con Git y GitHub	25
3.2 Descargar un proyecto por primera vez	26

3.3 Ver los cambios realizados en el repositorio	27
3.4 Añadir cambios	27
3.5 Enviar los cambios al repositorio alojado en GitHub	28
3.6 Descargar los últimos cambios	29
3.7 Crear un proyecto	29
3.8 Crear el repositorio	30
3.9 Confirmar los cambios	32
3.10 Ignorando archivos	34
3.11 Observando los cambios	37
3.12 Recuperando versiones anteriores	38
3.13 Volver a la última versión de la rama master	39
3.14 Etiquetando versiones	39
3.15 Borrar etiquetas	40
3.16 Visualizar cambios	40
CAPÍTULO 4	41
IMPLEMENTACIÓN DE SEGURIDAD	41
4.1 Determinar dónde se ha usado la clave	41
4.2 Administrar las llaves de despliegue	41
4.3 Reenvío del agente SSH	42
4.4 Llaves de implementación	42
4.5 Utilizar el reenvío del agente SSH	46
4.6 Configurar el reenvío del agente SSH	47
4.7 Probar el reenvío del agente SSH	47
4.8 Solucionar problemas del reenvío del agente SSH	48
4.9 Debes utilizar una URL con SSH para revisar el código	48
4.10 Tus llaves SSH deben funcionar localmente	48
4.11 Tu sistema debe permitir el reenvío del agente SSH	49
4.12 Tu servidor debe permitir el reenvío del agente SSH en las conexiones entrantes	49
4.13 Tu ssh-agent local debe estar ejecutándose	49
CONCLUSIÓN	52
REFERENCIAS	53

INTRODUCCIÓN

Git y GitHub son plataformas de desarrollo que ayudan a los desarrolladores a alojar y revisar código, gestionar proyectos y crear software junto con millones de otros desarrolladores. Git es el sistema de control de versiones distribuido de código abierto que facilita las actividades de GitHub. Los proyectos que usan Git se almacenan públicamente en GitHub y GitHub Pages, por lo que, de una manera muy generalizada, Git es lo que haces en tu propia computadora y GitHub es el lugar donde se almacena públicamente en un servidor.

Un repositorio es un directorio donde se almacenan los archivos de un proyecto, imágenes, audios, etc. Esta herramienta fue impulsada por Linux Torvalds y el equipo de desarrollo del Kernel de Linux, que al igual que este sistema operativo, lo lanzaron como código abierto, además de estar para las distintas plataformas del mercado. Podemos encontrar una gran diferencia del uso de Git respecto a otras herramientas similares.

Git Bash es una aplicación para entornos de Microsoft Windows que ofrece una capa de emulación para una experiencia de líneas de comandos de Git. Que además te permite lanzar comandos de Linux básicos, "ls -l" para listar archivos, "mkdir" para crear directorios, etc. Es la opción más común para usar Git en Windows.

JUSTIFICACIÓN

Git y GitHub son plataformas de desarrollo que ayudan a los desarrolladores a alojar y revisar código, gestionar proyectos y crear software junto con millones de otros desarrolladores. Git es el sistema de control de versiones distribuido de código abierto que facilita las actividades de GitHub. Los proyectos que usan Git se almacenan públicamente en GitHub y GitHub Pages, por lo que, de una manera muy generalizada, Git es lo que haces en tu propia computadora y GitHub es el lugar donde se almacena públicamente en un servidor.

Un repositorio es un directorio donde se almacenan los archivos de un proyecto, imágenes, audios, etc. Esta herramienta fue impulsada por Linux Torvalds y el equipo de desarrollo del Kernel de Linux, que al igual que este sistema operativo, lo lanzaron como código abierto, además de estar para las distintas plataformas del mercado. Podemos encontrar una gran diferencia del uso de Git respecto a otras herramientas similares.

Git Bash es una aplicación para entornos de Microsoft Windows que ofrece una capa de emulación para una experiencia de líneas de comandos de Git. Que además te permite lanzar comandos de Linux básicos, "ls -l" para listar archivos,

"mkdir" para crear directorios, etc. Es la opción más común para usar Git en Windows.

ALCANCES

Tener en claro cómo funciona la plataforma de GitHub dejando una estructura (repositorio) para adicionar archivos recabados de distintas áreas. Mediante una serie de comandos para poder subir dicha información.

OBJETIVOS:

Desarrollar una estructura (repositorio) en GitHub, como plataforma de comunicación, que favorezcan la difusión y compartición de información entre usuarios. Así como gestionar la producción científica y académica.

OBJETIVO GENERAL

- ❖ Recabar información.
- ❖ Facilitar el acceso a la información científica y académica.
- ❖ Retroalimentar la investigación.
- ❖ Guardar información.
- ❖ Permitir a los usuarios consultar la información (Acceso abierto).

MARCO TEÓRICO E HIPÓTESIS

CAPITULO 1.

REPOSITORIO DENTRO DE LA INGENIERÍA

Los repositorios digitales de acceso abierto son los soportes de la vía verde para alcanzar el acceso abierto a la producción científica. Clifford Lynch define el repositorio de acceso abierto como un conjunto de servicios que una Universidad ofrece a los miembros de su comunidad para la difusión de materiales digitales creados por la institución o sus miembros. Los repositorios de acceso abierto tienen la misión de difundir y preservar la producción académica. Además, tienen un ámbito claro: una institución, una entidad financiadora o una disciplina académica y, por otra parte, deben depender de organizaciones que tienen calidad académica para asegurar su misión. Estas características son importantes para que el repositorio sea una herramienta adecuada para difundir la investigación. Sin embargo, el repositorio no tiene la propiedad de los documentos. Como principio, el autor siempre controla su obra, tanto el depósito como la retirada o modificación. Los repositorios deben hacer accesibles los metadatos y contenidos de los registros que contienen siguiendo las políticas editoriales y las leyes aplicables en cada caso (Rocio, 2017).

Existen varias clasificaciones de repositorios. La más frecuente es dividirlos entre repositorios institucionales (los que reúnen los contenidos digitales de una organización: universidad, centro de investigación, etc.), o temáticos (son aquellos que centran su contenido en una materia: ArXiv en Física, RePec en económicas, E-LIS en documentación, etc.). Otros autores añaden una tercera tipología: format repositories, que se limitan a reunir documentos de determinados formatos: tesis doctorales, datos de investigación, imágenes digitales, etc. Afinando un poco más, se ha establecido una clasificación que considera cuatro tipos de repositorios: temático, institucional, repositorio de investigación (subvencionados por entidades financiadoras, con el fin de mostrar y compartir los resultados de sus investigaciones), y los repositorios de sistemas nacionales, que se crean para recoger los resultados académicos de un modo más general, y no solo para su preservación. Otra modalidad son los repositorios huérfanos (orphan repository), que sirven para el depósito de objetos digitales de autores que no tienen dónde hacerlo, como es el caso de Zenodo, creado a petición de la Comisión Europea para trabajos financiados con fondos de europeos.

En el informe también se analiza el impacto que tienen los repositorios sobre las bibliotecas, los investigadores y profesores, los editores y las agencias gubernamentales u otras entidades financiadoras. Para las bibliotecas, la creación de un repositorio institucional transforma su función de simple custodia de la información, a ser un agente activo en la comunicación académica. Este hecho,

además, propicia el aumento de la visibilidad de la biblioteca dentro de la propia institución. En cuanto a los profesores e investigadores, este informe describe claramente las dificultades existentes para que se adapten a este nuevo modelo: problemas con el copyright, resistencia al cambio, etc. Pero también describe las ventajas que les reporta: más visibilidad, mayor impacto de la publicación, más facilidad para defender la titularidad de las obras, etc. Los materiales docentes, al poder incluirlos también en el repositorio se convierten en un apoyo para las clases. Adelanta también este informe las dificultades a las que se enfrentan las editoriales comerciales, para las que el acceso abierto supone cambiar el modelo de negocio. Solo aquellas que sepan modificar sus políticas y adaptarse a la nueva situación podrán sobrevivir a este cambio.

Los contenidos que se pueden encontrar en los repositorios institucionales van desde publicaciones institucionales, hasta materiales docentes, e incluso inventarios de archivos de la institución, pasando por fondo antiguo, colecciones de imágenes o videos. En realidad, toda la producción digital de la institución tiene cabida en el repositorio, aunque hay que distinguir dentro del mismo la producción científica del resto de materiales.

1.1 Acceso abierto

El desarrollo tecnológico facilita la comunicación del conocimiento científico, permitiendo ampliar los canales de difusión y reduciendo significativamente los costes de la transmisión de la investigación. Surgen nuevos paradigmas de comunicación científica como son los repositorios de acceso abierto que deben ser aprovechados para ofrecer contenidos académicos y de investigación libremente, con el fin de que la producción científica generada mundialmente esté al alcance de la sociedad.

El objetivo de este estudio es informar de los beneficios del modelo de comunicación científica en acceso abierto a través de los repositorios, utilizando como ejemplo la Revista de la Sociedad Otorrinolaringológica de Castilla y León, Cantabria y la Rioja depositada y difundida en acceso abierto a través de Gredos, el Repositorio Institucional de la Universidad de Salamanca.

Método: Se presentan los fundamentos, el estado actual, las tendencias y las ventajas del acceso abierto, entendido como un cambio radical en el funcionamiento del sistema de comunicación científica.

Se definen y analizan los repositorios, que constituyen la “vía verde” para conseguir el acceso abierto al conocimiento.

Los repositorios de acceso abierto constituyen una nueva vía para difundir las revistas científicas de tal forma que los trabajos publicados en ellas alcanzan la máxima difusión y visibilidad, aumentando la tasa de citación.

Discusión: El estudio se centra en el depósito de los artículos científicos en los repositorios, y en concreto en el caso del repositorio institucional Gredos de la Universidad de Salamanca para explicar las ventajas de la difusión de los contenidos de la Revista de la Sociedad Otorrinolaringológica de Castilla y León, Cantabria y La Rioja a través del repositorio Gredos, haciendo hincapié en el aumento de visibilidad de los contenidos científicos alojados en el repositorio Gredos (Ferrerías-Fernández T, 2015).

Conclusiones: Actualmente el movimiento a favor del acceso abierto está suficientemente consolidado como se muestran los datos ofrecidos en este trabajo.

Los repositorios son una pieza clave en el desarrollo de este movimiento, ofreciendo múltiples ventajas a autores, instituciones y al público en general. La Revista de la Sociedad Otorrinolaringológica de Castilla y León, Cantabria y La Rioja al difundirse a través del repositorio Gredos aumentará la visibilidad de sus contenidos y aumentará la tasa de citación de los mismos, a la vez que contribuye al bien público.

Esto quiere decir que las dos condiciones para que los contenidos científicos se consideren de acceso abierto son que estos contenidos sean gratuitos y, además que estén libres de algunas restricciones de derechos de explotación.

El acceso abierto al conocimiento es un movimiento que reclama la difusión y reutilización del conocimiento libremente en internet y esto representa un cambio radical de modelo en el funcionamiento de la comunicación científica.

Los efectos generados tanto por internet y como por la digitalización de contenidos unidos al sistema de comunicación científica han sido fundamentales para facilitar el desarrollo del acceso abierto, haciendo posible que los contenidos científicos digitalizados sean difundidos de forma instantánea y a bajo coste a través de internet.

1.2 Las ventajas del acceso abierto al conocimiento

Una vez que sabemos qué es el acceso abierto, cabe preguntarnos qué ventajas reporta este nuevo modelo de comunicación científica y para quién.

Es notorio que el acceso abierto tiene repercusiones en cualquier campo de la economía, de la industria y de la tecnología, pero es claramente en el campo de la educación y de la producción científica donde este movimiento genera una evolución que afecta a muchos modelos colaterales de negocio. El acceso abierto

se ha perfilado como un cambio radical en la diseminación de los resultados científicos y de transferencia en el sector de producción hacia la innovación abierta.

El acceso abierto, en la actualidad, es ampliamente apoyado por los gobiernos y los organismos de financiación que muestran actitudes cada vez más favorables a la apertura de datos y al acceso abierto a los contenidos.

No cabe la menor duda de que el acceso abierto es una forma de amortizar la inversión en investigación de un país, al poner a disposición de los usuarios, la documentación derivada de la misma, además de ayudar a la inclusión digital de los ciudadanos al ofrecer los contenidos a texto completo a los investigadores, los docentes, los estudiantes, las instituciones y a toda la ciudadanía.

1.3 El acceso abierto como nuevo modelo de comunicación científica

Esto quiere decir que las dos condiciones para que los contenidos científicos se consideren de acceso abierto son que estos contenidos sean gratuitos y, además, que estén libres de algunas restricciones de derechos de explotación.

El acceso abierto al conocimiento es un movimiento que reclama la difusión y reutilización del conocimiento libremente en internet y esto representa un cambio de modelo en el funcionamiento de la comunicación científica (3).

La vieja tradición es la voluntad de científicos y académicos de publicar los frutos de sus investigaciones en revistas científicas sin remuneración, solo por el bien de la investigación y la difusión del conocimiento. La nueva tecnología es internet. El bien público que las dos hacen posible es la distribución digital a todo el mundo de la literatura científica revisada por expertos, así como el acceso totalmente libre y sin restricciones a ella para todos los científicos, académicos, profesores, estudiantes y otras personas interesadas.

Internet (nueva tecnología) ha propiciado grandes cambios en el acceso a la información, a la cultura, al ocio y al entretenimiento, ofreciéndonos la oportunidad de construir una representación global e interactiva del conocimiento humano, incluyendo el patrimonio cultural, y una perspectiva de acceso mundial [4].

Los efectos generados tanto por internet y como por la digitalización de contenidos unidos al sistema de comunicación científica han sido fundamentales para facilitar el desarrollo del acceso abierto, haciendo posible que los contenidos científicos digitalizados sean difundidos de forma instantánea y a bajo coste a través de internet.

Lo deseable sería lograr que el cambio de modelo en la comunicación científica fuese total, es decir que toda la ciencia, todo el conocimiento estuviese disponible

libremente en internet, pero para ello habría que conseguir que todos los autores publicaran en revistas de acceso abierto.

El acceso abierto se convirtió en una realidad cuando un gran número de instituciones unieron sus fuerzas para promover la libre disseminación de la producción científica y empujaron a las administraciones públicas a crear repositorios digitales que pudieran ser consultados libremente.

1.4 Las políticas que apoyan el acceso abierto

Las políticas a favor de acceso abierto que se están desarrollando por parte de los gobiernos, instituciones y organismos de financiación nacionales y europeos contribuyen a que se multiplique el impacto de la investigación producida en los diferentes organismos científicos. El acceso abierto es un deber de las instituciones públicas que destinan sus presupuestos a la investigación. Los resultados de la investigación financiada con presupuestos públicos deben ser también públicos, asegurando así el retorno de la inversión en investigación y estableciendo como prioridad que sea la institución financiadora quien obtenga el rendimiento económico y académico de su investigación (5).

Los resultados de la investigación financiada con presupuestos públicos deben ser también públicos, asegurando sí el retorno de la inversión en investigación y estableciendo como prioridad que sea la institución financiadora quien obtenga el rendimiento económico y académico de su investigación. La inversión pública en investigación no puede ser rentabilizada por empresas privadas antes que por organismos públicos.

1.5 Un elemento clave en el acceso abierto: los repositorios

El archivo de documentos en repositorios constituye la ruta verde para alcanzar el acceso abierto al conocimiento. Los repositorios no publican documentos, hacen pública documentación a menudo ya publicada en revistas científicas o por los canales editoriales habituales.

De las muchas definiciones de lo qué es un repositorio, preferimos la definición que proporciona Abadal al definirlo como un sitio web que recoge, preserva y difunde la producción académica de una institución (repositorios institucionales) o de una disciplina científica (repositorios temáticos), permitiendo el acceso a los objetos digitales que contiene y a sus metadatos (3).

Los objetivos primordiales de los repositorios son los de favorecer la difusión de los contenidos académicos de la institución a la que sirven o de la disciplina a la que se

dedican, dar visibilidad a la investigación realizada por la institución y sus miembros y facilitar la conservación y preservación de los documentos generados por una institución. Un repositorio institucional es, sobre todo, además de un depósito donde almacenar y preservar documentos digitales, la imagen de la producción científica y académica de su institución.

Los directorios de repositorios son una excelente fuente de información para conocer los archivos abierto de las organizaciones (6).

1.6 La interoperabilidad y visibilidad de los repositorios

Una característica primordial de los repositorios es que sean interoperables. Los diferentes repositorios institucionales siguen protocolos o estándares técnicos que posibilitan su consulta desde el mismo portal del repositorio de forma individual, o a través de otros portales o sitios como son los recolectores que actúan como metabuscadores de documentación científica (8).

La normalización en cuanto a la descripción de los documentos digitales y la interconexión de servidores permite que los datos de los documentos depositados en los repositorios científicos se localicen desde buscadores y plataformas de recuperación de información.

Los documentos científicos se depositan en un repositorio concreto, por ejemplo, en Gredos, pero los datos descriptivos y de localización se comparten, de forma que la visibilidad de un documento en acceso abierto se multiplica exponencialmente, así como el posible impacto de sus contenidos.

1.7 Introducción a Git

Git es un sistema de control de versiones distribuido que se diferencia del resto en el modo en que modela sus datos. La mayoría de los demás sistemas almacenan la información como una lista de cambios en los archivos, mientras que Git modela sus datos más como un conjunto de instantáneas de un mini sistema de archivos.

El seguimiento de los archivos se efectuará para la protección de los cambios. En efecto, cuando haya terminado de editar un archivo, indicará al VCS que ha terminado el trabajo y la razón para hacerlo. Las modificaciones efectuadas en el archivo son registradas por el sistema (DAUZON, Mayo 2018).

Cuando se habla de sistema de gestión de versiones, el neófito tiende a considerarlo como un sistema de salvaguarda incremental: Gracias a un sistema de gestión de versiones, los desarrolladores pueden recuperar todas sus versiones anteriores.

Pero Git no solo permite eso, abarca otras muchas posibilidades.

Git permite llevar en paralelo varias versiones del mismo software, por ejemplo, cuando un desarrollador está trabajando en una nueva funcionalidad, pero esta no debe ser integrada en el software final.

Git también sirve como documentación completa. Cada nueva modificación del código irá acompañada de un mensaje. Al cabo de varios años, estos mensajes pueden ser miles y convertirse en interesantes documentos sobre el contexto en el cual fueron realizados los cambios.

Los sistemas de gestión de versiones pueden parecer limitantes al principio, pero se vuelven indispensables cuando el desarrollador les coge el gusto.

1.8 Clasificación

Podemos clasificar los sistemas de control de versiones atendiendo a la arquitectura utilizada para el almacenamiento del código: locales, centralizados y distribuidos (16).

Locales

Los cambios son guardados localmente y no se comparten con nadie. Esta arquitectura es la antecesora de las dos siguientes.

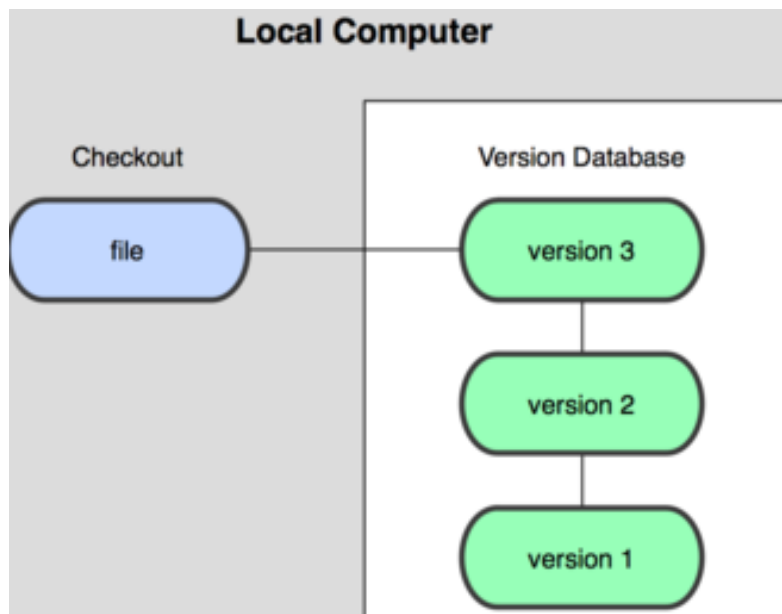


Figura1. Diagrama que representa el sistema de control local de versiones

Centralizados

Existe un repositorio centralizado de todo el código, del cual es responsable un único usuario (o conjunto de ellos). Se facilitan las tareas administrativas a cambio de reducir flexibilidad, pues todas las decisiones fuertes (como crear una nueva rama) necesitan la aprobación del responsable. Algunos ejemplos son CVS y Subversión.

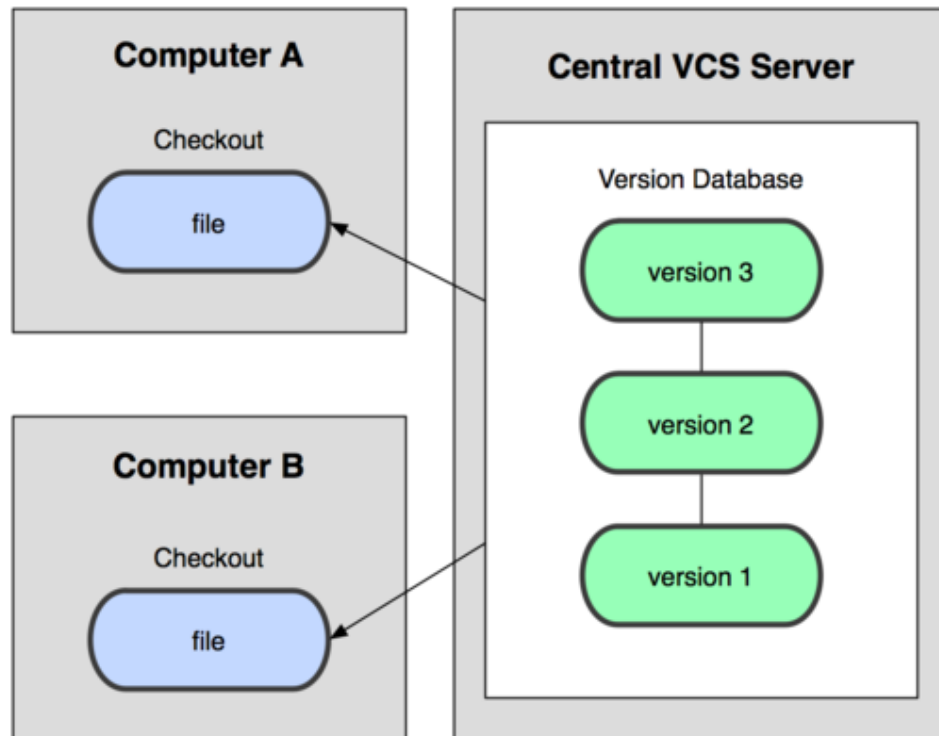


Figura2.Diagrama donde intervienen varias personas y cada una realiza una tarea determinada.

Distribuidos

Cada usuario tiene su propio repositorio. Los distintos repositorios pueden intercambiar y mezclar revisiones entre ellos. Es frecuente el uso de un repositorio, que está normalmente disponible, que sirve de punto de sincronización de los distintos repositorios locales. Ejemplos: Git y Mercurial.

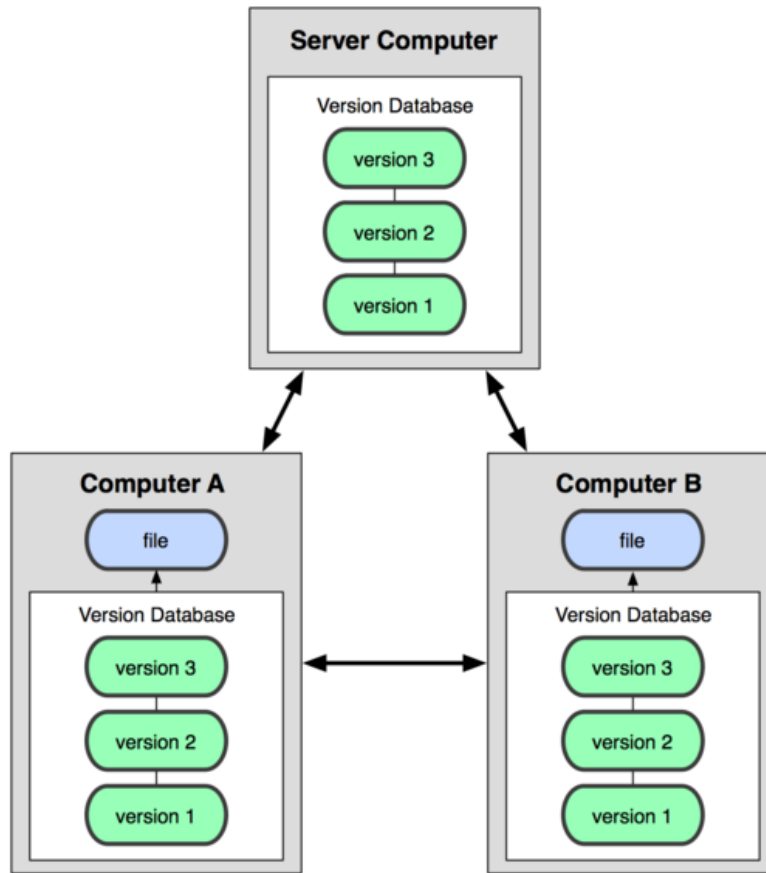


Figura 3. Diagrama donde cada usuario tiene su propio repositorio

Ventajas de sistemas distribuidos

- No es necesario estar conectado para guardar cambios.
- Posibilidad de continuar trabajando si el repositorio remoto no está accesible.
- El repositorio central está más libre de ramas de pruebas.
- Se necesitan menos recursos para el repositorio remoto.
- Más flexibles al permitir gestionar cada repositorio personal como se quiera.

1.9 Github

Github es lo que se denomina una forja, un repositorio de proyectos que usan Git como sistema de control de versiones. Es la forja más popular, ya que alberga más de 10 millones de repositorios. Debe su popularidad a sus funcionalidades sociales, principalmente dos: la posibilidad de hacer forks de otros proyectos y la posibilidad de cooperar aportando código para arreglar errores o mejorar el código. Si bien, no es que fuera una novedad, sí lo es lo fácil que resulta hacerlo (10). A raíz de este proyecto han surgido otros como Gitorius o Gitlab, pero Github sigue siendo el más popular y el que tiene mejores y mayores características. algunas de estas son:

- ❖ Un wiki para documentar el proyecto, que usa Mark Down como lenguaje de marca.
- ❖ Un portal web para cada proyecto.
- ❖ Funcionalidades de redes sociales como followers.
- ❖ Gráficos estadísticos.
- ❖ Revisión de código y comentarios.
- ❖ Sistemas de seguimiento de incidencias.

Los repositorios son archivos donde se almacenan recursos digitales de manera que estos pueden ser accesibles a través de internet.

Existen tres tipos principales de repositorios:

Repositorios institucionales: son los creados por las propias organizaciones para depositar, usar y preservar la producción científica y académica que generan. Supone un compromiso de la institución con el acceso abierto al considerar el conocimiento generado por la institución como un bien que debe estar disponible para toda la sociedad.

Repositorios temáticos: son los creados por un grupo de investigadores, una institución, etc. que reúnen documentos relacionados con un área temática específica.

Repositorios de datos: repositorios que almacenan, conservan y comparten los datos de las investigaciones.

1.10 Importancia GitHub

En los últimos años los Repositorios Institucionales (RIs) han cobrado importancia en la sociedad académica y científica, porque representan una fuente de información digital especializada, organizada y accesible para los lectores de diversas áreas. Los RIs son sistemas informáticos dedicados a gestionar los trabajos científicos y académicos de diversas instituciones de forma libre y gratuita, es decir, siguiendo las premisas del movimiento Open Access (OA).

Nuestra investigación está centrada en revisar los repositorios de dataset en la Ingeniería y sus implicaciones educativas, fortaleciendo a los repositorios como una herramienta más en la formación del futuro ingeniero, además de lograr la preservación de esos documentos y datos en el tiempo, garantizando su acceso a futuras generaciones.

El OA se entiende como el acceso inmediato a trabajos académicos, científicos, o de cualquier otro tipo sin requerimientos de registro, suscripción o pago. Por ello, este movimiento y los repositorios están ayudando a transformar el proceso de publicación de artículos científicos (1), permitiendo el acceso instantáneo o inmediato a las publicaciones arbitradas, gracias a las diferentes aplicaciones (Google Scholar, Microsoft Academic, Arxiv, Repositorios Institucionales de las Universidades) y servicios informáticos (alertas a partir de criterios previamente definidos, RSS, listas de correos, las diferentes redes sociales, etc.).

1.11 Los tres estados de Git

Git tiene tres estados principales en los que se pueden encontrar tus archivos: confirmado (committed), modificado (modified), y preparado (staged). Confirmado significa que los datos están almacenados de manera segura en tu base de datos local. Modificado significa que has modificado el archivo, pero todavía no lo has confirmado a tu base de datos. Preparado significa que has marcado un archivo modificado en su versión actual para que vaya en tu próxima confirmación (9).

Esto nos lleva a las tres secciones principales de un proyecto de Git: el directorio de Git (Git directory), el directorio de trabajo (working directory), y el área de preparación (staging área).

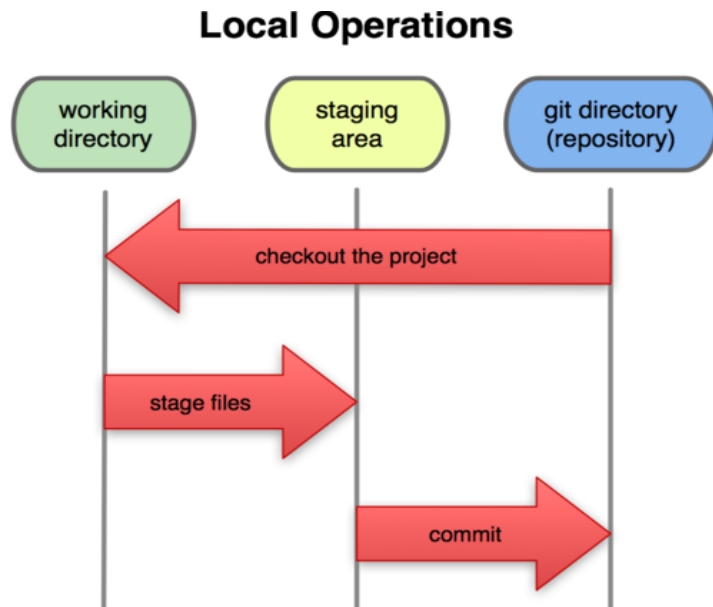


Figura 4. Flujo de trabajo en Git

1.12 Flujos de trabajo distribuidos con git

Hemos visto en qué consiste un entorno de control de versiones distribuido, pero más allá de la simple definición, existe más de una manera de gestionar los repositorios.

Estos son los flujos de trabajo más comunes en Git.

1.13 Flujo de trabajo centralizado

Existe un único repositorio o punto central que guarda el código y todo el mundo sincroniza su trabajo con él. Si dos desarrolladores clonan desde el punto central, y ambos hacen cambios; tan solo el primero de ellos en enviar sus cambios de vuelta lo podrá hacer limpiamente. El segundo desarrollador deberá fusionar previamente su trabajo con el del primero, antes de enviarlo, para evitar el sobrescribir los cambios del primero.

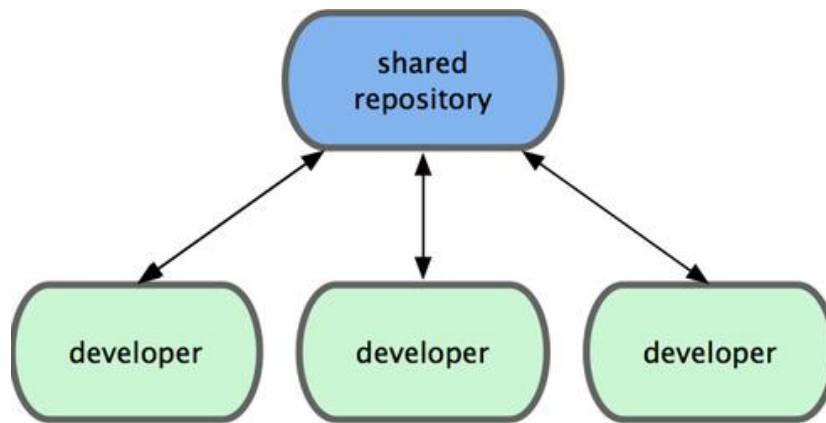


Figura 5. Flujo centralizado

1.14 Flujo de trabajo del Gestor de Integraciones

Al permitir múltiples repositorios remotos, en Git es posible tener un flujo de trabajo donde cada desarrollador tenga acceso de escritura a su propio repositorio público y acceso de lectura a los repositorios de todos los demás. Habitualmente, este escenario suele incluir un repositorio canónico, representante "oficial" del proyecto.

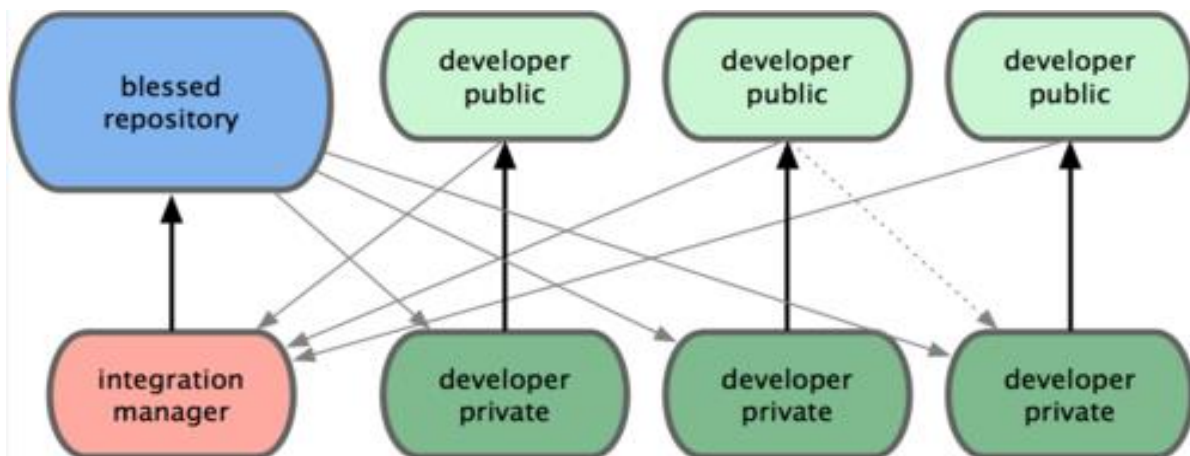


Figura 6. Gestor de integraciones

1.15 Flujo de trabajo con Dictador y tenientes

Es una variante del flujo de trabajo con múltiples repositorios. Se utiliza generalmente en proyectos muy grandes, con cientos de colaboradores. Un ejemplo muy conocido es el del kernel de Linux. Unos gestores de integración se encargan de partes concretas del repositorio; y se denominan tenientes. Todos los tenientes rinden cuentas a un gestor de integración; conocido como el dictador benevolente. El repositorio del dictador benevolente es el repositorio de referencia, del que recuperan (pull) todos los colaboradores.

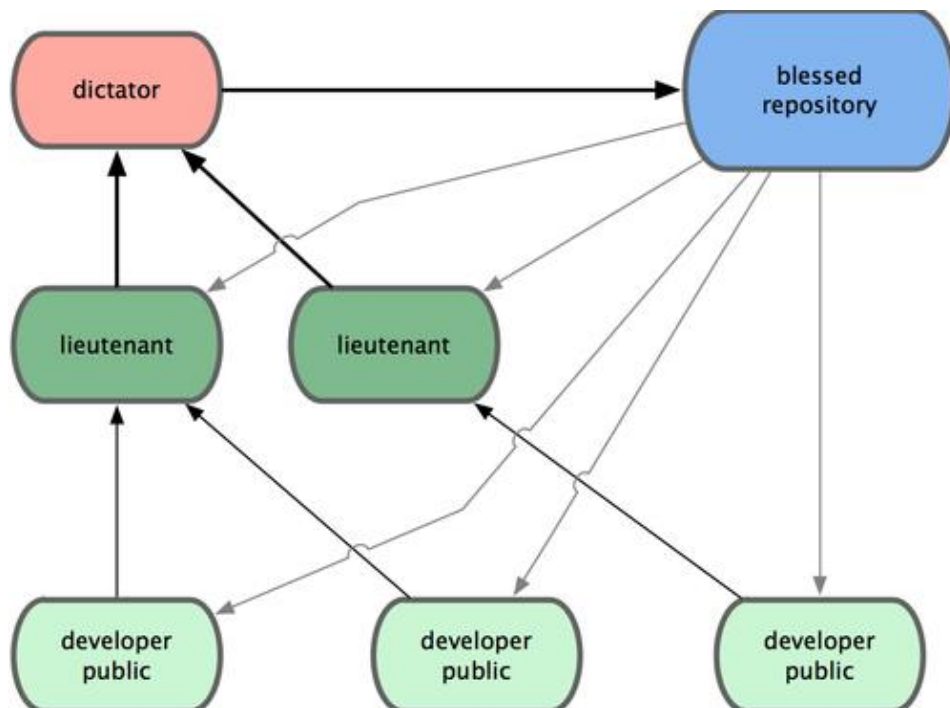


Figura 7. Flujo distribuido

CAPÍTULO 2

NECESIDADES DE UN REPOSITORIO

Los repositorios son sistemas de información que preservan y organizan materiales científicos y académicos como apoyo a la investigación y el aprendizaje, a la vez que garantizan el acceso a la información.

Los recursos educativos abiertos tienen relación con la enseñanza, el aprendizaje, la evaluación y la investigación. Su principal característica es estar disponibles para el uso de profesores y estudiantes de todo el mundo. Estos recursos son abiertos ya que permiten su consulta libre; incluyen: cursos, materiales educativos, libros de texto, vídeos, pruebas y cualquier otra herramienta de diferentes disciplinas con el propósito de apoyar el acceso al conocimiento y el uso de estrategias educativas innovadoras (11).

Las Universidades y bibliotecas de investigación de todo el mundo utilizan los Repositorios Institucionales del siguiente modo (Mary R. Barton, 2005):

- ❖ Comunicación académica.
- ❖ Conservación de materiales de aprendizaje y de cursos.
- ❖ Publicaciones electrónicas.
- ❖ Organización de las colecciones de documentos de investigación.
- ❖ Conservación de materiales digitales a largo plazo.
- ❖ Aumento del prestigio de la Universidad exponiendo sus investigaciones académicas.
- ❖ Relevancia institucional del papel de la biblioteca.
- ❖ Conocimiento sobre la dirección.
- ❖ Evaluación sobre la investigación.
- ❖ Animación a la creación de un acceso abierto a la investigación académica.
- ❖ Conservación de colecciones digitalizadas.

2.1 GitHub aplicado a la docencia

Ingeniería Web es una asignatura de 6 créditos ECTS del cuarto curso del grado de Ingeniería Informática de la Universidad de Zaragoza. La asignatura es parte del itinerario de Ingeniería del Software. A lo largo de la asignatura el alumno debe desarrollar en solitario o en grupo varios sistemas Web orientados a servicios, desarrollando así competencias relacionadas con el desarrollo Web. Ingeniería Web es una asignatura de evaluación continua en la que el trabajo práctico pondera un 80 % en la nota final. A continuación, describimos herramientas y funcionalidades de GitHub utilizadas en este curso distinguiendo entre las derivadas de Git y las exclusivas de GitHub. Las herramientas y funcionalidades se presentan agrupadas bajo tres criterios. El primero atiende a su uso por parte del alumno en su aprendizaje. El segundo se fija en su papel como soporte del curso. Finalmente, el tercero denominado facilitadores describe aspectos relacionados con tecnologías, licencias y precios que han facilitado el diseño curso (14).

2.2 Aprendizaje del alumno

Dentro de esta categoría incluimos las herramientas y funcionalidades relacionadas con actividades de comunicación, productividad y participación de los alumnos. Las derivadas de Git son:

1. **Diario de actividades:** El histórico de cambios de un repositorio hospedado en GitHub es accesible vía Web, pudiendo accederse a todos los cambios que se han producido con indicación de quién lo ha cambiado, cuándo se ha cambiado, qué ha cambiado y el origen del cambio (otro repositorio o vía un editor Web proporcionado por GitHub).
2. **Trabajo fuera de línea:** Implícito por soportar Git, pero restringido a los recursos gestionados en los repositorios por Git. No hay soporte fuera de línea a las herramientas de trabajo colaborativo de GitHub.
3. **Trabajo en grupo:** El uso de Git y GitHub permite implementar diferentes flujos de trabajo en entornos educativos.

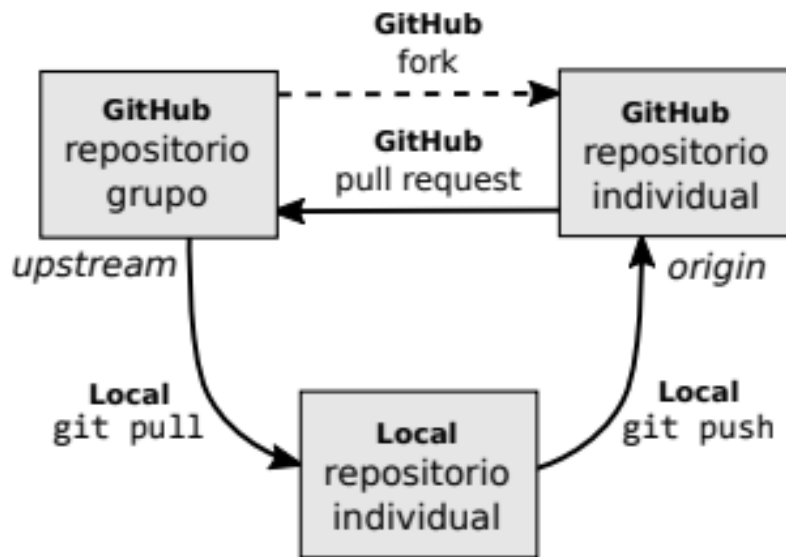


Figura 8. Flujo de trabajo en grupo utilizando GitHub/Git

La relevancia del trabajo en grupo en esta asignatura hace que merezca la pena detallar su implementación. Primero hay que preparar los repositorios remotos compartidos de los grupos y los repositorios remotos y locales de los alumnos.






GitHub actúa como la máquina remota donde se crean los repositorios remotos compartidos. A continuación, cada alumno clona en GitHub el repositorio remoto compartido de su grupo para crear su repositorio remoto

El repositorio remoto del alumno se referencia localmente en Git como origin y el del grupo como upstream. Una vez que está preparada toda la infraestructura, el flujo de trabajo esperado de cada alumno sigue los siguientes pasos: integrar los cambios del repositorio del grupo en su repositorio local (`git pull --rebase upstream`), realizar los cambios pertinentes, enviar dichos cambios a su repositorio individual remoto (`git push --force origin`) y, finalmente, solicitar vía un pull request que los cambios de su repositorio remoto se integren en el repositorio remoto del grupo. Dentro del grupo hay un alumno con el rol gestión del repositorio remoto. Este alumno es el que decide si se integra el cambio o no en el repositorio del grupo.

La única herramienta de GitHub que puede ser considerada de aprendizaje no asociada con Git y de la cual se tiene constancia de su uso por parte de los alumnos son los micro foros de discusión. Los alumnos que han iniciado sesión en GitHub pueden añadir vía Web comentarios a cualquier cambio y a los issues asociados a un repositorio público.

Esta herramienta se ha utilizado principalmente en la interacción profesor-alumno durante las entregas.

2.3 Las herramientas y funcionalidades exclusivas de GitHub son:

-  **Transparencia:** señala que la disponibilidad en GitHub de pistas visibles como el volumen de actividad, la actividad a lo largo del tiempo, la relevancia del flujo de cambios, y la información detallada son herramientas útiles para resolver problemas de coordinación y comunicación. Aplicada a la gestión del curso, la transparencia ha resuelto problemas de coordinación y comunicación entre los profesores y los alumnos, y entre los alumnos entre sí.
-  **Publicación de contenido:** GitHub genera automáticamente un sumario en HTML del contenido del repositorio o de una carpeta si existe un fichero denominado README o README.md (.md es la extensión que corresponde al lenguaje de marcado ligero Markdown).
-  **Edición en línea de contenido:** GitHub permite editar contenido vía un editor Web. Según el tipo de extensión, por ejemplo .md, permite su pre visualización.
-  **Seguimiento de los alumnos:** Es posible acceder a diferentes visiones de la actividad de los alumnos durante el curso en forma de diferentes tipos de informes: informes globales de actividad, seguimiento del trabajo en equipo, informes comparativos de actividad e informes individuales de actividad.
-  **Plataforma de entrega con realimentación:** La combinación del uso de pull requests como herramienta de entrega junto con los micro foros de discusión permiten controlar la entrega y realimentar al alumno con el resultado. Este procedimiento es uno de los recomendados por GitHub cuando se utiliza como plataforma de entrega en la docencia (Sawers, Febrero, 2014)

CAPITULO 3

METODOLOGÍA

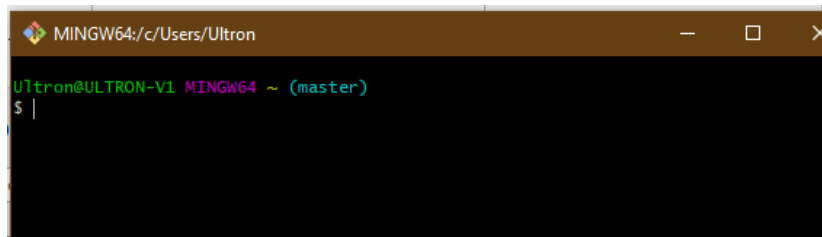
Metodología de desarrollo de software en ingeniería de software es un marco de trabajo usado para estructurar, planificar y controlar el proceso de desarrollo en sistemas de información (17).

Básicamente el punto aquí es controlar el proceso de desarrollo para que sea ordenado y de esta forma lo más productivo posible.

Este método de desarrollo estaría basado en Git y Github ofreciéndonos una enorme versatilidad y posibilitando que todo el equipo de desarrollo esté al día en cuanto a novedades en el desarrollo, así como multitud de ventajas que más adelante explicaré. Esta metodología estaría basada en las branches y pull request.

3.1 Uso básico para trabajar con Git y GitHub

Empezar a trabajar con Git y GitHub no es nada complicado. Lo primero que debemos hacer es realizar la instalación de Git en nuestro equipo; dependiendo del tipo de sistema operativo que utilicemos, descargaremos una versión u otra (18). Una vez instalado, será necesario realizar una configuración básica en la que indicaremos nuestro nombre y correo electrónico. Para ello ejecutamos la aplicación Git, con lo que abriremos el terminal desde donde ejecutaremos todas las instrucciones necesarias.



Lo primero será indicar nuestro nombre y para ello ejecutaremos la siguiente instrucción en el terminal.

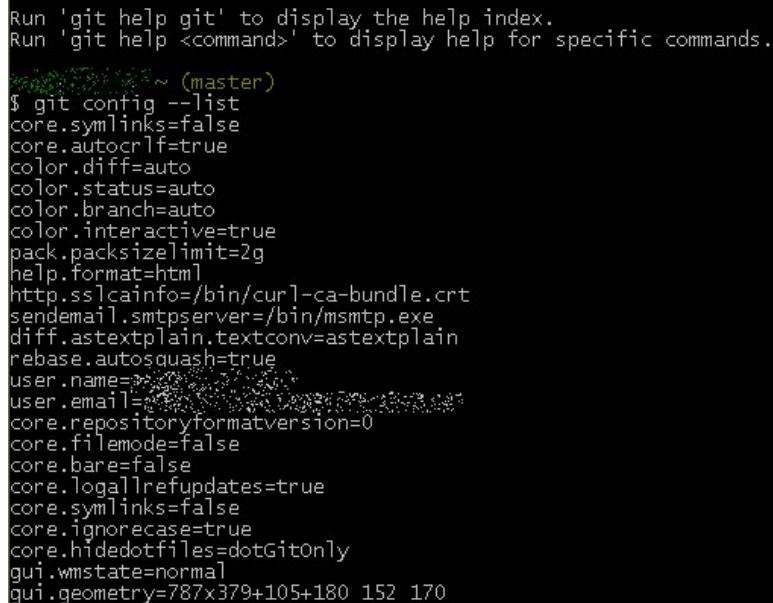
git config --global user.name "xxxxxxx"

Lo siguiente será configurar nuestro correo electrónico, y la instrucción para esto será.

git config --global user.email loquesea@tudominio.com

Si queremos ver la configuración, bastará con ejecutar lo siguiente.

git config --list



```
Run 'git help git' to display the help index.
Run 'git help <command>' to display help for specific commands.

~ (master)
$ git config --list
core.symlinks=false
core.autocrlf=true
color.diff=auto
color.status=auto
color.branch=auto
color.interactive=true
pack.packsizelimit=2g
help.format=html
http.sslcainfo=/bin/curl-ca-bundle.crt
sendemail.smtpserver=/bin/msmtp.exe
diff.astextplain.textconv=astextplain
rebase.autosquash=true
user.name=
user.email=
core.repositoryformatversion=0
core.filemode=false
core.bare=false
core.logallrefupdates=true
core.symlinks=false
core.ignorecase=true
core.hidedotfiles=dotGitOnly
gui.wmstate=normal
gui.geometry=787x379+105+180 152 170
```

Una vez configurados esos datos, es hora de empezar a trabajar.

3.2 Descargar un proyecto por primera vez

Para descargarnos un proyecto por primera vez, hay que utilizar el comando “git clone” seguido de la url de github que corresponde al proyecto.

git clone <https://github.com/acens/xxxxxxx.git>

Cada proyecto tendrá una ruta distinta que asignará GitHub por defecto. Al ejecutar la instrucción anterior, lo que hemos conseguido es hacer una copia del proyecto que estuviese subido en GitHub a nuestro equipo local (18).

3.3 Ver los cambios realizados en el repositorio

Si queremos ver los cambios que hemos ido haciendo en el repositorio, lo que se utiliza es el comando “git status”.

```
$ git status
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   app/SymfonyRequirements.php
#       modified:   app/check.php
#       modified:   bin/doctrine
#       modified:   bin/doctrine.php
#       modified:   web/config.php
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       __MACOSX/
#       bin/doctrine.bat
#       bin/doctrine.php.bat
#       composer.lock
#       git taus
#       import.zip
#       web/assetic/
#       web/css/
#       web/import/
#       web/js/
no changes added to commit (use "git add" and/or "git commit -a")
```

En este caso nos encontramos dos apartados bien diferenciados:

- ❖ **Changes not staged for commit:** Se corresponde con archivos ya existentes en el repositorio y que han sido modificados
- ❖ **Untraked files:** En este caso corresponden a nuevos archivos que aún no han sido subidos al repositorio.

3.4 Añadir cambios

Si queremos subir un archivo modificado al repositorio alojado en GitHub, lo primero que deberemos hacer es indicarlo mediante la instrucción “git add”, seguido de la ruta hasta llegar al archivo que queremos subir.

git add ruta_archivo_subir

Si ahora volvemos a ejecutar el comando “git status”, este archivo nos aparecerá en otra sección llamada “changes to be committed”.

```
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       modified:   src/Tuinper/PublicBundle/Controller/DefaultController.php
#
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   app/SymfonyRequirements.php
#       modified:   app/check.php
#       modified:   bin/doctrine
#       modified:   bin/doctrine.php
#       modified:   web/config.php
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       __MACOSX/
#       bin/doctrine.bat
#       bin/doctrine.php.bat
#       composer.lock
#       git taus
#       import.zip
#       web/assetic/
#       web/css/
#       web/import/
#       web/js/
```

3.5 Enviar los cambios al repositorio alojado en GitHub

Con el paso anterior, lo que hemos hecho ha sido indicar los archivos que queríamos subir al repositorio, pero aún no han sido subido ya que para ello hay que ejecutar la instrucción “git commit –m ‘mensaje explicativo’”.

El apartado del mensaje, consiste en una breve explicación de los cambios realizados.

```
$ git commit -m "cambios en el defaultcontroller"
[master 80f3dd0] cambios en el defaultcontroller
1 file changed, 1 insertion(+)
```

Por último, ejecutaremos el comando “**git push**”

```
$ git push
warning: push.default is unset; its implicit value is changing in
Git 2.0 from 'matching' to 'simple'. To squelch this message
and maintain the current behavior after the default changes, use:

    git config --global push.default matching

To squelch this message and adopt the new behavior now, use:

    git config --global push.default simple

See 'git help config' and search for 'push.default' for further information.
(the 'simple' mode was introduced in Git 1.7.11. Use the similar mode
'current' instead of 'simple' if you sometimes use older versions of Git)
Username for 'https://github.com':
```

Para lanzar los cambios, habrá que indicar nuestro usuario y password de GitHub.

3.6 Descargar los últimos cambios

Si por el contrario lo que queremos es descargarnos los últimos cambios subidos por cualquier otro miembro del equipo al repositorio, utilizaremos el comando “**git pull**”. Al igual que en el caso anterior, habrá que indicar nuestro usuario y contraseña para realizar la descarga de esa información.

```
$ git pull
Username for 'https://github.com': angelcarrero
Password for 'https://angelcarrero@github.com':
remote: Counting objects: 13432, done.
Receiving objects: 72% (9775/13432), 49.79 MiB | 318.00 KiB/s
```

Gracias a la combinación de estas dos herramientas, trabajar en equipo para el desarrollo de software es mucho más sencillo y seguro, ya que podremos volver a una versión anterior en cualquier momento.

3.7 Crear un proyecto

Crear un programa "Hola Mundo"

Creamos un directorio donde colocar el código

```
$ mkdir curso-de-git
```

```
$ cd curso-de-git
```

Creamos un fichero hola.php que muestre Hola Mundo.

```
<?php  
echo "Hola Mundo\n";
```

3.8 Crear el repositorio

Para crear un nuevo repositorio se usa la orden git init

```
$ git init  
Initialized empty Git repository in /home/cc0gobas/git/curso-de-git/.git/
```

Añadir la aplicación

Vamos a almacenar el archivo que hemos creado en el repositorio para poder trabajar, después explicaremos para qué sirve cada orden(19).

```
$ git add hola.php  
$ git commit -m "Creación del proyecto"  
[master (root-commit) e19f2c1] Creación del proyecto  
1 file changed, 2 insertions(+)  
create mode 100644 hola.php
```

Comprobar el estado del repositorio

Con la orden git status podemos ver en qué estado se encuentran los archivos de nuestro repositorio.

```
$ git status  
# On branch master  
nothing to commit (working directory clean)
```

Si modificamos el archivo hola.php:

```
<?php
```

```
@print "Hola {$argv[1]}\n";
```

Y volvemos a comprobar el estado del repositorio:

```
$ git status
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#   modified:   hola.php
#
no changes added to commit (use "git add" and/or "git commit -a")
```

Añadir cambios

Con la orden `git add` indicamos a git que prepare los cambios para que sean almacenados.

```
$ git add hola.php
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#   modified:   hola.php
#
```

3.9 Confirmar los cambios

Con la orden `git commit` confirmamos los cambios definitivamente, lo que hace que se guarden permanentemente en nuestro repositorio.

```
1 $ git commit -m "Parametrización del programa"
2 [master efc252e] Parametrización del programa
3 1 file changed, 1 insertion(+), 1 deletion(-)
4 $ git status
5 # On branch master
6 nothing to commit (working directory clean)
```

Diferencias entre `workdir` y `staging`

Modificamos nuestra aplicación para que soporte un parámetro por defecto y añadimos los cambios.

```
<?php
$nombre = isset($argv[1]) ? $argv[1] : "Mundo";
@print "Hola, {$nombre}\n";
```

Esta vez añadimos los cambios a la fase de `staging`, pero sin confirmarlos (`commit`).

`git add hola.php`

volvemos a modificar el programa para indicar con un comentario lo que hemos hecho.

```
<?php
// El nombre por defecto es Mundo
$nombre = isset($argv[1]) ? $argv[1] : "Mundo";
@print "Hola, {$nombre}\n";
```

Y vemos el estado en el que está el repositorio

```
$ git status
# On branch master
```

```
# Changes to be committed:
# (use "git reset HEAD <file>..." to unstage)
#
# modified: hola.php
#
# Changes not staged for commit:
# (use "git add <file>..." to update what will be committed)
# (use "git checkout -- <file>..." to discard changes in working directory)
#
# modified: hola.php
#
```

Podemos ver como aparecen el archivo hola.php dos veces. El primero está preparado para ser confirmado y está almacenado en la zona de staging. El segundo indica que el directorio hola.php está modificado otra vez en la zona de trabajo (workdir).

Almacenamos los cambios por separado:

```
$ git commit -m "Se añade un parámetro por defecto"
[master 3283e0d] Se añade un parámetro por defecto
1 file changed, 2 insertions(+), 1 deletion(-)
$ git status
# On branch master
# Changes not staged for commit:
# (use "git add <file>..." to update what will be committed)
# (use "git checkout -- <file>..." to discard changes in working directory)
#
# modified: hola.php
```



```
#  
no changes added to commit (use "git add" and/or "git commit -a")  
$ git add .  
$ git status  
# On branch master  
# Changes to be committed:  
#   (use "git reset HEAD <file>..." to unstage)  
#  
#   modified:   hola.php  
#  
$ git commit -m "Se añade un comentario al cambio del valor por defecto"  
[master fd4da94] Se añade un comentario al cambio del valor por defecto  
1 file changed, 1 insertion(+)
```

3.10 Ignorando archivos

La orden **git add .** o **git add nombre_directorio** es muy cómoda, ya que nos permite añadir todos los archivos del proyecto o todos los contenidos en un directorio y sus subdirectorios. Es mucho más rápido que tener que ir añadiéndolos uno por uno. El problema es que, si no se tiene cuidado, se puede terminar por añadir archivos innecesarios o con información sensible (19).

Por lo general se debe evitar añadir archivos que se hayan generado como producto de la compilación del proyecto, los que generen los entornos de desarrollo (archivos de configuración y temporales) y aquellos que contentan información sensible, como contraseñas o tokens de autenticación. Por ejemplo, en un proyecto de C/C++, los archivos objeto no deben incluirse, solo los que contengan código fuente y los make que los generen.

Para indicarle a git que debe ignorar un archivo, se puede crear un fichero llamado **.gitignore**, bien en la raíz del proyecto o en los subdirectorios que queramos. Dicho fichero puede contener patrones, uno en cada línea, especifiquen qué archivos deben ignorarse.

El formato es el siguiente:

```
# .gitignore
dir1/      # ignora todo lo que contenga el directorio dir1
!dir1/info.txt # El operador ! excluye del ignore a dir1/info.txt (sí se guardaría)
dir2/*.txt  # ignora todos los archivos txt que hay en el directorio dir2
dir3/**/*.*txt # ignora todos los archivos txt que hay en el dir3 y sus
subdirectorios
*.o        # ignora todos los archivos con extensión .o en todos los directorios
```

Cada tipo de proyecto genera sus ficheros temporales, así que para cada proyecto hay un **.gitignore** apropiado. Existen repositorios que ya tienen creadas plantillas.

3.11 Ignorando archivos globalmente

Si bien, los archivos que hemos metido en **.gitignore**, deben ser aquellos ficheros temporales o de configuración que se pueden crear durante las fases de compilación o ejecución del programa, en ocasiones habrá otros ficheros que tampoco debemos introducir en el repositorio y que son recurrentes en todos los proyectos. En dicho caso, es más útil tener un gitignore que sea global a todos nuestros proyectos. Esta configuración sería complementaria a la que ya tenemos. Ejemplos de lo que se puede ignorar de forma global son los ficheros temporales del sistema operativo (*~, .nfs*) y los que generan los entornos de desarrollo.

Para indicar a git que queremos tener un fichero de gitignore global, tenemos que configurarlo con la siguiente orden:

```
git config --global core.excludesfile $HOME/.gitignore_global
```

Ahora podemos crear un archivo llamado **.gitignore_global** en la raíz de nuestra cuenta con este contenido:

Compiled source

#####

***.com
*.class
*.dll
*.exe
*.o
*.so**

Packages

#####

**# it's better to unpack these files and commit the raw source
git has its own built in compression methods
*.7z
*.dmg
*.gz
*.iso
*.jar
*.rar
*.tar
*.zip**

Logs and databases

#####

***.log
*.sql
*.sqlite**

IDEs

#####

**.idea
.settings/
.classpath
.project**

OS generated files

#####

**DS_Store
.DS_Store?
._*
.Spotlight-V100
.Trashes
ehthumbs.db
Thumbs.db
*~
*.swp**

3.11 Observando los cambios

Con la orden git log podemos ver todos los cambios que hemos hecho:

```
$ git log  
commit fd4da946326fbe8b24e89282ad25a71721bf40f6  
Author: Sergio Gómez <sergio@uco.es>  
Date: Sun Jun 16 12:51:01 2013 +0200
```

Se añade un comentario al cambio del valor por defecto

```
commit 3283e0d306c8d42d55ffcb64e456f10510df8177  
Author: Sergio Gómez <sergio@uco.es>  
Date: Sun Jun 16 12:50:00 2013 +0200
```

Se añade un parámetro por defecto

```
commit efc252e11939351505a426a6e1aa5bb7dc1dd7c0  
Author: Sergio Gómez <sergio@uco.es>  
Date: Sun Jun 16 12:13:26 2013 +0200
```

Parametrización del programa

```
commit e19f2c1701069d9d1159e9ee21acaa1bbc47d264  
Author: Sergio Gómez <sergio@uco.es>  
Date: Sun Jun 16 11:55:23 2013 +0200
```

Creación del proyecto

También es posible ver versiones abreviadas o limitadas, dependiendo de los parámetros:

```
$ git log --oneline  
fd4da94 Se añade un comentario al cambio del valor por defecto  
3283e0d Se añade un parámetro por defecto
```

```
efc252e Parametrización del programa
e19f2c1 Creación del proyecto
git log --oneline --max-count=2
git log --oneline --since='5 minutes ago'
git log --oneline --until='5 minutes ago'
git log --oneline --author=sergio
git log --oneline --all
```

Una versión muy útil de git log es la siguiente, pues nos permite ver en qué lugares está master y HEAD, entre otras cosas:

```
$ git log --pretty=format:'%h %ad | %s%d [%an]' --graph --date=short
* fd4da94 2013-06-16 | Se añade un comentario al cambio del valor por defecto
(HEAD, master) [Sergio Gómez]
* 3283e0d 2013-06-16 | Se añade un parámetro por defecto [Sergio Gómez]
* efc252e 2013-06-16 | Parametrización del programa [Sergio Gómez]
* e19f2c1 2013-06-16 | Creación del proyecto [Sergio Gómez]
```

3.12 Recuperando versiones anteriores

Cada cambio es etiquetado por un hash, para poder regresar a ese momento del estado del proyecto se usa la orden git checkout.

```
$ git checkout e19f2c1
Note: checking out 'e19f2c1'.
```

You are in 'detached HEAD' state. You can look around, make experimental changes and commit them, and you can discard any commits you make in this state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may do so (now or later) by using -b with the checkout command again. Example:

```
git checkout -b new_branch_name
```

```
HEAD is now at e19f2c1... Creación del proyecto
$ cat hola.php
<?php
echo "Hello, World\n";
```

El aviso que nos sale nos indica que estamos en un estado donde no trabajamos en ninguna rama concreta. Eso significa que los cambios que hagamos podrían "perderse" porque si no son guardados en una nueva rama, en principio no podríamos volver a recuperarlos. Hay que pensar que Git es como un árbol donde un nodo tiene información de su nodo padre, no de sus nodos hijos, con lo que siempre necesitaríamos información de dónde se encuentran los nodos finales o de otra manera no podríamos acceder a ellos.

3.13 Volver a la última versión de la rama master

Usamos git checkout indicando el nombre de la rama:

```
$ git checkout master
Previous HEAD position was e19f2c1... Creación del proyecto
```

3.14 Etiquetando versiones

Para poder recuperar versiones concretas en la historia del repositorio, podemos etiquetarlas, lo cual es más fácil que usar un hash. Para eso usaremos la orden git tag.

```
$ git tag v1
```

Ahora vamos a etiquetar la versión inmediatamente anterior como v1-beta. Para ello podemos usar los modificadores ^ o ~ que nos llevarán a un ancestro determinado.

Las siguientes dos órdenes son equivalentes:

```
$ git checkout v1^
$ git checkout v1~1
$ git tag v1-beta
```

Si ejecutamos la orden sin parámetros nos mostrará todas las etiquetas existentes.

```
$ git tag  
v1  
v1-beta
```

Y para verlas en el historial:

```
$ git hist master --all  
* fd4da94 2013-06-16 | Se añade un comentario al cambio del valor por defecto  
(tag: v1, master) [Sergio Gómez]  
* 3283e0d 2013-06-16 | Se añade un parámetro por defecto (HEAD, tag: v1-beta)  
[Sergio Gómez]  
* efc252e 2013-06-16 | Parametrización del programa [Sergio Gómez]  
* e19f2c1 2013-06-16 | Creación del proyecto [Sergio Gómez]
```

3.15 Borrar etiquetas

Para borrar etiquetas:

```
git tag -d nombre_etiqueta
```

3.16 Visualizar cambios

Para ver los cambios que se han realizado en el código usamos la orden `git diff`. La orden sin especificar nada más, mostrará los cambios que no han sido añadidos aún, es decir, todos los cambios que se han hecho antes de usar la orden `git add`. Después se puede indicar un parámetro y dará los cambios entre la versión indicada y el estado actual. O para comparar dos versiones entre sí, se indica la más antigua y la más nueva.

CAPÍTULO 4

IMPLEMENTACIÓN DE SEGURIDAD

Una clave de implementación es una clave SSH que se almacena en tu servidor y concede acceso a un único repositorio GitHub. Esta clave se adjunta directamente al repositorio en lugar de una cuenta de usuario personal (20).

4.1 Determinar dónde se ha usado la clave

Para determinar dónde se ha usado la clave, abre una terminal y escribe el comando ssh. Usa la marca -i para obtener la ruta a la clave que deseas verificar:

```
$ ssh -T -i ~/.ssh/id_rsa git@nombre de host
```

```
# Connect to tu instancia de servidor de GitHub Enterprise using a specific  
ssh key
```

```
> Hi username! Has autenticado con éxito, pero GitHub no
```

```
> proporciona acceso al shell.
```

El nombre de usuario que aparece en la respuesta es la cuenta de GitHub Enterprise a la que la clave se encuentra actualmente vinculada. Si la respuesta se parece a "username/repo", la llave se ha vinculado a un repositorio como llave de implementación.

4.2 Administrar las llaves de despliegue

Aprende las diversas formas de administrar llaves SSH en tus servidores cuando automatizas los scripts de despliegue y averigua qué es lo mejor para ti (21).

Puedes administrar llaves SSH en tus servidores cuando automatices tus scripts de despliegue utilizando el reenvío del agente de SSH, HTTPS con tokens de OAuth, o usuarios máquina.

4.3 Reenvío del agente SSH

En muchos casos, especialmente al inicio de un proyecto, el reenvío del agente SSH es el método más fácil y rápido a utilizar. El reenvío de agentes utiliza las mismas llaves SSH que utiliza tu ordenador de desarrollo local.

Pros

- ❖ No tienes que generar o llevar registros de las llaves nuevas.
- ❖ No hay administración de llaves; los usuarios tienen los mismos permisos en el servidor y localmente.
- ❖ No se almacenan las llaves en el servidor, así que, en caso de que el servidor se ponga en riesgo, no necesitas buscar y eliminar las llaves con este problema.

Contras

- Los usuarios deben ingresar cno SSH para hacer los despliegues; no pueden utilizarse los procesos de despliegue automatizados.
- El reenvío del agente SSH puede ser difícil de ejecutar para usuarios de Windows.

Configuración

- 1.- Habilita el reenvío de agente localmente. Consulta nuestra guía sobre el redireccionamiento del agente SSH para obtener más información.
- 2.- Configura tus scripts de despliegue para utilizar el reenvío de agente. Por ejemplo, el habilitar el reenvío de agentes en un script de bash se vería más o menos así: `ssh -A serverA 'bash -s' < deploy.sh`

4.4 Llaves de implementación

Puedes lanzar proyectos desde un repositorio de GitHub Enterprise hacia tu servidor al utilizar una llave de despliegue, la cual es una llave SSH que otorga acceso a un repositorio específico. GitHub Enterprise adjunta la parte pública de la llave directamente en tu repositorio en vez de hacerlo a una cuenta de usuario, y la parte

privada de ésta permanece en tu servidor. Para obtener más información, consulta la sección "Entregar despliegues".

Las claves de despliegue con acceso de escritura pueden llevar a cabo las mismas acciones que un miembro de la organización con acceso administrativo o que un colaborador en un repositorio personal. Para obtener más información, consulta las secciones "Niveles de permiso en los repositorios para una organización" y "Niveles de permiso para un repositorio de una cuenta de usuario".

Pros

- Cualquiera que tenga acceso al repositorio y al servidor tiene la capacidad de desplegar el proyecto.
- Los usuarios no tienen que cambiar su configuración local de SSH.

Las llaves de despliegue son de solo lectura predeterminadamente, pero les puedes otorgar acceso de escritura cuando las agregas a un repositorio.

Contras

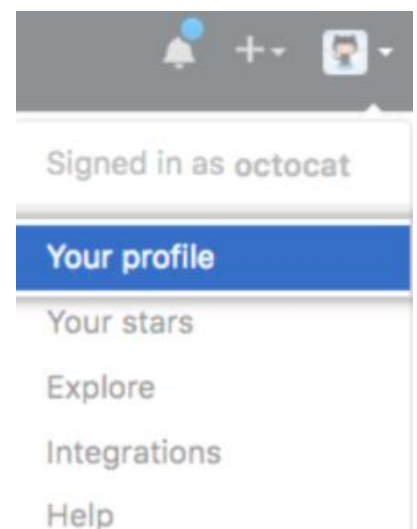
Las llaves de despliegue solo otorgan acceso a un solo repositorio. Los proyectos más complejos pueden tener muchos repositorios que extraer del mismo servidor.

Las llaves de lanzamiento habitualmente no están protegidas con una frase de acceso, lo cual hace que se pueda acceder fácilmente a ellas si el servidor estuvo en riesgo (21).

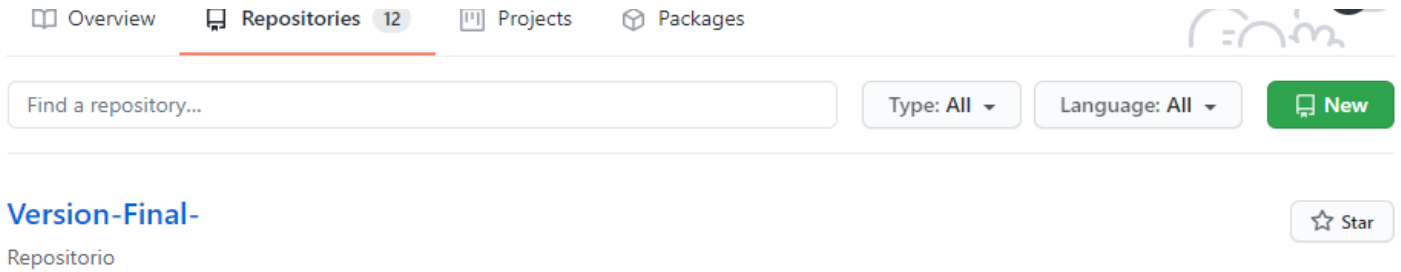
Configuración

Ejecuta el procedimiento de ssh-keygen en tu servidor, y recuerda en donde guardaste el par de llaves pública/privada de rsa.

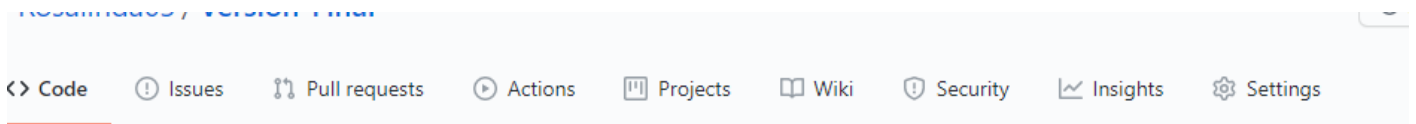
En la esquina superior derecha de cualquier página de GitHub Enterprise, da clic en tu foto de perfil y luego da clic en Tu perfil.



1. En tu página de perfil, da clic en **Repositorios** y luego en el nombre de tu repositorio.



2. Desde tu repositorio, da clic en **Configuración**.



3. En la barra lateral, da clic en **Desplegar llaves** y luego en **Agregar llave de despliegue**.



Proporciona un título, pégalo en tu llave publica

<> Code Issues 0 Pull requests 1 Projects 0 Wiki Insights Settings

Options
Collaborators
Branches
Webhooks
Integrations & services
Deploy keys

Deploy keys / Add new

Title

MyNewKey

Key

```
ssh-rsa  
**ThisisOnlyaSampleDeployKey**ABCDEFGHIJKLMNOPQRSTUVWXYZ1234  
DAQnfW/NCcUpmTDoyNd3d41fmcWxWA46qedJdlekfHLxdoYcyqxSHi+6nr  
U4z0A0TQi5y41wQ4aMGNDHyyA9+mfK0Xe3Hb7oP2IXS/y8I4iW72robMMc  
7QsPneGOADHYT1Lv4+zN4OjOcRi6jdHA46qedJdlekfHLxdoYcyqxSHiWm  
oGi6y+6nm6rxeKfcYzE/ZzjUcxN9LqTkmdU4z0A0TQi5y41wQ4aMGNDHyy/  
72robMMdjGYnzjUpQ6zIKB3DI+v+fR+iW4gj61tYPH==octocat@github.com
```

☐ **Allow write access**
Can this key be used to push to this repository? Deploy keys always have pull access.

Add key

4. Selecciona **Permitir acceso de escritura** si quieres que esta llave tenga acceso de escritura en el repositorio. Una llave de despliegue con acceso de escritura permite que un despliegue cargue información al repositorio.
5. Da clic en **Agregar llave**.

4.5 Utilizar el reenvío del agente SSH

Para simplificar los despliegues en un servidor, puedes configurar el reenvío del agente SSH para utilizar las llaves SSH locales de forma segura.

El reenvío del agente de SSH puede utilizarse para hacer despliegues a un servidor simple. Te permite utilizar llaves SSH locales en vez de dejar las llaves (¡sin frases de acceso!) en tu servidor.

Si ya configuraste una llave SSH para que interactúe con GitHub Enterprise, probablemente estás familiarizado con el ssh-agent. Es un programa que se ejecuta en segundo plano y que mantiene tu llave cargada en la memoria para que no tengas que ingresar tu frase de acceso cada que quieres utilizar esta llave. Lo ingenioso de esto es que puedes elegir dejar que los servidores accedan a tu ssh-agent local como si ya se estuvieran ejecutando en el servidor (22). Esto es como pedirle a un amigo que ingrese su contraseña para que puedas utilizar su computadora.

4.6 Configurar el reenvío del agente SSH

Asegúrate de que tu propia llave SSH está configurada y funciona.

Puedes probar que tu llave local funciona si ingresas `ssh -T git@github.com` en la terminal:

```
$ ssh -T git@github.com
```

```
# Attempt to SSH in to github
```

```
> Hi username! You've successfully authenticated, but GitHub does not provide
```

```
> shell access.
```

Estamos empezando muy bien. Vamos a configurar SSH para permitir el reenvío del agente en tu servidor.

1. Utilizando tu editor de texto preferido, abre el archivo en `~/.ssh/config`. Si este archivo no existe, puedes crearlo si ingresas `touch ~/.ssh/config` en la terminal.
2. Ingresa el siguiente texto en el archivo, reemplazando `example.com` con el nombre de dominio o la IP de tu servidor:

```
Host example.com
```

```
ForwardAgent yes
```

4.7 Probar el reenvío del agente SSH

Para probar que el reenvío del agente funciona en tu servidor, puedes ingresar con SSH en tu servidor y ejecutar `ssh -T git@github.com` una vez más. Si todo sale bien, te regresará el mismo mensaje que salió cuando lo hiciste localmente.

Si no estás seguro de que se esté utilizando tu llave local, también puedes inspeccionar la variable `SSH_AUTH_SOCK` en tu servidor:

```
$ echo "$SSH_AUTH_SOCK"
```

```
# Print out the SSH_AUTH_SOCK variable
```

```
> /tmp/ssh-4hNGMk8AZX/agent.79453
```

Si no se ha configurado la variable, esto significa que el reenvío del agente no funciona:

```
$ echo "$SSH_AUTH_SOCK"
# Print out the SSH_AUTH_SOCK variable
> [No output]
$ ssh -T git@github.com
# Try to SSH to github
> Permission denied (publickey).
```

4.8 Solucionar problemas del reenvío del agente SSH

Aquí te mostramos algunos puntos en los cuales tener cuidado cuando intentes solucionar problemas relacionados con el reenvío del agente SSH.

4.9 Debes utilizar una URL con SSH para revisar el código

El reenvío SSH funciona únicamente con URL con SSH, no con aquellas de HTTP(s). Revisa el archivo. *git/config* en tu servidor y asegúrate de que la URL es de estilo SSH como se muestra a continuación:

```
[remote "origin"]
url = git@github.com:yourAccount/yourProject.git
fetch = +refs/heads/*:refs/remotes/origin/*
```

4.10 Tus llaves SSH deben funcionar localmente

Antes de que hagas que tus llaves funcionen a través del reenvío del agente, primero deben funcionar localmente. Nuestra guía para generar llaves SSH puede ayudarte a configurar tus llaves SSH localmente.

4.11 Tu sistema debe permitir el reenvío del agente SSH

Algunas veces, la configuración del sistema deja de permitir el reenvío del agente SSH. Puedes verificar si se está utilizando un archivo de configuración del sistema ingresando el siguiente comando en la terminal:

```
$ ssh -v example.com
# Connect to example.com with verbose debug output
> OpenSSH_5.6p1, OpenSSL 0.9.8r 8 Feb 2011
> debug1: Reading configuration data /Users/you/.ssh/config
> debug1: Applying options for example.com
> debug1: Reading configuration data /etc/ssh_config
> debug1: Applying options for *
$ exit
# Returns to your local command prompt
```

4.12 Tu servidor debe permitir el reenvío del agente SSH en las conexiones entrantes

El reenvío del agente también puede bloquearse en tu servidor. Puedes verificar que se permita este reenvío si entras al servidor mediante SSH y ejecutas `sshd_config`. La salida de este comando deberá indicar que se configuró `AllowAgentForwarding`.

4.13 Tu ssh-agent local debe estar ejecutándose

En la mayoría de las computadoras, el sistema operativo lanza el `ssh-agent` automáticamente. Sin embargo, en Windows, tienes que hacerlo manualmente. Tenemos una guía de cómo empezar con el `ssh-agent` cuando abres Git Bash.

Para verificar que el `ssh-agent` se está ejecutando en tu computadora, teclea el siguiente comando en la terminal:


```
$ echo "$SSH_AUTH_SOCKET"
```

```
# Print out the SSH_AUTH_SOCKET variable
```

```
> /tmp/launch-kNSlgU/Listeners
```

4.15 Tu llave debe estar disponible para el ssh-agent

Puedes verificar que tu llave esté visible para el ssh-agent si ejecutas el siguiente comando:

```
ssh-add -L
```

Si el comando dice que no hay identidad disponible, necesitarás agregar tu llave:

```
$ ssh-add yourkey
```

sintaxis

Parámetro	Detalles
--format = <fmt>	Formato del archivo resultante: <code>tar</code> o <code>zip</code> . Si no se dan estas opciones y se especifica el archivo de salida, el formato se infiere del nombre de archivo si es posible. De lo contrario, el valor predeterminado es <code>tar</code> .
-l, --list	Mostrar todos los formatos disponibles.
-v, --verbose	Reportar el progreso a stderr.
--prefijo = <prefijo> /	Prepone <prefix> / a cada nombre de archivo en el archivo.
-o <archivo>, -- output = <archivo>	Escribe el archivo en <archivo> en lugar de stdout.
--trabajo-atributos	Busque los atributos en los archivos <code>.gitattributes</code> en el árbol de trabajo.

<extra>	Esta puede ser cualquier opción que el backend del archivador entienda. Para <code>zip</code> backend <code>zip</code> , el uso de <code>-0</code> almacenará los archivos sin desinflarlos, mientras que de <code>-1</code> a <code>-9</code> se puede usar para ajustar la velocidad de compresión y la relación.
--remote = <repo>	Recupere un archivo tar desde un repositorio remoto <code><repo></code> lugar del repositorio local.
--exec = <git-upload-archive>	Se usa con <code>--remote</code> para especificar la ruta al <code><git-upload-archive></code> en el control remoto.
<tree-ish>	El árbol o el compromiso de producir un archivo para.
<ruta>	Sin un parámetro opcional, todos los archivos y directorios en el directorio de trabajo actual se incluyen en el archivo. Si se especifican una o más rutas, solo se incluyen estas.

CONCLUSIÓN

Un repositorio es un sistema en red que proporciona servicios web sobre una colección de objetos digitales, basado en una arquitectura abierta.

Finalmente, los repositorios son una herramienta indispensable, para los usuarios ya que:

- La creación de un repositorio requiere del esfuerzo conjunto entre grupos de diversas especialidades.
- Acceso abierto y aporta beneficios de visibilidad e impacto a investigaciones.
- Existe la necesidad de coordinar actividades entre entidades publicas y privadas para desarrollar mas y mejor los repositorios.
- Ofrece a la comunidad educativa un conjunto de servicios.

REFERENCIAS

- [1] Ferreras-Fernández T, M.-V. J. (2015). Repositorios de acceso abierto: Un nuevo modelo de comunicacion científica. La revista de la sociedad ORL CLCR en el repositorio Gredos, 94-113.
- [2] Rocio, S. V. (2017). Evaluacion de los repositorios institucionales de acceso abierto en España. Barcelona: Universitat Barcelona.
- [3] Gredos, <http://gredos.usal.es>
- [4] Berlin Declaration, <http://oa.mpg.de/lang/en-uk/berlin-prozess/berliner-erklarung/>
- [5] Open Access Policies and Mandates, <https://www.openaire.eu/policies-and-mandates/open-access-pilot/open-access-policies-and-mandates>
- [6] OpenDOAR, <http://www.opendoar.org/>
- [7] DAUZON, S. (Mayo 2018). *Git controle la gestion de sus versiones* . Barcelona: Ediciones ENI.
- [8] Diez años desde la Budapest Open Access Initiative: hacia lo abierto por defecto” <http://www.soros.org/openaccess/boai-10-translations/spanish>
- [9] <https://aulasoftwarelibre.github.io/taller-de-git/introduccion/>
- [10] <https://aulasoftwarelibre.github.io/taller-de-git/github/>
- [11] <https://www.uv.mx/ciies/documentos/biblioteca/>
- [12] <https://www.recolecta.fecyt.es/sites/default/files/contenido/documentos/mit.pdf>
- [13] Mary R. Barton, y. M. (2005). *Como crear un repositorio institucional*. The Cambridge-MIT: Institute (CMI).
- [14] https://upcommons.upc.edu/bitstream/handle/2117/76761/JENUI2015_76-83.pdf
- [15] Paul Sawers. GitHub Wants Schools to Collaborate Around code, Febrero 2014. <http://thenextweb.com/insider/2014/02/11/github-wants-schoolscollaborate-code/>
- [16] <https://aulasoftwarelibre.github.io/taller-de-git/cvs/>
- [17] <https://github.com/Hispano/Guia-sobre-Git-Github-y-Metodologia-de-Desarrollo-de-Software-usando-Git-y-Github>
- [18] <https://www.acens.com/comunicacion/white-papers/control-versiones-git-github/>
- [19] <https://aulasoftwarelibre.github.io/taller-de-git/usobasico/>

[20]<https://docs.github.com/es/enterprise-server@2.22/github/authenticating-to-github/error-key-already-in-use>

[21]<https://docs.github.com/es/enterprise-server@2.22/developers/overview/managing-deploy-keys#deploy-keys>

[22] <https://docs.github.com/es/enterprise-server@2.22/developers/overview/using-ssh-agent-forwarding>