INSTITUTO de ENSEÑANZAS a DISTANCIA de ANDALUCÍA

2º de Bachillerato

Tecnologías de la Información y Comunicación

Contenidos

Ciclo de vida: Trabajo en equipo y control de versiones

Imagen en Pixabay de TusitaStudio bajo licencia CC0 Puclic Domain

Trabajar en equipo consiste en reunir a dos o más personas organizándolas de tal manera que puedan cooperar para realizar en común un proyecto que sería muy difícil o imposible de conseguir de forma individual. También se origina cuando se pretende agilizar y mejorar algunas condiciones que obstaculizan el desarrollo de tareas diarias y la consecución de objetivos en las organizaciones.

Cuando se trabaja en equipo, se aúnan las aptitudes de los miembros y se potencian sus esfuerzos, disminuyendo el tiempo invertido en las tareas y aumentando la eficacia en la consecución de los resultados.

No todo grupo de personas reunidas forma un equipo de trabajo, ya que para ello **se necesita alcanzar un cierto grado de cohesión y compenetración entre los componentes**. Para ello se han de crear lazos de atracción interpersonal, fijar una serie de normas que dirijan el comportamiento de todos los miembros, donde la figura de un líder es fundamental, promover una buena comunicación entre el conjunto de integrantes, trabajar por el logro de los objetivos comunes y establecer relaciones positivas.

La cohesión de un equipo de trabajo se expresa a través del compañerismo y el sentido de pertenencia al grupo que manifiestan sus componentes. Mientras más cohesión exista, mejor trabajarán sus miembros y más productivos serán los resultados de sus acciones.

Cuando se detectan dificultades y problemas en los equipos de trabajo, se recurre a otro equipo que será el encargado de revisar y analizar el trabajo que se está realizando, es el llamado **equipo de mejora continua,**formado por un grupo de personas que pueden pertenecer al mismo departamento o sección que los del equipo de trabajo o a otro departamento diferente. La misión de este equipo es la de mejorar el nivel de calidad y productividad del equipo de trabajo. Una vez solucionados los problemas, el equipo de mejora continua puede disolverse.

Para ayudar al equipo de trabajo a sincronizar sus tareas, sobre todo cuando más de un miembro opera sobre una misma parte del proyecto, se necesita realizar un **control de versiones** del trabajo realizado. Para ello se recurre a los **sistemas de control de versiones**. En este tema conocerás su uso y podrás apreciar la potencia y la necesidad de los mismos.

1. ¿Qué lograrás?

Una vez que hayas terminado de estudiar este tema tendrás claras varias cuestiones relacionadas con el mundo de la programación:

- La importancia del trabajo en equipo, y todo lo necesario para el buen desempeño del mismo.
- La conveniencia de mejora continua, sobre todo de cara a superar las dificultades surgidas durante el trabajo en equipo.
- La necesidad de los sistemas de control de versiones y todo lo relacionado con el funcionamiento de los mismos.

Imagen en Flickr de Nguyen Hung Vu con Algunos derechos reservados

Por un lado, apreciarás las ventajas del trabajo en equipo, sobre todo para tareas de amplio volumen o complejas. Verás los aspectos necesarios e imprescindibles que deben cumplirse para que un equipo de trabajo funcione adecuadamente, haciendo que cada miembro del mismo desempeñe su labor correctamente y sincronizando a la perfección sus tareas con las de los demás para conseguir un resultado global final adecuado.

Te darás cuenta que al trabajar en equipo obtendremos muchas ventajas, traducidas sobre todo en reducción de costes (de tiempo, de producción, etc) pero también surgirán algunas desventajas. En cualquier caso, verás que las desventajas serán más livianas que las dificultades que tendríamos al realizar las mismas tareas pero de forma individual en lugar de grupal, y por supuesto, conocerás las maneras de solventar dichas desventajas.

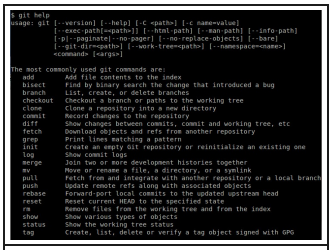


Imagen en Flickr de Linux Screeshots con Algunos derechos reservados

Comprobarás que una de las dificultades que nos encontramos durante el desarrollo de software es la de controlar las distintas versiones que vamos obteniendo de nuestros programas. Efectivamente, reconocerás que tendremos la imperiosa necesidad de llevar un buen control de los distintos cambios que realizamos sobre nuestro código fuente. Para ello se requiere de un sistema de control de versiones idóneo. Aprenderás el vocabulario necesario para poder manejar un sistema de control de versiones, con toda la carga de operaciones básicas requeridas para ello. Palabras como repositorio, repositorio local, repositorio remoto, commit, branch, etc... dejarán de ser una incógnita para tí.

Posteriormente pondrás en práctica todos esos conceptos utilizando un Sistema de Control de Versiones distribuido llamado Git. Aprenderás la utilidad de los comandos más frecuentes del mismo para manipular los proyectos a controlar. Para terminar conocerás GitHub, un hosting para el sistema de control de versiones Git, gratuito para proyectos opensource y que aporta toda la potencia de Git con una interfaz gráfica.

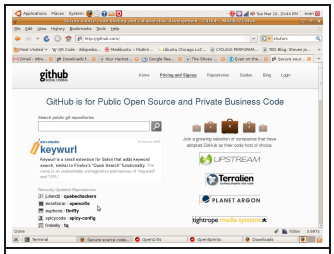


Imagen en Flickr de evan con Algunos derechos reservados

Al principio del tema no lo tendrás demasiado claro, pero al final del mismo te darás cuenta de la potencia que ofrecen los sistemas de control de versiones.

2. Trabajo en equipo

Cuando se desarrolla software, a medida que aumenta la complejidad del mismo, aumenta considerablemente el número de líneas de código necesarias y con ello las horas de desarrollo a emplear. Parece evidente que si un programador necesita un tiempo determinado para realizar un programa, dos tardarían la mitad del tiempo, sin embargo, cuando se trabaja en equipo, los esfuerzos de los miembros se potencian, disminuyendo el tiempo de acción y aumentando la eficacia de los resultados, haciendo que el tiempo de desarrollo se reduzca mucho más.

Para proyectos de embergadura se hace pues indispensable que varios programadores trabajen sobre el mismo código. Pongamos por caso el ejemplo de la elaboración del un programa completo de gestión empresarial que abarcara desde la gestión de stock en almacenes, entradas, salidas, conformación de pedidos y paquetería, etc., hasta el control de facturación de ventas y compras, albaranes, control de flujos de tesorería, vencimientos de cobros y pagos, envíos de remesas digitales a bancos para cobros de recibos, domiciliación de pagos, etc. También podríamos implementar una parte para gestión contable, entrada de asientos, libro diario, libro mayor de cuentas, cuentas anuales, memoria, balances de comprobación, balances de situación, cuentas de explotación, cuenta de Pérdidas y Ganancias. Como vemos el trabajo es arduo y extenso. Qué ocurriría si no coordinamos los trabajos y lo programado por un desarrollador es machacado por lo elaborado por otro. Qué pasaría si en una modificación o revisión uno de los programadores no se percatara de que su última adición al programa no se ha mantenido porque el punto de partida tomado por otro desarrollador ha sido una versión anterior.

Si el trabajo en equipo no funciona bien, el proyecto puede llegar a ser un desastre. Para ello es de vital importancia incidir en el propio equipo. Existe un esquema llamado de las "5c" que define los roles del trabajo en equipo, y son los siguientes:

- **Complementariedad**: Conseguir que cada miembro del equipo domine una parcela determinada del proyecto, de tal forma que uniendo los conocimientos de todos se pueda sacar el trabajo adelante.
- **Coordinación**: Todos los miembros, guiados por un líder, deberán actuar de forma organizada para conseguir los objetivos marcados en el proyecto.
- **Comunicación**: Para alcanzar un grado de coordinación óptimo deberá existir una comunicación abierta entre todos los componentes del equipo, logrando que cada miembro sepa lo que tiene que hacer y cuándo lo tiene que hacer.
- **Confianza**: Cada componente del equipo deberá confiar en el buen hacer del resto de miembros, anteponiendo el éxito del equipo al suyo propio y pensando que el resto de sus compañeras y compañeros harán lo mismo.
- **Compromiso**: Cada miembro se comprometerá a aportar lo mejor de si mismo y a poner todo su empeño en sacar el trabajo adelante.

Video en YouTube de GoEmprendedor bajo licencia de YouTube estándar

En definitiva, si el equipo no está bien dirigido por un líder, cumpliendo con todos los requisitos anteriormente mencionados, el trabajo realizado puede volverse contradictorio, pues tener que volver sobre pasos anteriores será como mínimo una pérdida de tiempo, o aún peor, puede llevar al equipo a realizar una serie secuencial de parches para resolver problemas que lleguen a distorsionar en gran medida la estructura misma del programa.

Imagen en Flickr de Antonio Marín Segovia con Algunos derechos reservados

¡Como diría el gran emperador romano Julio César, "divide para vencer"! Tanta razón tenía que otro gran estratega, cientos de años más tarde, lo aplicó en sus batallas: Napoleón Bonaparte. Estos dos grandes personajes de la historia hicieron suya esta frase y la llevaron a la práctica en multitud de ocasiones con gran éxito. La idea se basa en **"romper" un problema complejo a resolver en problemas menores, para así poder abordar cada uno de estos últimos de forma separada y con menor esfuerzo**.

En informática, este es un principio muy conocido y utilizado, buena prueba de ello, **por ejemplo**, es la **programación modular**, que sugiere y permite el uso de subprogramas (procedimientos y funciones) para realizar aplicaciones. Esta particularidad, característica de la programación modular, **propicia que para el desarrollo del software se puedan emplear varios programadores**, o incluso varios cientos de ellos, de tal manera que cada uno pueda implementar una parte del proyecto. Cuanto mayor sea el proyecto a desarrollar, existirá más complejidad y, por tanto, mayor necesidad de programadores, haciéndose indispensable un buen trabajo de equipo en el que la regla de las "5c" se hará imprescindible y de vital importancia. Efectivamente, la complementariedad, la coordinación, la comunicación, la confianza y el compromiso de todos los programadores serán vitales para alcanzar un buen puerto. Aunque no bastará con todo ello, ya que la tarea se complicará cuando ocurra lo más frecuente, que no es otra cosa que la necesidad que tendrán todos los programadores de trabajar sobre el mismo código fuente incluso simultáneamente...

Imagen en Flickr de gadgetdude con Algunos derechos reservados

La técnica de "divide y vencerás" por tanto simplifica mucho los problemas a resolver, aunque puede generar otro tipo de "efectos secundarios" o problemas secundarios, aunque de menor envergadura. En cualquier caso, para ellos también se han pensado e ideado una serie de soluciones, como verás en los sucesivos apartados del tema.

Por tanto, cada vez que lo necesites, no lo dudes...**¡¡¡DIVIDE Y VENCERÁS!!!**

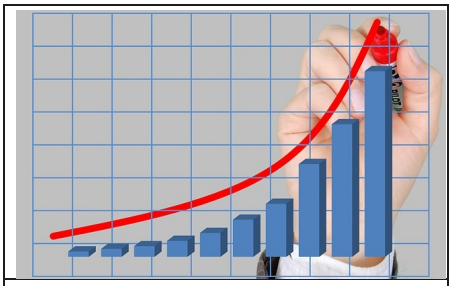


Imagen en Pîxabay de Tumisu bajo licencia CC0 Public Domain

Imagen en Pîxabay de 3dman_eu bajo licencia CC0 Public Domain

Cuando se tiene la necesidad imperiosa de que todos los miembros del equipo trabajen sobre el mismo código fuente, la coordinación es vital, ya que un pequeño descuido en la manipulación del mismo podría acarrear la pérdida de modificaciones o ampliaciones ya realizadas, haciendo que los tiempos de implementación se incrementen notablemente. **Se hace indispensable por tanto una mejora en la gestión del código fuente y en la forma de trabajar de todos los programadores**.

Cualquier persona que haya trabajado con un ordenador se habrá visto en la **necesidad de manejar distintas fases por las que un trabajo va pasando** (la creación de un informe, de una presentación, de un retoque fotográfico, de una hoja de cálculos, etc). Lo habitual es ir guardando distintas copias de un mismo fichero en las que se va cambiando el contenido de unas a otras, **de tal manera que podamos volver a un estado anterior si nos arrepentimos de los últimos cambios realizados o, en caso de pérdida del trabajo, poder recurrir a alguna de las últimas copias que tuviésemos del fichero, para así recuperar la mayor cantidad posible**. Seguramente todos hayamos utilizado diferentes métodos para llevar a cabo el control de los distintos ficheros que se van generando en la realización de un trabajo con el ordenador. En cualquier caso, la mayoría de las veces solemos recurrir a técnicas parecidas, comenzando por lo típico: realizar copias de ficheros con nombres como "Copia del trabajo", "Copia de Copia del trabajo", "Copia de Copia de Copia del trabajo", etc. Lo siguiente suele ser el juego con las fechas, añadiendo la misma en el propio nombre de los ficheros para reflejar cómo de actualizado está el contenido de los mismos. También solemos recurrir a almacenar lo distintos ficheros en carpetas separadas y clasificadas por fechas o haciendo referencia a algún cambio significativo que se haya producido en los contenidos. Añadimos a veces incluso la hora junto a la fecha para asegurarnos todavía más.

Como ves, en el primer párrafo se habla de coordinación, de colaboración, reducción de los tiempos, etc. En el segundo se hace alusión a la seguridad, a la necesidad de asegurar que no se pierda lo que ya hemos construido, a sistemas propios digamos más o menos artesanos de no tener que reelaborar un programa por algun fallo. También hemos hablado de optimizar rendimientos, alcanzar objetivos, trabajo en equipo, cohesión, compañerismo, etc. Todas estas metas o hitos que hay que ir consiguiendo son lógica consecuencia del primer vocablo de este apartado, la **mejora necesaria**, que nos lleva directamente a la evolución en la forma de trabajar, por no hablar de la **creciente competitivad en el terreno personal y la feroz competencia del mercado a la hora de asignar trabajos** y mucho más proyectos de gran peso. En este último sentido el segundo vocablo es de gran elocuencia.

Llegamos pues a una clara conclusión, en cuanto al trabajo, tanto personal como en equipo... **¡¡¡MEJORA O MORIRÁS!!!**

Imagen en Pixabay de qimono con licencia CC0 Public Domain

Parece ser, por tanto, que **el éxito o el fracaso en la realización de un proyecto no solo depende del equipo de trabajo** y de las aptitudes y actitudes que demuestren sus miembros, sino **también de las técnicas y herramientas empleadas para controlar y salvaguardar los cambios de forma adecuada**.

Hemos podido observar que el control de dichos cambios no es fácil y además que puede ser catastrófico si no se ejecuta de una forma ordenada y precisa. Aquí es donde entra en juego la idea de ejercer, de forma adecuada, un control de versiones en nuestro trabajo diario y con ello, de paso, utilizar algunas herramientas esenciales para aliviar toda esta problemática: Los **Sistemas de Control de Versiones**.

En los siguientes apartados del tema **estudiarás en mayor profundidad todo lo relacionado con el control de versiones**, los conceptos fundamentales y más generales del mismo, los tipos de sistemas existentes para realizar dicho control, así como sus operaciones básicas. **Y para finalizar podrás observar, en la práctica, cómo funciona un sistema de control de versiones** concreto, de uso libre y de gran impacto en las empresas hoy en día: **GIT**.

Imagen en Pixabay de geralt bajo licencia CC0 Public Domain

3. Control de versiones

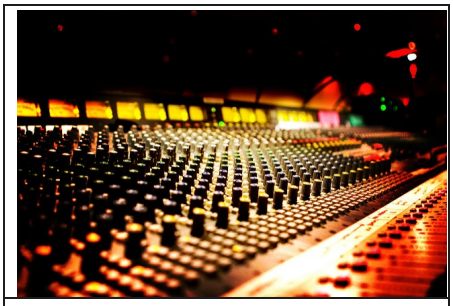


Imagen en Flickr de Photographer con Algunos derechos reservados

Cuando se elabora un programa o cualquier producto de software en general, aunque se haya planificado bien con antelación, se haya diseñado convenientemente o se hayan elaborado borradores y diagramas de flujo, necesariamente en la evolución del producto se producen modificaciones, ediciones, revisiones, actualizaciones, cambios en su configuración, etc. Por otro lado cuando se elabora software industrial para proyectos grandes lo normal es que sean varios los desarrolladores de programación que intervengan en la ejecución de un programa. Esta colaboración necesaria entre ellos hace que sea indispensable contar con herramientas adecuadas para coordinar los trabajos pues el control manual no es suficiente.

Para gestionar todos estos cambios de forma adecuada existe lo que se denomina **Sistema de Control de Versiones** (VCS del inglés *Version Control System*). Es necesario disponer de un mecanismo de almacenamiento de elementos o archivos, fácil acceso a esos elementos para modificarlos, saber qué elementos se modificaron y cuáles no, registrar los usuarios que han realizado dichos cambios, en qué momento se realizaron, etc, es decir poseer un registro de esos cambios. En definitiva es necesario tener un historial que registre toda esta información, almacenado en algún lugar llamado **Repositorio**. También llamado depósito, suele estar en el disco duro de un servidor. Un VCS dispone de todo eso y además incluye funcionalidad para revertir la colección de archivos a otro estado. Otro estado puede significar a otra colección diferente de archivos o contenido diferente de los archivos. Por ejemplo podemos cambiar la colección de archivo por la que teníamos la semana anterior.

Es conveniente también tener la posibilidad, en un momento determinado, de abrir varias vías de trabajo de un mismo proyecto para luego decidir con cuál quedarse, o incluso poder mezclar las ideas y los cambios aplicados en todas o en varias de esas vías distintas. Es precisamente otra característica importante que aportan de estos sistemas.

Ya hemos dicho que un repositorio es básicamente un lugar de almacenamiento, un depósito de archivos que pueden ser de todo tipo (de imágenes, videos, animaciones, textos, etc). **En el campo que nos atañe, el del desarrollo del software, en un repositorio se almacenan proyectos y programas, versiones de éstos y sus modificaciones en el tiempo**, de forma que se tiene un histórico de esos programas o proyectos, pudiendo revertirlos a un estado anterior en el tiempo sin problemas. Normalmente suelen estar en el disco duro de un servidor que a través de internet está disponible.

Imaginemos que queremos hacer un programa. Empezamos grabando el primer archivo con un nombre, por ejemplo "proyecto". A medida que vamos modificando y aumentando sus líneas de código vamos grabando sobre el propio archivo, es decir "machacando" ese archivo con las últimas grabaciones, de tal forma que si queremos ver algo que hace varios días teníamos ya no es posible debido al número elevado de variaciones que hemos hecho. Llega un momento que decidimos grabar el archivo con otro nombre para conservar el actual en este preciso estado que tenemos y grabar lo próximo con otro nombre, por ejemplo "proyecto avanzado". En el segundo tenemos repetido todo el código del primero pero si en este segundo modificamos algo del contenido del primero ya no podríamos ver su estado anterior si nos faltara el primer archivo. Si repetimos varias veces el proceso tendremos por ejemplo los siguientes archivos: "proyecto", "proyecto avanzado", "proyecto avanzado mejorado", "proyecto V1", "proyecto V2", etc. Es una forma de tener un histórico del programa con las modificaciones y fechas de las mismas. Para hacer esto mismo pero de forma avanzada existen los programas de Sistemas de Control de Versiones (VCS). Este software hace todo eso de forma automática almacenando toda la información en los repositorios.

Un ejemplo de este tipo de sistemas es **Git**, que podrás estudiar en este mismo apartado.

Curiosidad

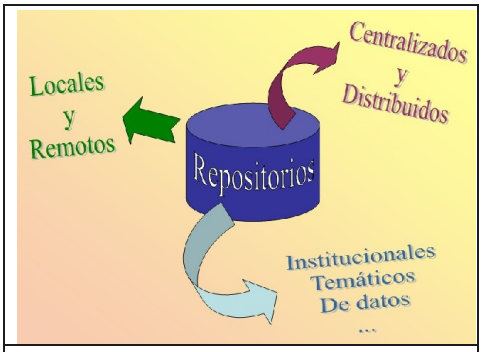
La mayoría de las veces, no nos damos cuenta que necesitamos un Sistema de Control de Versiones hasta que lo estamos utilizando. Es entonces cuando nos paramos a pensar cómo habíamos podido desarrollar las tareas hasta ese momento sin utilizarlo. Cuando termines el tema, tú también podrás valorar esta idea, y estarás en condiciones de plantearte el uso o no de un Sistema de Control de Versiones para tus trabajos y los de tus equipos.

Imagen en Pixabay de Open-Clipart Vectors bajo licencia CC0 Public Domain

Existen repositorios de varios tipos. Por un lado tenemos **repositorios locales**, es decir carpetas o directorios en el disco duro de nuestro propio ordenador, y por otro lado tenemos **repositorios remotos** en servidores, la web, en internet. Los repositorios remotos son versiones de tu proyecto grabados en algún sitio de Internet. Pueden ser de sólo lectura, o de lectura/escritura, según los permisos que tengas. Colaborar con otros implica gestionar estos repositorios remotos, y mandar (push) y recibir (pull) datos de ellos cuando necesites compartir cosas.

Hay programas VCS, como Git, que son **distribuidos** y otros que son **centralizados**. En los distribuidos cada usuario trabaja independientemente sin tener que acceder al repositorio central remoto, elaborando el proyecto dentro de su copia de trabajo, grabando una copia completa de ésta en su propio repositorio local (todo en su disco duro, un directorio que contiene dos carpetas, una con la copia de trabajo y otra con el repositorio). Existe a su vez una copia completa de ese historial (del repositorio) en su repositorio remoto. Si otro desarrollador va a empezar a trabajar sobre el proyecto sólo tiene que copiar (bajar) el contenido del repositorio remoto (es lo que se llama clonar el repositorio) a su ordenador para poder comenzar. Por tanto el servidor remoto se usa para publicar el proyecto y que otros usuarios puedan acceder, pero siempre se trabaja en el repositorio local.

Todo está sincronizado, de forma que si cae el sistema de algún usuario o de varios, o incluso del servidor central, con cualquier copia de otro usuario tendríamos el programa totalmente actualizado, ya que la información está muy replicada. En el repositorio nos aparecerían los archivos grabados relacionando fechas y horas, con lo que accediendo a cada uno de ellos tendríamos el estado del programa en ese momento determinado del tiempo, facilitando al programador información sobre qué modificó en cada instante y por qué.



Fuente propia

En los sistemas centralizados existe un repositorio central de todo el código. Permiten la colaboración entre varios puestos de trabajo, manteniendo el repositorio centralizado en un único ordenador, que estará accesible para el resto de puestos a través de red local o Internet. Suele haber un usuario responsable del sistema que decide las cuestiones importantes, como por ejemplo qué modificaciones recibidas de repositorios locales incluir en el repositorio central y cuáles no. Algunos ejemplos de este software son, Subversion o Team Foundation Server.

Otra gran ventaja de sistemas VCS es que son básicos para trabajos colaborativos en el que múltiples programadores pueden trabajar simultáneamente sobre el mismo proyecto, en incluso en muchas ocasiones sobre los mismos archivos. El sistema se encarga de controlar que no haya problemas a la hora de sincronizar los trabajos. Así se garantiza, sea cual sea el momento en el que un programador acceda al proyecto para trabajar, que la copia de trabajo con la que arranca es la actual, la que contiene todo lo que los diferentes desarrolladores han incluido hasta ese instante.

También podemos clasificar los repositorios en **Institucionales, temáticos o de datos**. Los primeros son creados por organizaciones para depositar información, usarla facilitando acceso libre por considerarla útil y necesaria para toda la sociedad, y preservar conocimientos de tipo cultural, científico o académico. Tienen repositorios de este tipo por ejemplo las universidades de Sevilla o de Canarias. Los repositorios temáticos son creados por personas, investigadores o científicos sobre temas concretos o específicos y almacenan documentos relacionados con los mismos, sin importar en sí mismo los creadores sino exclusivamente la temática. Los repositorios de datos son dedicados a almacenar los datos obtenidos en los temáticos.

Para comprender mejor la esencia fundamental del control de versiones y de los sistemas de control de versiones es conveniente primero conocer algunos de los términos que se suelen utilizar con más frecuencia. Así pues, ahora vamos a hacer un pequeño desglose de los más importantes, de esta manera, podrás entender con mayor claridad las ideas principales de esta poderosa herramienta. Son muchos conceptos nuevos, pero no te asustes, poco a poco los irás comprendiéndolos e irás familiarizándote con ellos de una forma muy sencilla.

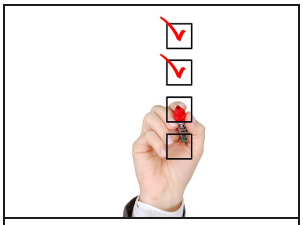


Imagen en Pixabay de Tumisu bajo licencia CC0 Public Domain

El repositorio de un sistema de control de versiones se organiza en directorios y ficheros, de tal forma que todos los que se refieren a un mismo proyecto constituyen lo que se denomina **Módulo**.

Hay distintas formas de nombrar las revisiones, es decir, de nombrar las nuevas versiones de un producto original, bien mendiante contadores, códigos o rótulos (**tags**). **Rotular** el módulo es nombrar todos los archivos de un proyecto en un momento determinado para reencontrar ese mismo estado de desarrollo en un momento posterior. Para ello es necesario congelar el módulo durante el rotulado. **Congelar** significa permitir los últimos cambios para solucionar las fallas a resolver en una entrega (***release***) y suspender cualquier otro cambio antes de una entrega, con el fin de obtener una versión consistente. Si no se congela el repositorio, un desarrollador podría comenzar a resolver una falla cuya resolución no está prevista y cuya solución dé lugar a efectos colaterales imprevistos.

El concepto de "**Línea Base**" (o baseline) es muy intuitivo. A partir de un fichero fuente aprobamos una primera revisión que nos servirá de base para futuras revisiones o modificaciones.

De un módulo se pueden tener en un momento dado varias copias (dos o más ramas, **branch**) que evolucionan de forma independiente siguiendo su propia línea de desarrollo. Es una gran ventaja pues después se puede operar con ellas de diversas formas, por ejemplo fusionándolas (**merge**). En general cada persona trabaja en el proyecto creando su propia rama y trabaja en una funcionalidad diferente. Es decir, se diversifican del proyecto principal nuevas ramas con sus propias versiones completas. Cuando cada uno termina de trabajar en su rama, se fusionan la rama actual con la rama "**master**". La rama "master" es el directorio principal del proyecto.

Imagen en Flickr de Eduard Yanquen con Algunos derechos reservados

Un **Despliegue** (**check-out**) crea una **copia de trabajo local** de los ficheros de un repositorio, en un momento del tiempo o revisión específicos (se puede elegir cualquier revisión pero por defecto se suele tomar de la última). Todo el trabajo realizado sobre los ficheros en un repositorio se realiza inicialmente sobre una copia de trabajo, de ahí su nombre. Conceptualmente, es un **cajón de arena** o ***sandbox***.

Se crea un **Commit** (**check-in**) cuando una copia de los cambios hechos a una copia de trabajo local es escrita o integrada en el repositorio. Se está creando en ese punto en el tiempo un punto de control al que se puede volver en caso de necesitar restaurar el proyecto.

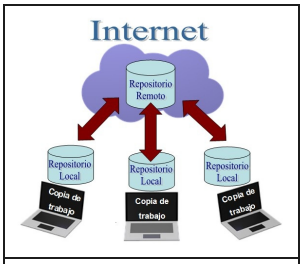
También pueden surgir **Conflictos** cuando el sistema no puede gestionar adecuadamente cambios realizados por dos o más usuarios en un mismo archivo. A veces un usuario tiene que intervenir para **Resolver** los conflictos originados cuando se realizan cambios en un mismo archivo. Para que una modificación se considere **Cambio** se tiene que producir bajo un sistema de control de versiones. Una **Lista de Cambios** puede representar una vista secuencial del código fuente.

Una **Exportación** es similar a un check-out, es decir es parecido a una copia de trabajo local desde el repositorio, pero en este caso se crea un árbol de directorios limpio sin los metadatos de control de versiones presentes en la copia de trabajo.

Una **Importación** es la acción de copia de un árbol de directorios local (que no es en ese momento una copia de trabajo) en el repositorio por primera vez.

Una **Fusión** une dos conjuntos de cambios sobre un fichero o un conjunto de ficheros en una revisión unificada de dicho fichero o ficheros.

Una **Actualización** integra los cambios que han sido hechos en el repositorio (por ejemplo por otras personas) en la copia de trabajo local.



Fuente propia

El objetivo básico perseguido pon un Sistema de Control de Versiones (VCS) es conseguir que varios desarrolladores trabajen simultáneamente, sin confllictos y vayan guardando cada versión obtenida, registrando cuándo se grabó, quién lo ejecutó, qué cambios se hicieron respecto a versiones anteriores, por qué se realizaron etc., es decir mantener mediante una gestión de archivos y directorios un histórico del proyecto o programa elaborado que permita incluso recuperar versiones anteriores del mismo.

Para conseguir este objetivo este software dispone de funcionalidades específicas en cada uno de los diferentes programas (Git, Subversión, Team Foundation Server, etc.) que desarrollan a través de distintas órdenes o comandos. En este punto no entraremos en diferentes programas pero si estableceremos con carácter general una serie de operaciones comunes o básicas que deben ejecutar todos ellos.

Todos ellos usan un mecanismo de almacenamiento de los datos y la información asociada (metadatos). Esta base de datos o almacén se llama repositorio. Las operaciones comunes que hemos comentado se producen en torno a él, introduciendo o sacando infomación del mismo.

Cada usuario tiene una copia de ese repositorio (repositorio local) que digamos sirve de puente entre el repositorio central (repositorio remoto) y la copia local de trabajo. El flujo de información circula desde la copia local de trabajo hasta el repositorio central (remoto) pasando por el repositorio de cada usuario (local) o bien en sentido inverso, es decir desde el repositorio remoto central al repositorio local y desde éste a la copia local de trabajo.

Cuando un desarrollador lo considera oportuno graba uno o más cambios (lista de cambios o changeset) en su repositorio estableciendo lo que se llama revisión **(Check-in o Commit)**. Usando una *terminología particular y a modo aclaratorio*, es lo que podriamos llamar puntos de versión, o de restauración del proyecto, de cara a una necesidad futura de acudir a una versión anterior. Las revisiones se identifican con números o etiquetas (tags). A su vez cuando un usuario quiere empezar a trabajar manteniendo las modificaciones hechas por otro, clona el repositorio central a su propio repositorio local y también hace una copia de éste en su copia de trabajo local **(Check-out)**. Estas operaciones en ambos sentidos se llaman integraciones. Si queremos integrar en nuestra copia de trabajo actual los cambios que han efectuado otros usuarios en el repositorio debemos usar la opción **update** (o **Sync**).

Imagen en Pixabay de geralt bajo licencia CC0 Public Domain

De esta forma es posible que dos o más usuarios desarrollen ramas de trabajo distintas e independientes partiendo de un mismo punto. Cuando existen conflictos por haber trabajado simultaneamente varios desarrolladores sobre un mismo fichero el propio usuario tiene que solucionarlos, aunque en algunos casos el propio sistema puede manejarlos.

Imagen en Flickr de rofiwbwy con Algunos derechos reservados

Ahora es el momento de poner en práctica toda esta maraña de conceptos y vocablos extraños. Vamos a ver los fundamentos prácticos del control de versiones utilizando un Sistema de Control de Versiones distribuido, de uso libre y que ya hemos mencionado en más de una ocasión: Git. Podemos descargar la versión adecuada para nuestro sistema operativo en la web de Git.

La instalación de Git es sencilla, no tiene complicación alguna, no obstante en internet puedes encontrar algunas guías de instalación, como por ejemplo esta de aquí.

Git está basado en el uso de comandos para la realización de las operaciones básicas del control de versiones, así pues, en este apartado verás una lista de ellos y sus funciones.

Una vez instalado GIT ya podremos usarlo, para ello iremos a la línea de comandos y comenzaremos a teclear comandos de git. A continuación verás los comandos más esenciales para utilizarlo.

Configurando...

Los primeros comandos a utilizar serán los de configuración de nuestros datos personales. Es muy importante configurarlo con nuestros datos (nombre y email) ya que serán almacenados al hacer operaciones sobre los repositorios y por tanto vitales para identificar adecuadamente a los autores de los cambios que se realicen en un proyecto. Con los siguientes comandos lo podremos hacer:

git config --global user.name "Nombre del usuario"

git config --global user.email "<Correo electrónico del usuario>"

Creando un repositorio...

Una vez configurado, ya podremos crear un repositorio. Hay dos formas de crear un repositorio: o bien creando uno vacío, o bien clonando uno existente. Para crear uno vacío, nos situaremos en la carpeta donde queramos crear el repositorio y allí teclearemos el siguiente comando:

git init

Git creará para el repositorio una carpeta llamada ".git" que según el sistema operativo que estemos usando estará oculta o no.

La otra forma de crear un repositorio es clonar otro existente, ya sea de otra carpeta de nuestro ordenador o de algún sitio remoto:

git clone ruta_y_nombre_del_repositorio_a_clonar

Por ejemplo, si tenemos un repositorio llamado MiRepositorio en la carpeta /home/usuario teclearíamos: ***git clone /home/usuario/MiRepositorio***

Si el repositorio a clonar es remoto pondríamos su url, por ejemplo: ***git clone http://github.com/torvalds/linux***

Obteniendo ayuda...

Ya has visto los comandos más usuales de Git, aunque hay más. Para ver la lista completa de comandos de Git puedes solicitar en la línea de comandos ayuda tecleando alguno de los siguientes comandos:

git help

git help -a

git help --a

También puedes pedirle a Git ayuda sobre un comando en particular de la siguiente forma:

git help nombre_comando

Este comando abrirá el navegador web con la página oficial de Git que contiene ayuda sobre el comando en particular.

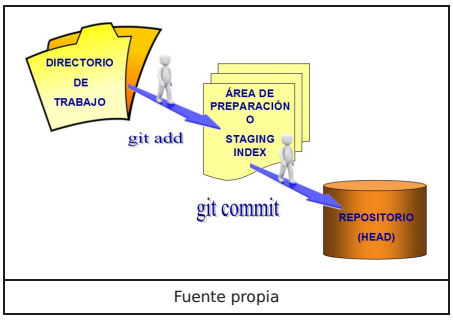
Puedes incluso solicitar ayuda del propio Git tecleando el comando siguiente:

git help git

Trabajando con GIT...

Cuando se trabaja con GIT tendremos 3 zonas de almacenamiento:

1. **El directorio de trabajo**: Será el directorio donde hayamos creado el repositorio, y en el tendremos los archivos que se quieran añadir al proyecto en algún momento determinado. Estos archivos aún no serán tenidos en cuenta por GIT para controlar sus cambios.
2. **Staging index**: O **área de preparación**, es un área temporal de GIT donde se guardarán los archivos que están a punto de ser enviados al repositorio.
3. **El repositorio**: También llamado **HEAD**, será la zona donde los archivos del proyecto estarán almacenados, ordenados y controlados para ser recuperados en cualquier momento.



Fuente propia

El modo de trabajo será el siguiente: Crearemos y modificaremos los archivos que tenemos en el directorio de trabajo, posteriormente pasaremos al staging index aquellos archivos del directorio de trabajo que queramos controlar con GIT, y después confirmaremos los cambios pasando los archivos del staging index al repositorio de GIT, para así ser controlados definitivamente.

Imaginemos que tenemos varios archivos en el directorio de trabajo para nuestro proyecto: principal.php, funciones.php, librerías.php,... Si queremos añadir alguno de ellos, o varios, al staging index podríamos usar los siguientes comando:

git add principal.php (para añadir el fichero principal.php)

git add *.php (para añadir todos los que tienen extensión php de golpe)

Cuando queramos confirmar los cambios y enviarlos al repositorio desde el staging index ejecutaremos el comando commit con la siguiente sintaxis:

git commit -m "mensaje descriptivo de los cambios realizados"

Con el comando commit añadiremos un mensaje que será almacenado en el repositorio con los cambios que se están registrando, de esa forma, en la base de datos del repositorio aparecerán dichos cambios asociados a este mensaje y sabremos qué cambios se hicieron en el proyecto. Es decir, este mensaje es una breve descripción de los cambios realizados en este paso y deberá tener menos de 50 caracteres, aunque si se teclea solo git commit aparecerá un editor de textos para escribir varias líneas (la primera deberá ser de menos de 50 caracteres y las demás de menos de 72).

Eliminando archivos...

Cuando se elimina un archivo del directorio de trabajo tendremos que notificárselo a GIT, ya que el archivo seguirá estando presente en el índice del repositorio (en el staging index). Lo haremos con el comando siguiente:

git rm nombre_archivo_eliminado

Posteriormente se podría hacer un commit para hacer definitiva la eliminación y registrarla en el repositorio.

Moviendo archivos...

Cuando movemos un archivo del directorio de trabajo a algún subdirectorio de éste, GIT se lo tomará como dos operaciones distintas: por un lado la de crear un archivo nuevo en la nueva ubicación y por otro lado la eliminación del archivo del directorio de trabajo. En teoría, por tanto, deberíamos ejecutar dos comandos git, uno para eliminar el archivo del directorio de trabajo (git rm) y otro para añadir el nuevo archivo al staging index (***git add***). Sin embargo, GIT es inteligente y solo ejecutando el segundo de estos comandos realizará también el primero, ya que comprobará que se trata del mismo archivo.

Examinando el registro...

En cualquier momento podremos ver una lista de los commits realizados en el repositorio utilizando alguno de los siguientes comandos:

git log (mostrará la relación de commits en formato largo, usando varias líneas por commit)

git log –oneline (mostrará la relación de commits en formato abreviado, usando una línea por commit)

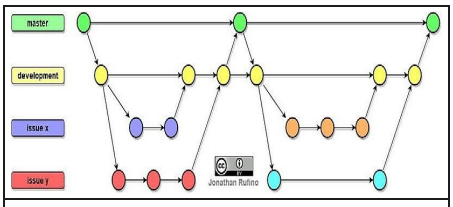
Creando ramas...

El flujo de trabajo de GIT se basa en el uso de ramas. Una rama (branch) es una línea de trabajo alternativa y simultánea a la línea de trabajo principal, en la que podremos realizar cambios en el proyecto que luego podríamos fijar, fusionar con otras ramas o simplemente descartar. **A la rama principal del proyecto se le llama master**.
El comando que nos permite crear ramas es el siguiente:

git branch nombre_rama

Este comando creará una rama de trabajo paralela a la rama en la que estemos situados en el momento de teclear el comando. Podemos movernos de una rama a otra con el siguiente comando:

git checkout nombre_rama



Veremos todas las ramas creadas con el comando:

git branch

Con este comando saldrá una lista de todas las ramas existentes en el proyecto, marcando con un asterisco la rama en la que estamos situados en ese momento.

Si necesitamos crear una rama y luego movernos a ella, en lugar de dos comandos podemos ejecutar uno solo de la siguiente manera:

git checkout -b nombre_rama

Fusionando ramas...

Podemos fusionar dos ramas, con lo que Git mezclará en una sola rama el contenido de las dos. La manera de fusionar dos ramas es situarse en la rama a la que quiero añadir el contenido de otra rama y ejecutar el comando siguiente:

git merge nombre_rama_a_fusionar

De esta forma, el contenido de la rama_a_fusionar se unirá al contenido de la rama en la que estamos situados. Esta fusión puede salir bien o no, dependiendo de si en ambas ramas se han modificado distintas partes del proyecto o las mismas. Puede ocurrir que en ambas ramas se haya modificado las mismas zonas del proyecto o no. Git mezclará la información de ambas ramas si no hay conflictos, y en caso de haberlos, añadirá en los lugares donde los hay unas marcas e introducirá ahí las distintas alternativas diferentes que encuentra , de tal manera que luego el usuario será el que tenga que editar esa zona y quedarse con la alternativa adecuada desechando el resto (y quitando las marcas que Git añadió para marcar el conflicto). Una vez corregidos los conflictos, si los hubiera, el usuario podría eliminar la rama que se fusionó con la actual (o no si desea conservarla).

Eliminando ramas...

Para eliminar una rama del repositorio local hay que introducir el siguiente comando:

git branch -d nombre_rama_a_eliminar

Si la rama aún contiene elementos que no se han fusionado con otras ramas, Git mostrará un error y no eliminará la rama, aunque podría forzarse la eliminación poniendo la d en mayúsculas:

git branch -D nombre_rama_a_eliminar

Importante

Como has podido ver, Git es un sistema de control de versiones bastante potente, ya que no solo permite a varias personas trabajar sobre un mismo proyecto mediante el uso de las ramas (cada persona podría trabajar en una rama de forma independiente), sino que además, es bastante inteligente a la hora de mezclar el trabajo de las mismas para conseguir el proyecto final (fusionando y mezclando ramas), haciendo todas las fusiones que detecta que puede hacer e incluso marcando las zonas donde hay conflictos. Sin embargo, cuando ha detectado conflictos, deja al usuario la responsabilidad resolverlos, además de forma manual. Esto genera una pérdida de tiempo doble, ya que al tiempo que han perdido los autores de los cambios al trabajar sobre la misma zona del proyecto, hay que sumarle el tiempo que hay que dedicarle para resolver el conflicto. Toda esta pérdida de tiempo, y los conflictos generados, se puede evitar teniendo en cuenta dos aspectos esenciales:

1. **Hacer una buena planificación en el reparto de las tareas del proyecto** entre los miembros del grupo de trabajo para evitar que dos personas trabajen sobre la misma información.
2. Estableciendo una **comunicación constante** entre los miembros del grupo de trabajo.

Recuperando un commit anterior...

Si en algún momento necesitamos recuperar un estado anterior de nuestro proyecto, podemos utilizar el mismo comando que nos cambia de rama, es decir, git checkout, pero en lugar de escribir el nombre de la rama a la que nos queremos mover, ponemos el código identificador del commit al que queremos volver. Su sintaxis por tanto es la siguiente:

git checkout código_commit

Este código se puede ver tecleando el comando git log --oneline. Por ejemplo, supongamos que al teclear el comando ***git log --oneline*** obtenemos la siguiente información:

79a4e5f Tercer commit del proyecto.
f449007 Segundo commit del proyecto.
55df4c2 Primer commit del proyecto.

Git nos ofrece un listado con los commit que se han realizado en el proyecto, con sus códigos y sus descripciones. Si necesitamos devolver el proyecto al estado en que se encontraba en el primer commit teclearíamos:

git checkout 55df4c2

Git recuperará el commit especificado y automáticamente, en el directorio de trabajo, obtendremos los archivos y la información del proyecto exactamente como estaban cuando se hizo el primer commit.

También tenemos la opción de recuperar un solo archivo (o un grupo de ellos) y no el proyecto completo. Para ello bastaría con añadir al comando el nombre del archivo que queremos recuperar. Por ejemplo, supongamos que del primer commit solo necesitamos recuperar un único archivo llamado "funciones.php", el cual deseamos recuperar en el estado en que se encontraba en ese primer commit, teclearíamos el siguiente comando:

git checkout 55df4c2 funciones.php

Creando tags (etiquetas)...

Un tag o etiqueta es un alias o nombre que se puede asignar al código de un commit para así poder hacer referencia al mismo sin tener que especificar dicho cógido, sino solo el nombre de etiqueta o tag asignado al mismo. Los tags nos pueden servir para establecer nombres o números de versiones a algunos commits determinados a lo largo del proyecto, para facilitar las recuperaciones de las distintas versiones de un mismo proyecto. La sintaxis del comando que crea un tag es la siguiente:

git tag nombre_tag

Este comando crearía una etiqueta o tag para el commit en el que nos encontramos actualmente. Por ejemplo, podríamos teclear: ***git tag Versión3.0***

También podemos crear un tag para otro commit distinto especificando su código de esta forma:

git tag nombre_tag código_commit

Por ejemplo: ***git tag Versión1.0 55df4c2***

Si queremos ver todos los tags creados en el proyecto podemos teclear el siguiente comando:

git tag

Añadiendo repositorios remotos...

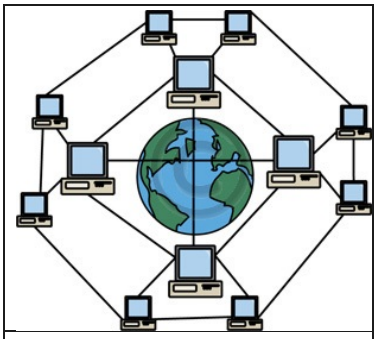


Imagen en Flickr de LearnerWeb con Algunos derechos reservados

Para poder enviar el contenido de nuestro repositorio local a un repositorio remoto, previamente debemos añadirlo a nuestro repositorio local creando un enlace al mismo. Usaremos el siguiente comando:

git remote add nombre_repositorio_a_asignar url_repositorio_remoto

Por ejemplo, para añadir un repositorio llamado "origin" (suele ser el primer repositorio remoto que se crea) que conecte con el repositorio remoto cuya url es "http://github.com/torvalds/linux", teclearíamos el siguiente comando:

git remote add origin http://github.com/torvalds/linux

Consultando los repositorios remotos...

Para poder ver los distintos repositorios remotos que tenemos conectados con nuestro repositorio local telcearemos el siguiente comando:

git remote -v

El parámetro -v es opcional, si se pone se mostrarán los repositorios remotos y las url's que cada uno tiene asociada. En caso contrario solo aparecerán los nombres que le hemos asignado a los repositorios.

Enviando cambios al repositorio remoto...

Para enviar cambios a un repositorio remoto utilizaremos el comando push de la siguiente manera:

git push nombre_repositorio_remoto nombre_rama_a_enviar

Por ejemplo, si queremos enviar la rama master de nuestro repositorio local al repositorio remoto origin lo haremos de la siguiente forma:

git push origin master

Recibiendo cambios del repositorio remoto...

Con el comando git pull podemos sincronizar los cambios con los existentes en un repositorio remoto, su sintaxis es la siguiente:

git pull nombre_repositorio_remoto nombre_rama

Por ejemplo, si queremos descargar a nuestro repositorio local la rama master del repositorio remoto origin teclearíamos:

git pull origin master

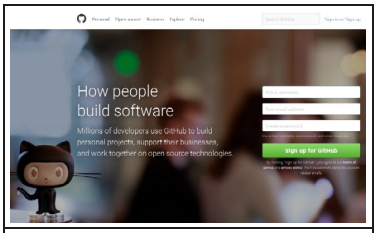
Conviene realizar sincronizaciones frecuentes, ya que si las realizáramos de tarde en tarde, correríamos el riesgo de que nuestro repositorio local fuese muy diferente del repositorio remoto cuando sincronicemos, y por tanto podrían saltar muchos conflictos e incongruencias.

Para saber más

Cuando se trabaja con repositorios remotos, además de las ramas que tenemos en nuestro repositorio local, también tendremos las ramas de los repositorios remotos. Por ejemplo, existirá una rama master en nuestro repositorio local y otra rama master en el repositorio remoto. Para ver todas las ramas, tanto locales como remotas, puedes utilizar el comando:

git branch --all

5. Facilita el trabajo con Github



Fuente propia

Git es el sistema de control de versiones utilizado en la web por muchos servidores para alojar sus repositorios remotos, uno de esos casos es el de Github. **Github es una plataforma de desarrollo colaborativo de software que utiliza Git para alojar proyectos**. Github nos permite, bien abriendo una cuenta de forma gratuita y pública, o bien haciéndolo de forma privada mediante pago, almacenar códigos de proyectos. Mediante la versión pública tendremos acceso a infinidad de repositorios y podremos colaborar con los autores originales de proyectos ofreciéndoles modificaciones o añadidos a dichos programas. A su vez todos nuestros proyectos serán accesibles públicamente para otros desarrolladores de la misma forma. Si por el contrario estamos elaborando un proyecto importante, bajo contrato, o bien bajo patrocinio privado, o que simplemente por motivos personales no queremos que sea difundido tendremos que usar una cuenta privada de pago. En este caso lógicamente dispondremos de más funcionalidades que en el caso de acceso público.

Además de ofrecer el alojamiento de repositorios de forma remota (repositorios remotos), también aporta una interfaz gráfica para el manejo del sistema de control de versiones, lo que facilita el uso del mismo. Podemos pensar el salto que supuso para el usuario pasar de utilizar sistemas operativos mediante la línea de comandos exclusivamente a usarlos con una interfaz gráfica a base de ventanas, iconos, cuadros y listas desplegables, barras de desplazamiento y otros elementos visuales. A modo de ejemplo y de forma exclusivamente aclaratoria, fuera del rigor docente, podríamos pensar que ocurre lo mismo con Git y Github, es decir esta plataforma emplea una interfaz gráfica que evita tener que operar con comandos de Git, siendo mucho más fácil el acceso y la gestión de toda la información. Así, dispone de cuadros gráficos con la secuencia de modificaciones o commits hechas sobre un proyecto indicando fechas, usuarios, etc. de forma directa, o bien por ejemplo de una relación de repositorios con los que estamos trabajando pudiendo pasar más fácilmente de trabajar con unos a hacerlo con otros.

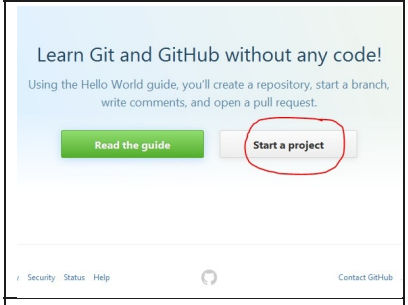
Github es, por tanto, un hosting para el sistema de control de versiones Git, **gratuito para proyectos opensource**, por lo que puedes utilizarlo para alojar tu repositorio de código y servirte de sus herramientas, que son muy útiles para el trabajo en equipo, destacando entre ellas las siguientes:

- Una wiki para el mantenimiento de las distintas versiones de las páginas.
- Un sistema de seguimiento de problemas que permiten a los miembros de tu equipo detallar un problema con tu software o una sugerencia que deseen hacer.
- Una herramienta de revisión de código, donde se pueden añadir anotaciones en cualquier punto de un fichero y debatir sobre determinados cambios realizados en un commit específico.
- Un visor de ramas donde se pueden comparar los progresos realizados en las distintas ramas de nuestro repositorio.

Además de eso, puedes contribuir a mejorar el software de los demás.

Pero Github no solo proporciona la gestión de repositorios remotos en internet, también posee una versión de Escritorio: **Github Desktop**. Esta versión es una aplicación instalable en tu propio ordenador para gestionar tus repositorios locales y comunicarlos con otros remotos.

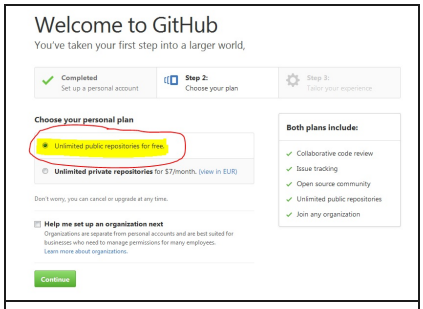
Analicemos primero el uso de Github en su versión gratuita en la web...



Fuente propia

Una vez instalado y configurado Git en tu ordenador, accederemos a la web de Github para registrarnos y abrir nuestra cuenta en versión pública (gratuita), tecleando nuestros datos personales en un formulario como en cualquier otro servicio remoto en la nube. No tiene mayor dificultad, tan solo hay que tener la precaución de elegir la opción gratuita, como se muestra en la imagen de la derecha.

A partir de ese momento, solo tendrás que confirmar el alta con un correo electrónico que recibirás. Realizada dicha confirmación ya podrás acceder a la web con tu nombre de usuario y contraseña y comenzar a trabajar con Github.

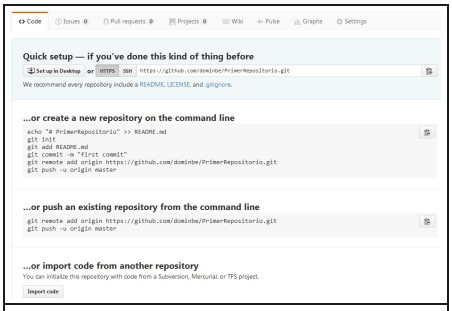


Fuente propia

El primer paso será crear un repositorio mediante el **botón "Start a project"** (posteriormente aparecerán más botones que permitirán crear nuevos proyectos). Github te pedirá el nombre del repositorio y una descripción que es opcional, además del grado de privacidad del mismo (recuerda que **deberás elegir un repositorio público para que sea gratuito**). Una vez creado el proyecto ya lo podrás ver en tu perfil.

El repositorio aparecerá en pantalla vacío, ya que aún no se ha realizado ninguna operación con él, es por lo que Github mostrará información de ayuda para agregar datos a nuestro repositorio. Hay varias posibilidades para ello:

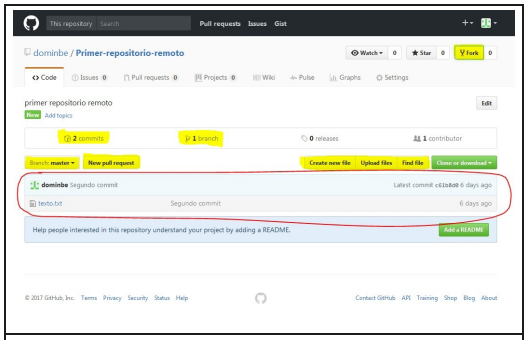
- Crear desde la línea de comandos, en Git, un repositorio local en nuestro ordenador, enlanzando en él el repositorio remoto recién creado en Github y subir al mismo dicho repositorio local. Para ello, Github nos ofrece los comandos necesarios.
- Agregar a un repositorio local que ya tengamos en nuestra máquina el repositorio remoto recién creado y subir el repositorio local al remoto. Github nos mostrará los comandos que necesitaríamos para ello.
- Importar código procedente de otro repositorio remoto que ya exista en Github. En este caso Github nos solicitará la url del repositorio remoto del que importar.



Fuente propia

En esta ventana de ayuda puedes ver, entre otros detalles, la url del repositorio remoto, que te servirá para hacer la conexión del mismo con otros repositorios, ya sean locales o remotos.

Puedes comenzar creando en tu repositorio local (el que tengas creado con Git en tu ordenador) los archivos de tu proyecto y luego subirlos con dichos comandos a la página. Aunque si no tienes aún ningún repositorio local puedes crearlo en este momento y enlazarlo directamente con el repositorio remoto que acabas de crear en Github.

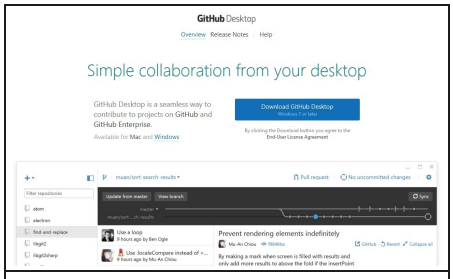


Fuente propia

A partir de aquí, ya verás en tu repositorio el código que hayas subido procedente de otros repositorios (ya sean estos últimos locales o remotos): archivos, commits, ramas, etc. En este momento Github habilitará una serie de opciones y **botones que te permitirán añadir al repositorio y gestionar nuevos archivos, nuevas ramas, hacer commits, etc**. Además también podrás mover información entre tus repositorios locales y los remotos que tengas alojados en la web.

Otra característica interesante de Github es que brinda la **posibilidad de colaborar en otros proyectos que no sean de tu propiedad**. Para colaborar en un proyecto ajeno simplemente **basta con buscarlo** dentro de los repositorios, **y luego presionar el botón fork**. Esto genera automáticamente una copia del mismo en tu perfil. **Al terminar tus modificaciones podrás presionar "Pull Request"** para enviárselo al creador del mismo, que decidirá si incorporar los cambios que propones a su proyecto o no.

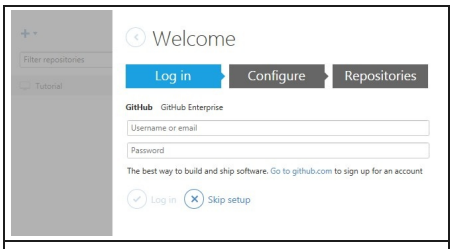
Pero Github guarda una sorpresa más. Además de ofrecer su web para gestionar repositorios remotos, cuenta con una aplicación de escritorio que permite gestionar tus repositorios locales y sincronizarlos con algún repositorio remoto de la web de Github. Se trata de la aplicación llamada **Github Desktop**.



Fuente propia

Efectivamente, también Github facilita al usuario la tarea del control de versiones aportando esta aplicación de escritorio que perfectamente puede realizar las tareas esenciales de los comandos de Git, no todos los comandos y con tanta potencia pero suple bastante bien lo esencial de Git, proporcionándole al usuario una interfaz gráfica muy fácil de utilizar. Puedes descargar e instalar Github Desktop sin problemas, ya que es software libre.

Cuando se inicia por primera vez Github Desktop aparece el asistente de configuración, que solicitará al usuario, mediante tres pequeños formularios, los datos esenciales para la aplicación, entre ellos se solicita el nombre de usuario y contraseña de la web de Github, para configurar las futuras conexiones con los repositorios remotos disponibles en la propia web. Posteriormente se buscará en el equipo los repositorios locales existentes y los mostrará si los encuentra, en caso contrario permitirá crear uno.

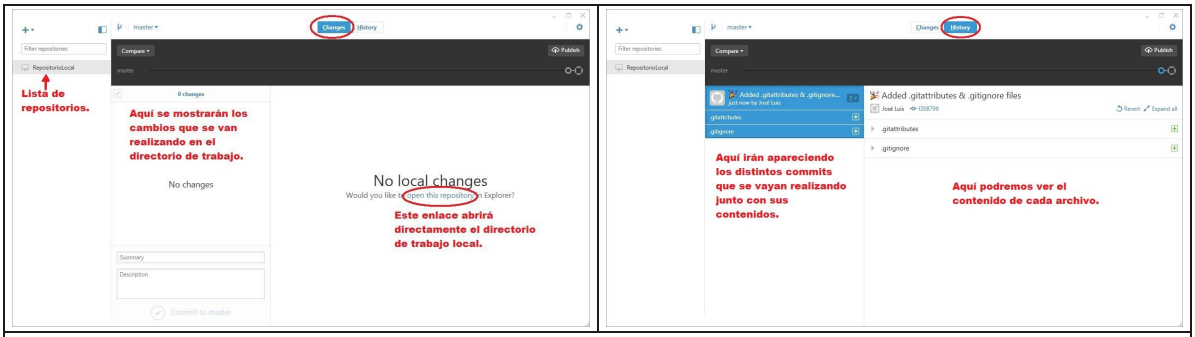


Fuente propia

Una vez que dispongamos del programa adecuadamente configurado y con un repositorio creado, el funcionamiento es muy intuitivo y sencillo.

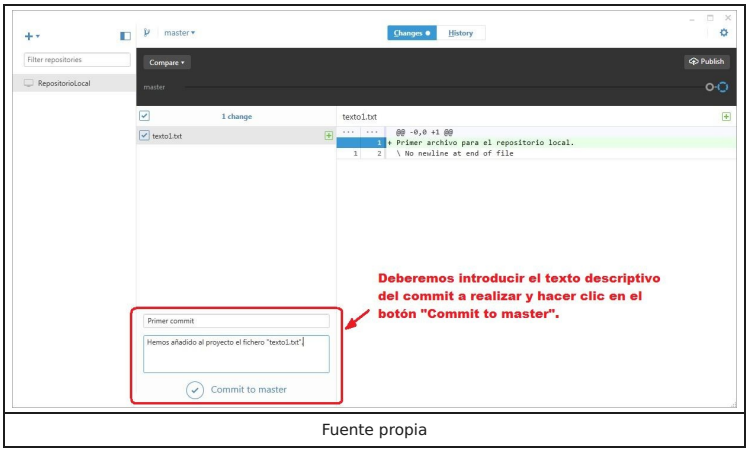
En la parte izquierda de la ventana de la aplicación nos irán apareciendo todos los repositorios que tengamos creados. Haciendo clic sobre el nombre de cada repositorio nos moveremos entre ellos, y la información que contiene cada uno irá apareciendo en el resto de la ventana. Deberemos hacer clic en el repositorio que deseemos controlar.

Cuando seleccionamos un repositorio de la lista, en la parte superior de la ventana correspondiente al mismo encontramos dos botones llamados "Changes" e "History". **El botón "Changes" mostrará los cambios realizados en la zona correspondiente al directorio de trabajo**, con los archivos que estén pendientes de ser confirmados para pasar al repositorio. **El botón "History" mostrará la lista de commits realizados con sus descripciones y contenidos, es decir, el repositorio local**.



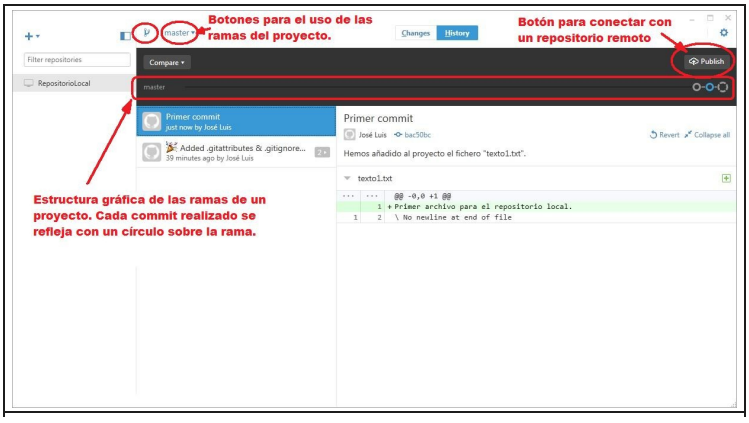
Fuente propia

Una vez que comencemos a realizar cambios en el directorio de trabajo, dichos cambios irán apareciendo en la ventana de cambios pendientes. Cuando necesitemos realizar un commit, en la parte inferior izquierda podremos escribir los mensajes descriptivos del commit y pulsar el botón "Commit to master". Para asegurar que cada confirmación lleva asociada una descripción, este botón no aparecerá activo hasta que se introduzcan los mensajes descriptivos del commit a realizar.



Fuente propia

La aplicación también cuenta con botones para crear ramas, cambiarnos entre ellas y visualizar la estructura de ramas que contiene el proyecto. Las opciones del manejo de ramas las puedes encontrar en la parte superior de la ventana. Además, dispone de un botón llamado "Publish", situado también en la parte superior, en la zona derecha, que permitirá conectar nuestros repositorios locales con un repositorio remoto. En la imagen inferior puedes apreciar todos estos detalles.



Fuente propia

Como puedes apreciar, el aporte de la interfaz gráfica que ofrece la aplicación es bastante bueno, intuitivo y muy fácil de utilizar, lo que permite un control básico de versiones de nuestros repositorios prácticamente sin necesidad de acudir a la línea de comandos de Git. No obstante, no ofrece la potencia total de los comandos de Git y por tanto no podría considerarse un sustituto de Git. Aun así, es una buena herramienta para iniciarse en el mundo de los Sistemas de Control de Versiones.

6. Inténtalo tú mismo

Tú mismo puedes instalar Git en tu equipo, crear un repositorio local y probar cómo funciona. Empieza por acceder a la web de Git y descargar la versión adecuada para tu sistema operativo. Si tienes Linux no necesitarás acceder a la web de Git, ya que puedes instalarlo utilizando el gestor de paquetes del sistema operativo, con las sentencias apt-get o equivalentes. En cualquier caso, aquí tienes información para instalar Git en cualquier sistema operativo.

La instalación en sistemas Windows es similar a todas las aplicaciones para estos entornos, solo necesitas responder a unas pocas preguntas, aunque prácticamente puedes dejar intactas las propuestas del asistente de instalación en todas ellas. En cualquier caso, aquí tienes una explicación que te puede ayudar.

Imagen en Pixabay de geralt bajo licencia CC0 Public Domain

Una vez instalado Git, intenta los siguientes ejercicios:

1. Crea un repositorio local con el comando "git init" a partir de una carpeta vacía de tu ordenador.
2. Configura Git con tus datos personales (nombre y correo electrónico).
3. Crea en el directorio de trabajo un archivo de textos llamado "texto1.txt" y escribe una frase en él.
4. Consulta el estado del repositorio con "git status".
5. Añade al staging index el archivo texto1.txt.
6. Realiza tu primer commit especificando el mensaje "Primer commit".
7. Crea en el directorio de trabajo un segundo archivo de textos llamado texto2.txt y escribe dentro tu nombre y apellidos.
8. Modifica el archivo "texto1.txt" añadiéndole una segunda frase.
9. Consulta el estado del repositorio para ver lo que hay pendiente de controlar.
10. Añade al staging index todo lo que hay pendiente.
11. Realiza el segundo commit con el mensaje "Creado texto2 y modificado texto1".
12. Vuelve a editar el archivo "texto1.txt" y añade en él tu dirección postal.
13. Realiza un tercer commit con el mensaje "Añadida dirección en texto1".
14. Crea una rama llamada "rama1".
15. Cámbiate a la rama1.
16. Crea otro archivo llamado "texto3.txt" y escribe en él una de tus aficiones.
17. Haz un commit con el mensaje "Creado texto3".
18. Vuelve a la rama master.
19. Modifica el archivo "texto1.txt" y añadele una línea más describiendo otra de tus aficiones.
20. Añade todos los ficheros con extensión txt al staging index.
21. Realiza otro commit con el texto "Último commit".
22. Lista todos los commits en una línea por commit.
23. Recupera el segundo commit que hiciste (que que tiene el mensaje "Creado texto2 y modificado texto1").
24. Comprueba ahora los ficheros que tienes en tu directorio de trabajo, y el contenido de los archivos que tienes.
25. ¿Puedes volver a recuperar el último commit?
26. ...

Ahora puedes seguir investigando y probando tú mismo el resto de comandos que ya conoces, ánimo, el potencial de la herramienta y las posibilidades son enormes.

Comprueba lo aprendido

El Sistema de Control de Versiones (VCS):

- ◯ Es prácticamente necesario para grandes proyectos industriales de software.
- ◯ Es un sistema obsoleto de control.
- ◯ Es innecesario si se nombran bien los archivos y se organizan adecuadamente.

Opción correcta

Incorrecto

Incorrecto

Solución

1. Opción correcta (Retroalimentación)
2. Incorrecto (Retroalimentación)
3. Incorrecto (Retroalimentación)

Un repositorio:

- ◯ Es un programa restaurador de archivos a un estado anterior.
- ◯ Es un almacén o depósito de archivos.
- ◯ Es un programa organizador de directorios.

Incorrecto

Opción correcta

Incorrecto

Solución

1. Incorrecto (Retroalimentación)
2. Opción correcta (Retroalimentación)
3. Incorrecto (Retroalimentación)

El historial informático de un proyecto:

- ◯ Se almacena en el propio programa del proyecto.
- ◯ Se almacena en repositorios locales y remotos.
- ◯ Se almacena en repositorios pero no es accesible con gran facilidad.

Incorrecto

Opción correcta

Incorrecto

Solución

1. Incorrecto (Retroalimentación)
2. Opción correcta (Retroalimentación)
3. Incorrecto (Retroalimentación)

En el historial de un proyecto:

- ◯ Dentro del repositorio figuran las modificaciones del proyecto pero sin definición temporal.
- ◯ Figuran con detalle de fecha y hora todas las modificaciones.
- ◯ No figuran los usuarios que han efectuado cambios.

Incorrecto

Opción correcta

Incorrecto

Solución

1. Incorrecto (Retroalimentación)

2. Opción correcta (Retroalimentación)
3. Incorrecto (Retroalimentación)

Todo lo referente a un mismo proyecto se denomina:

- ◯ Módulo.
- ◯ Repositorio.
- ◯ Línea Base.

Opción correcta

Incorrecto

Incorrecto

Solución

1. Opción correcta (Retroalimentación)
2. Incorrecto (Retroalimentación)
3. Incorrecto (Retroalimentación)

Para rotular una versión de un proyecto:

- ◯ No se rotulan proyectos sino repositorios.
- ◯ Es necesario congelar el módulo.
- ◯ Es indiferente rotular con o sin congelación.

Incorrecto

Opción correcta

Incorrecto

Solución

1. Incorrecto (Retroalimentación)
2. Opción correcta (Retroalimentación)
3. Incorrecto (Retroalimentación)

Tres ramas de un mismo módulo:

- ◯ Siguen interconectadas y dependientes entre si durante el desarrollo del proyecto.
- ◯ Evolucionan de forma independiente.
- ◯ No son posibles más de dos ramas de un mismo módulo.

Incorrecto

Opción correcta

Incorrecto

Solución

1. Incorrecto (Retroalimentación)
2. Opción correcta (Retroalimentación)
3. Incorrecto (Retroalimentación)

Un Commit es:

- ◯ Una copia efectuada en el repositorio de la copia de trabajo local.
- ◯ Una llamada de acceso a un repositorio remoto.
- ◯ La fusión (merge) entre un repositorio local y uno remoto.

Opción correcta

Incorrecto

Incorrecto

Solución

1. Opción correcta (Retroalimentación)
2. Incorrecto (Retroalimentación)
3. Incorrecto (Retroalimentación)

Existen VCS:

- ◯ Locales y remotos.
- ◯ Centralizados y distribuidos.
- ◯ Locales y distribuidos.

Incorrecto

Opción correcta

Incorrecto

Solución

1. Incorrecto (Retroalimentación)
2. Opción correcta (Retroalimentación)
3. Incorrecto (Retroalimentación)

Git es:

- ◯ Un tipo avanzado de repositorio.
- ◯ Un sistema de control de versiones centralizado.
- ◯ Un sistema de control de versiones distribuido.

Incorrecto

Incorrecto

Opción correcta

Solución

1. Incorrecto (Retroalimentación)
2. Incorrecto (Retroalimentación)
3. Opción correcta (Retroalimentación)

En los VCS distribuidos:

- ◯ Se trabaja en el repositorio central.
- ◯ Existe dependencia de cada desarrollador respecto del repositorio central.
- ◯ Cada usuario trabaja independientemente.

Incorrecto

Incorrecto

Opción correcta

Solución

1. Incorrecto (Retroalimentación)
2. Incorrecto (Retroalimentación)
3. Opción correcta (Retroalimentación)

Una de las ventajas de los VCS distribuidos es que:

- ◯ Todo lo importante lo controla un único usuario administrador del repositorio central.
- ◯ La información está muy replicada lo que ofrece gran seguridad.
- ◯ Gran parte del trabajo se realiza en el repositorio remoto.

Incorrecto

Opción correcta

Incorrecto

Solución

1. Incorrecto (Retroalimentación)
2. Opción correcta (Retroalimentación)
3. Incorrecto (Retroalimentación)

Cuando un trabajador que trabaja con Git quiere iniciar una sesión de trabajo:

- ◯ Clona el repertorio remoto desde la web de Git e inicia el trabajo en su copia de trabajo local.
- ◯ Inicia sin más, y después sube información a su repertorio local.
- ◯ Lo primero que debe hacer es un Commit desde su copia de trabajo local a su repositorio.

Opción correcta

Incorrecto

Incorrecto

Solución

1. Opción correcta (Retroalimentación)
2. Incorrecto (Retroalimentación)
3. Incorrecto (Retroalimentación)

Existen repositorios:

- ◯ Institucionales y de datos.
- ◯ Institucionales y temáticos.
- ◯ Institucionales, temáticos y de datos.

Incorrecto

Incorrecto

Opción correcta

Solución

1. Incorrecto (Retroalimentación)
2. Incorrecto (Retroalimentación)
3. Opción correcta (Retroalimentación)

Comprueba lo aprendido

Para trabajar como equipo es necesario promover una buena comunicación entre el conjunto de los integrantes del mismo.

◯ Verdadero ◯ Falso

Verdadero

Para trabajos de envergadura son fundamentales la coordinación y el compromiso de todos sus desarrolladores.

◯ Verdadero ◯ Falso

Verdadero

El trabajo en equipo disminuye la eficacia de los resultados.

◯ Verdadero ◯ Falso

Falso

El trabajo en equipo retrasa la consecución de los objetivos de las organizaciones.

◯ Verdadero ◯ Falso

Falso

La base de la programación modular radica en la independencia de programadores que nunca implementan partes de otros proyectos trabajando exclusivamente en sus propios módulos.

◯ Verdadero ◯ Falso

Falso

Todos los ficheros que se refieren a un mismo proyecto constituyen lo que se denomina repositorio.

◯ Verdadero ◯ Falso

Falso

La rama "master" es el directorio principal del proyecto.

◯ Verdadero ◯ Falso

Verdadero

Un Sistema de Control de Versiones (VCS) hace posible revertir un proyecto a un estado anterior en el tiempo.

◯ Verdadero ◯ Falso

Verdadero

Team Foundation Server es un ejemplo de VCS distribuido.

◯ Verdadero ◯ Falso

Falso

Git es un ejemplo de VCS centralizado.

◯ Verdadero ◯ Falso

Falso

Los repositorios institucionales son creados por personas y están enfocados hacia temas concretos.

◯ Verdadero ◯ Falso

Falso

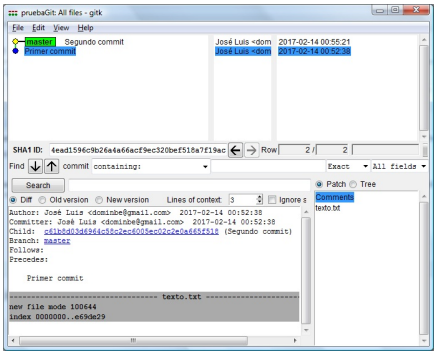
El objetivo perseguido por los repositorios institucionales es depositar, facilitar y preservar información.

◯ Verdadero ◯ Falso

Verdadero

Más entornos gráficos para Git

Ya has comprobado que, además de manipular Git mediante la línea de comandos, existen herramientas gráficas que te facilitan el trabajo, pero Github no es la única...



Fuente propia

Si observas bien la ayuda de Git, podrás comprobar que el propio Git lleva ya instalado un entorno gráfico llamado **gitk**. Para acceder a él solo tienes que teclear en la línea de comandos lo siguiente:

gitk

En este entorno gráfico podrás realizar las acciones más comunes de Git. Puedes ver documentación sobre el mismo en esta web.

También cuentas con otro entorno gráfico más al instalar Git, denominado **git-gui**. Igualmente puedes acceder a él tecleando en la línea de comandos cualquiera de los dos comandos siguientes:

git-gui

git gui

La lista de entornos gráficos para el uso de Git no acaba aquí, en realidad hay muchos más, a continuación lo vas a poder comprobar.

Además de los entornos gráficos de GitHub, Gitk y Git-gui, tienes más opciones. Si quieres conocer más entornos gráficos para manejar tus repositorios de Git, puedes acceder a los siguientes:

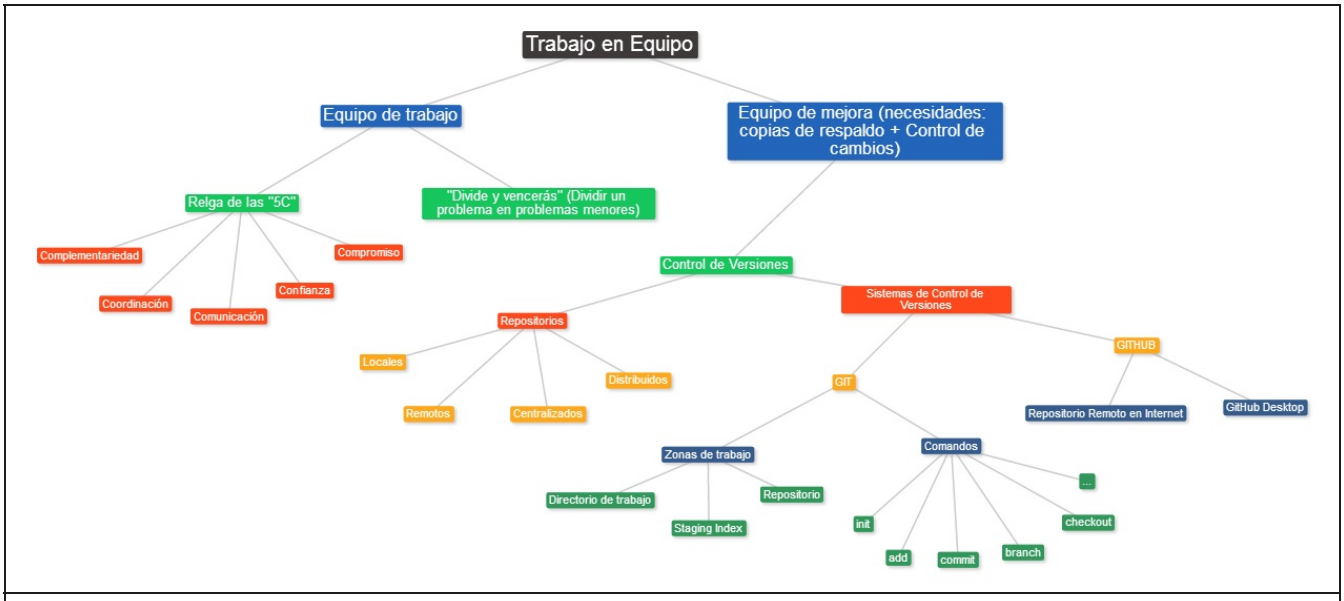
- Git Kraken.
- SourceTree.
- GitEye.

Aunque si no tienes suficiente, en la propia web oficial de Git se especifican muchos más. Puedes acceder a ellos haciendo clic aquí.

Importante

A pesar de existir una gran variedad de entornos gráficos para Git, ninguno tiene la potencia total y absoluta que aporta la línea de comandos, así pues, no olvides que no debes dejarla a un lado. Lo ideal para sacar un buen rendimento a Git es complementar y simultanear el uso de ambos métodos: la sencillez de los entornos gráficos y la potencia de la línea de comandos.

Como ves, hay bastantes opciones para manejar Git, por tanto, es una buena elección como Sistema de Control de Versiones. En cualquier caso, sea Git u otro sistema de control de versiones el que elijas, no dejes a un lado estas herramientas cuando necesites programar, ya sea de forma individual o en equipo, pues te facilitarán el trabajo y te ahorrarán tiempo y quebraderos de cabeza.



Fuente propia

Aviso legal

El presente texto (en adelante, el "**Aviso Legal**") regula el acceso y el uso de los contenidos desde los que se enlaza. La utilización de estos contenidos atribuye la condición de usuario del mismo (en adelante, el "**Usuario**") e implica la aceptación plena y sin reservas de todas y cada una de las disposiciones incluidas en este Aviso Legal publicado en el momento de acceso al sitio web. Tal y como se explica más adelante, la autoría de estos materiales corresponde a un trabajo de la **Comunidad Autónoma Andaluza, Consejería de Educación (en adelante Consejería de Educación**).

Con el fin de mejorar las prestaciones de los contenidos ofrecidos, la Consejería de Educación se reservan el derecho, en cualquier momento, de forma unilateral y sin previa notificación al usuario, a modificar, ampliar o suspender temporalmente la presentación, configuración, especificaciones técnicas y servicios del sitio web que da soporte a los contenidos educativos objeto del presente Aviso Legal. En consecuencia, se recomienda al Usuario que lea atentamente el presente Aviso Legal en el momento que acceda al referido sitio web, ya que dicho Aviso puede ser modificado en cualquier momento, de conformidad con lo expuesto anteriormente.

1. Régimen de Propiedad Intelectual e Industrial sobre los contenidos del sitio web

1.1. Imagen corporativa

Todas las marcas, logotipos o signos distintivos de cualquier clase, relacionados con la imagen corporativa de la Consejería de Educación que ofrece el contenido, son propiedad de la misma y se distribuyen de forma particular según las especificaciones propias establecidas por la normativa existente al efecto.

1.2. Contenidos de producción propia

En esta obra colectiva (adecuada a lo establecido en el artículo 8 de la Ley de Propiedad Intelectual) los contenidos, tanto textuales como multimedia, la estructura y diseño de los mismos son de autoría propia de la Consejería de Educación que promueve la producción de los mismos.

Descargar PDF