**Índice de contenido**

1. [¿Qué es GIT?](Que%20es%20un%20GIT.docx#quees)
2. [¿Cómo funciona GIT?](Que%20es%20un%20GIT.docx#comofunciona)
3. [Cómo instalar GIT](Que%20es%20un%20GIT.docx#comoinstalar)
4. [Configuración inicial de GIT](Que%20es%20un%20GIT.docx#configuracion)

**¿Qué es GIT?**

GIT es uno de los muchos sistemas de control de versiones que tenemos a nuestra disposición. Seguramente tú, querido desarrollador, has vivido feliz durante años sin ni siquiera haber oído hablar de GIT en la empresa dónde trabajas, codeando directamente encima de los archivos y rezando para que tus semejantes no estén haciendo lo mismo en el mismo fichero.

Llevar el control de ficheros en un proyecto informático con un grupo de trabajo presencial relativamente bajo es sencillo. El problema se complica cuando cada uno es de su madre y de su padre, y más aún cuando los proyectos crecen en personal hasta límites insospechados (como suele ocurrir con el open source).

Es por ello que GIT (y en definitiva cualquier otro sistema de control de versiones) ofrece dos principales ventajas:

* **Recuperar versiones anteriores del proyecto:** GIT genera árboles de versiones con los proyectos, lo que permite en esencia volver a un punto anterior de forma sencilla. Si trabajas tú solo, y aún no has cerrado el archivo desde tu IDE preferida, podrás darle atrás para revertir cambios (hasta cierto punto). Pero en proyectos colaborativos ya no es tan sencillo. Además, permite llevar una especie de agenda con los cambios hechos por cada desarrollador, de tal forma que inmediatamente, todo el equipo está al tanto de las novedades, evitando la duplicidad de información en otros sistemas externos como puede ser el IRC.
* **Versatilidad y flexibilidad**: GIT permite trabajar a cada desarrollador desde su propio ecosistema, enviando los cambios a un nodo central, o bien directamente diseminándolo entre el resto de distribuciones del proyecto. De esta forma, se evita la duplicidad de ficheros de diferentes versiones. Basta con levantarte una mañana y descargarte el repositorio (sino se hace automáticamente), para tener la última versión de trabajo del proyecto, con tus cambios y con los del resto, y a sabiendas que todo lo anterior ha quedado correctamente documentado.

En el caso que nos compete, hablaremos del CVS para desarrolladores, pero seguramente a muchos se os habrá iluminado una bombilla. En efecto, GIT puede ser usado para cualquier proyecto colaborativo, ya sea de desarrollo, de diseño (por ejemplo, es muy usado para controlar versiones de diseño 3D o animación), multimedia, arquitectura, wikis,…

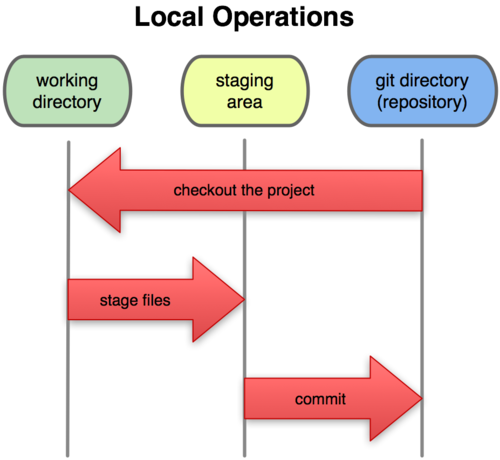
Existen varios tipos de CVS:

* **Control de versiones local:** el más primitivo, y que hereda el buen hacer de la mayoría de los desarrolladores noveles. Se basa en tener una carpeta en local con el proyecto real, y trabajar en otra carpeta con las nuevas versiones. Un paso más allá, es llevar mediante una base de datos el control de las versiones indexadas, de tal manera que podremos volver al inicial en cualquier momento.
* **Control de versiones centralizada:** O CVCSs, son aquellos que cuentan con un servidor con todos los archivos versionados, a los que se conectan cada cliente. Al haber un servidor central, la administración y control de los cambios de cada integrante se administra de forma mucho más sencilla, por lo que a priori es una fórmula más efectiva para trabajar en grupo que la local.
* **Control de versiones distribuida:** O DVCSs, son aquellos en los que tanto el servidor como el cliente tienen una copia completa de todos los archivos versionados, de tal manera que en la práctica, cualquier cliente puede ser servidor.

**¿Cómo funciona GIT?**

Me parece algo de vital importancia hablaros de la estructura interna de GIT, ya que si os queda bien clara, el resto del proceso es coser y cantar.

En GIT, existen **tres estados:**

1. Confirmado **(committed):** Significa que el archivo está guardado convenientemente en la base de datos.
2. Modificado **(modified):** El archivo se ha modificado, y aún no está almacenado.
3. Preparado **(staged):** El archivo está en espera de sea almacenado.

Atendiendo a la **arquitectura de GIT**, hablamos nuevamente de tres secciones:

1. Directorio GIT **(GIT directory):** Es el repositorio completo propiamente dicho. Aquí se almacenan los metadatos y todas las versiones del proyecto.
2. Área de trabajo **(Working Directory):** La carpeta donde trabajaremos, y podremos modificar los archivos.
3. Área de preparación **(Staging Area):** Se trata de un fichero que guarda toda la información referente a la próxima confirmación (commit) que enviaremos al Directorio GIT.

Lo que nos lleva al último punto. Las fases desde la creación de un cambio hasta su commit final:

En el directorio de trabajo, modificas algún archivo del proyecto, por lo que éstos archivos pasarán al estado modified.

Cambias el estado de todos los archivos modificados a staged, lo que hace que se actualize el área de presentación.

Haces el commit, lo que permite que la instantánea que tenías en staged pase a formar parte del repositorio guardado en el directorio GIT.

**Cómo instalar GIT**

El primer paso es instalar GIT en tu ordenador. Para ello, lo más cómodo es recurrir a los repositorios de cada sistema operativo.

1. Linux: Búscalo en el gestor de paquetes de tu distribución. Normalmente se llama git-core o git a secas. En el caso de Ubuntu, sería algo así:

$ apt-get install git

1. Windows: Entra en la web de GitHub (EN) e instala la interfaz gráfica (aunque no la vayas a volver a usar en tu vida).
2. Mac OS: Entra en la web de Google Code (EN) e instala la interfaz gráfica.

El siguiente paso es hacerlo compatible con tu IDE. Por supuesto, mediante la consola de comandos de tu sistema operativo puedes hacer los commit y gestionar cualquier directorio, pero por lo general, la mayoría de IDEs modernos te permiten hacerlo desde el propio entorno, sin tener que salir. En el caso de Sublime Text, bastaría con instalar el paquete Git, como ya expliqué en la segunda entrada sobre Sublime Text II, y que se traduce en 3 sencillos pasos:

Desde sublime text, le damos a ctrl+shift+P>Package Control, buscamos Package Control: Install Package, y tecleamos GIT.

Nos saldrá un error. Esto es debido a que la configuración del package no concuerda con el directorio GIT. Para solucionarlo, iremos a Preferences > Package Settings > Git > Settings – Default.

En el archivo que se nos abre, cambiamos el false de “git\_command” por “/usr/local/git/bin/git” (si estamos en Mac OS) o “C:/Archivos de programa (x86)/Git/bin/git.exe” en Windows.

**Configuración inicial de GIT**

Una vez tenemos instalado nuestro GIT en el terminal, hay que configurarlo correctamente para que funcione con los diferentes repositorios que estemos usando. Este paso se hace por tanto una sola vez por proyecto (o mejor dicho por repositorio), y será el último punto que trataremos en esta serie de dos entradas.

Básicamente la mayoría de configuraciones iniciales se harán con la herramienta git config, y como comprenderéis, tiene en su haber mil y un configuraciones distintas. En este tutorial, he resumido hasta lo básico que acabamos usando todos, pero puede ocurrir que algún proyecto con especificaciones muy complejas acabe por precisar una configuración más avanzada.

Para ello, vamos al terminal (consola de comandos o como diablos quieran llamarle) y configuramos nuestro usuario:

git config --global user.name "Tu Nombre"

git config --global user.email tu-email@servicio.com

Una vez hecho, configuramos el editor por defecto:

$ git config --global core.editor "url del editor"

* En mi caso, que uso Sublime Text 2, **en Mac OS** quedaría algo así:

git config --global core.editor "'/Applications/Sublime Text 2.app/Contents/SharedSupport/bin/subl' -w"

* **Y en Windows:**

git config --global core.editor "C:\Program Files\Sublime Text 2\sublime\_text.exe"

También he oído que se pueden usar acortadores genéricos en vez de las rutas al directorio. Algunos de ellos son:

* subl -w: Sublime.
* mate -w: TextMate.
* gvim -f: GVim.
* mvim -f: MacVim.

Para consultar la configuración inicial (útil por si has metido la pata):

$ git config --list

Hecho esto, ya tenemos correctamente configurado GIT, y accesible desde nuestro IDE. En la próxima entrega, empezaremos a juguetear con los comandos básicos de GIT, y hablaremos de GitHub como repositorio abierto.

<https://www.pabloyglesias.com/asktopablo-primeros-paso-con-git-control-de-versiones-i/>