Control de versiones: Git y GitHub

A lo largo de la vida del desarrollo de un software es muy habitual que se vayan produciendo cambios en el código, realizados por todos los miembros que trabajan en el proyecto. Para un buen funcionamiento y que todos tengan en todo momento las modificaciones que se han llevado a cabo, es necesario contar con una buena herramienta que permita la gestión del control de versiones. En el mercado hay muchas herramientas que ayudan a conseguir este objetivo pero a lo largo de este White Paper nos centraremos en la combinación de Git y GitHub, que ofrecen todo lo necesario para poder desarrollar software de calidad.

¿Qué es el control de versiones?

Uno de los sueños de todo el mundo es poder volver al pasado para solucionar algún tipo de problema que tengamos en el presente. Por desgracia, estos viajes temporales no son posibles en la vida real, pero al menos sí que es posible hacerlo en el desarrollo de software gracias al control de versiones.

Un sistema de control de versiones es una herramienta capaz de registrar todos los cambios que se realizan en uno o más proyectos, guardando a su vez versiones anteriores del proyecto, versiones a las que podemos acudir en caso de haber metido la pata o al no funcionar de la forma correcta.

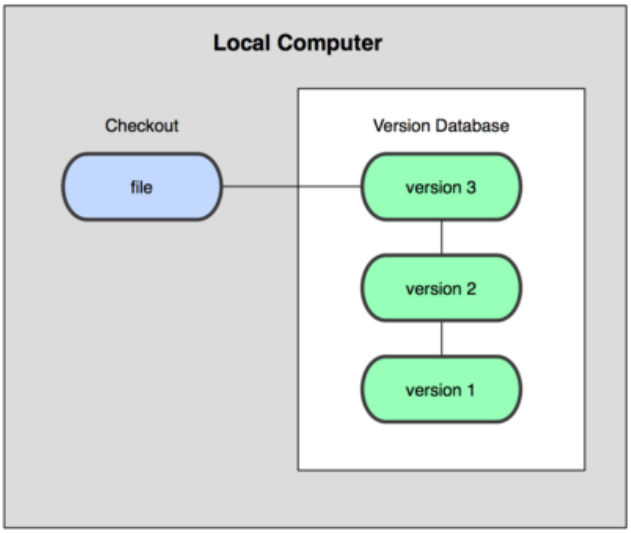
El uso de control de versiones no es algo nuevo, sino que se viene utilizando desde hace muchos años, aunque no de la forma en la que se puede hacer en la actualidad, y es que todo el mundo, alguna vez ante un cambio en el código de un proyecto, ha sacado copia del archivo en un directorio para volver al estado anterior en caso de que fallen los cambios. Ésta precisamente la principal ventaja que ofrecen los sistemas de control de versiones actuales.

Además de poder volver a un estado anterior, los actuales sistemas de versiones también ofrecen la posibilidad de comparar los cambios realizados a lo largo del tiempo, ver quién modificó algo en el proyecto que pueda estar causando problemas o el momento exacto en el que alguien introdujo un error en el código.

Clasificación de los sistemas de control de versiones

Dentro del campo de control de versiones, nos podemos encontrar de distintos tipo. Veamos a continuación una clasificación de los más importantes.

a) **Sistemas de control de versiones locales**
   Uno de los métodos más utilizados por la gente a la hora de realizar algún tipo de control de versión de sus cambios, consistía en copiar en un directorio de su equipo local el archivo que iba a ser modificado indicando la fecha de modificación, para que en caso de error se supiese cuál era la última versión guardada.

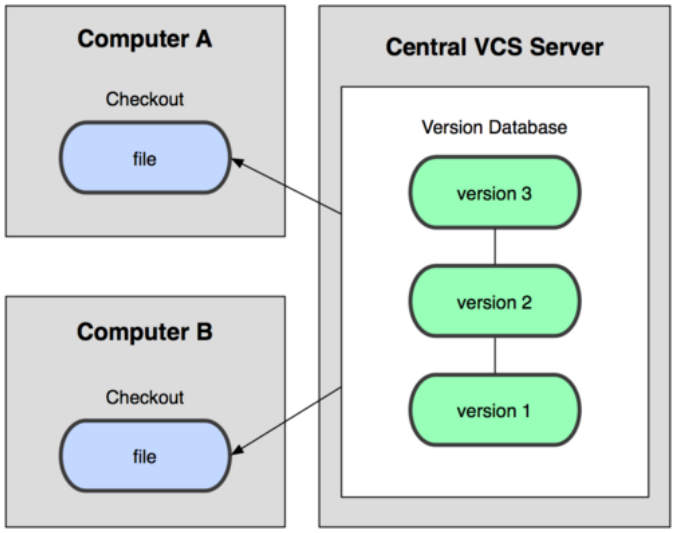


Este sistema podría valer para el desarrollo de una aplicación pequeña, pero ofrece ciertos problemas como el de no recordar dónde hemos guardado la copia o simplemente olvidarnos de hacerla.

Para hacer frente a estos problemas, los programadores desarrollaron los conocidos como VCSs locales, que consistían en una base de datos donde se llevaba un registro de los cambios realizados sobre los archivos.

b) Sistemas de control de versiones centralizados

El sistema comentado anteriormente nos puede valer en el caso de que estemos trabajando solos, pero en un proyecto suelen intervenir varias personas y cada una de ellas se encarga de realizar una tarea determinada. En este caso, es necesario poder contar con un sistema colaborador y es aquí donde entran en juego los sistemas de control de versiones centralizados.

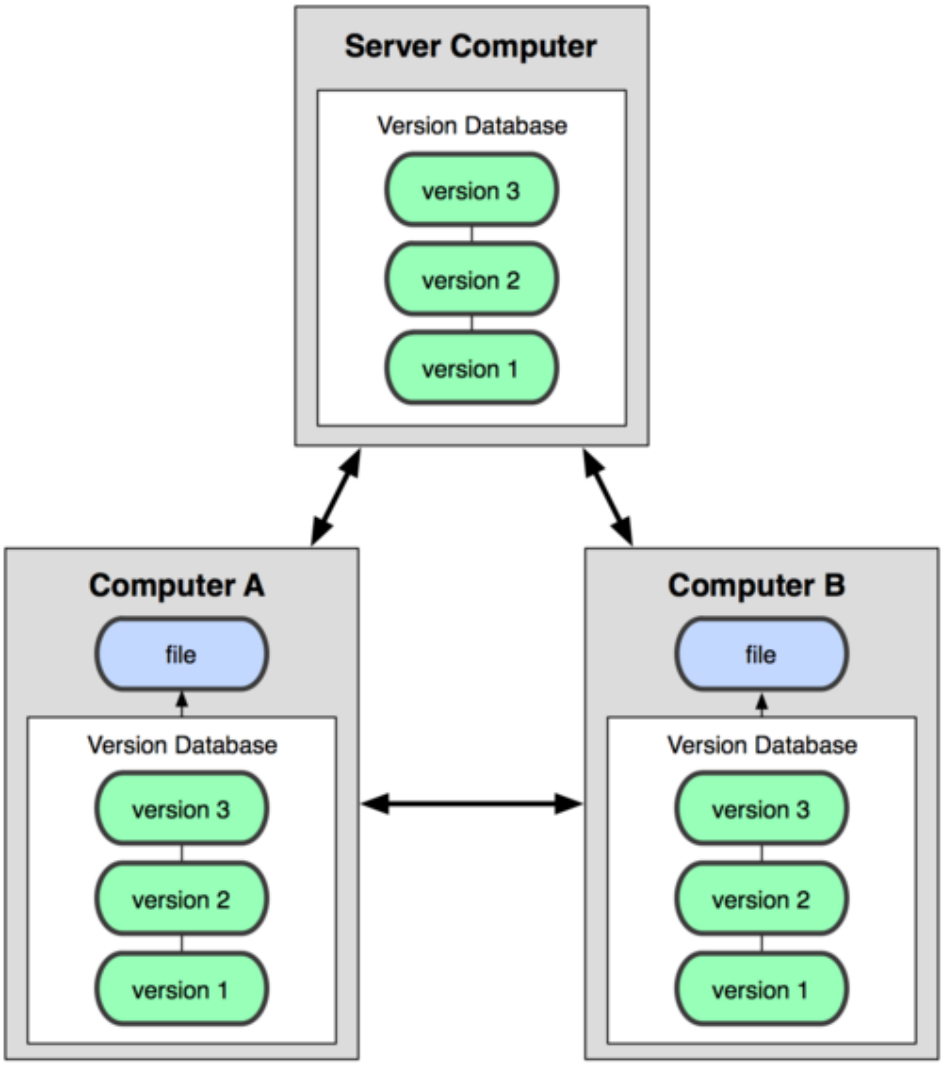


En estos sistemas nos encontramos un único servidor que contiene todos los archivos versionados, y los usuarios que forman parte del proyecto se los pueden descargar desde ese servidor centralizado.

Este sistema tiene un problema muy claro, y es que al utilizar un único servidor centralizado, en caso de problema en ese servidor toda la información se podría perder.

c) Sistemas de control de versiones distribuidas

A diferencia del caso anterior, en estos sistemas no tenemos un único servidor que mantenga la información del proyecto, sino que cada usuario contiene una copia completa del proyecto de forma local. De esta forma, si un servidor muere, cualquiera de los repositorios de los clientes se podría utilizar para restaurar el servidor.



Git pertenece a este tipo de sistemas, uno de los protagonistas de este libro blanco.

¿Qué es Git?

Git es un sistema de control de versiones distribuido, como habíamos indicado en el punto anterior, cuyo principal objetivo es ayudar en el desarrollo de cualquier tipo de aplicación manteniendo una gran cantidad de código de un gran número de programadores diferentes.

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 encontrárnoslo para las distintas plataformas del mercado. Para hacer uso de ella, lo único que debemos hacer es instalar en nuestro sistema la versión correspondiente al sistema operativo que estemos utilizando.

Podemos encontrar una gran diferencia del uso de Git respecto a otras herramientas similares. La primera de ella es que mientras que otros sistemas almacenan archivos originales, conservando una lista de los cambios realizados, Git guarda un snapshot del estado de cada archivo. Si el archivo no ha cambiado, no crea una nueva copia, sino que crea una referencia al archivo original.

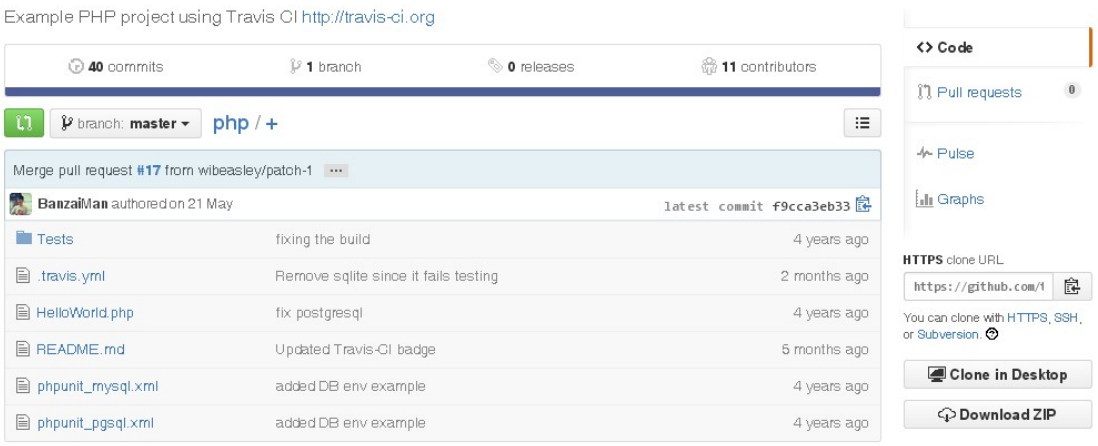
Con todo esto, Git nos aporta:

- Auditoría completa del código, sabiendo en todo momento quién ha tocado algo, cuándo y qué.
- Control sobre cómo ha ido cambiando nuestro proyecto con el paso del tiempo.
- Volver uno o más pasos hacia atrás de forma rápida.
- Control de versiones del proyecto por medio de etiquetas.
- Seguridad, ya que todas las estructuras internas de datos irán cifradas con el algoritmo SHA1.

¿Qué es GitHub?

Muchas personas pueden pensar que Git y GitHub son lo mismo, ya que la mayoría de las ocasiones están estrechamente relacionadas, pero a la hora de la verdad, son cosas totalmente distintas.

GitHub se puede definir como un servidor donde alojar los repositorios de los proyectos, añadiendo funcionalidades extra para la gestión del proyecto y del código fuente. A diferencia del proyecto Git, éste se trata de un servicio comercial, ya que aunque tiene una parte pública gratuita, cuenta con la desventaja de que todo el código que subamos estará disponible para cualquier persona. Si queremos decidir quién puede tener acceso a nuestro repositorio, entonces sería necesario pasarse a la modalidad de pago.

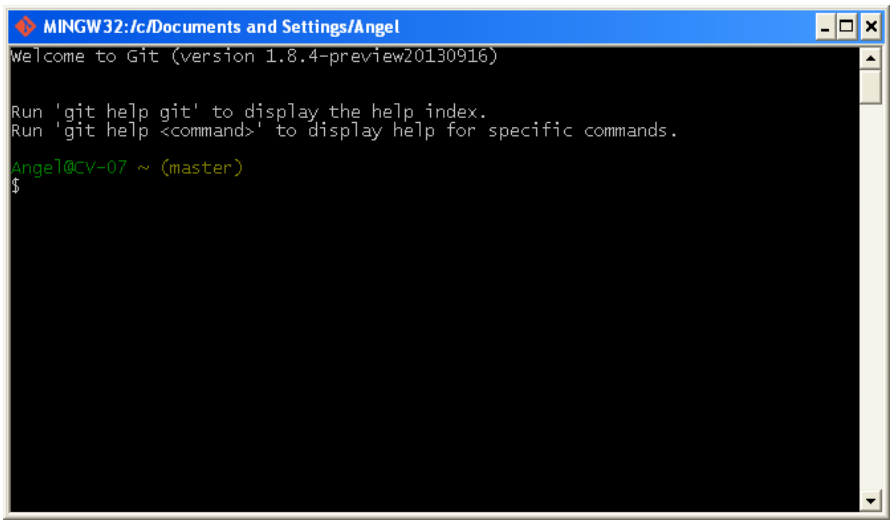


Entre las principales características que nos ofrece GitHub podemos destacar:

- Una Wiki que opera con Git para el mantenimiento de las distintas versiones de las páginas.
- Sistema de seguimiento de problemas. Se trata de un sistema muy parecido al tradicional ticket, donde cualquier miembro del equipo o persona (si nuestro repositorio es público) puede abrir una consulta o sugerencia que se quiera hacer.
- Herramienta de revisión de código. Permite añadir anotaciones en cualquier punto de un fichero.
- Visor de ramas que permite comparar los progresos realizados en las distintas ramas de nuestro repositorio.

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. 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.

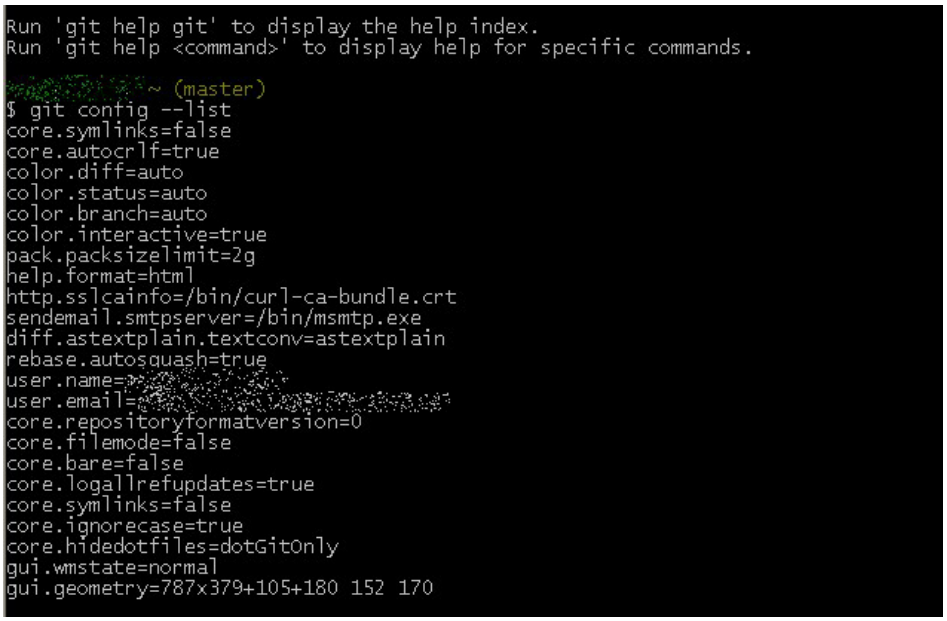
git config –global user.name “xxxxxxxx”

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



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

1.- Descargar un proyecto por primera vez

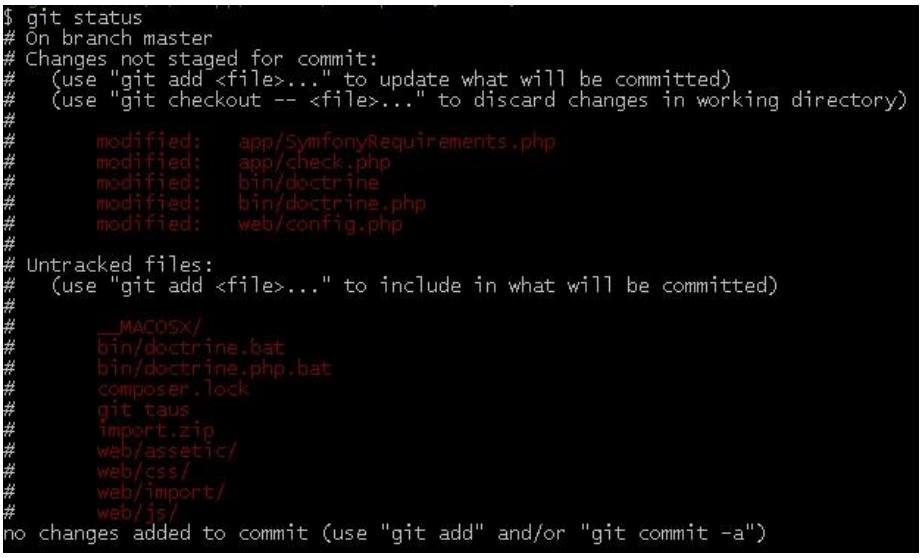
Para descargarnos un proyecto por primera vez, hay que utilizar el comando “git clone” seguido de la url de github que corresponde al proyecto.

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.

2.- 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”.



En este caso nos encontramos dos apartados bien diferenciados:

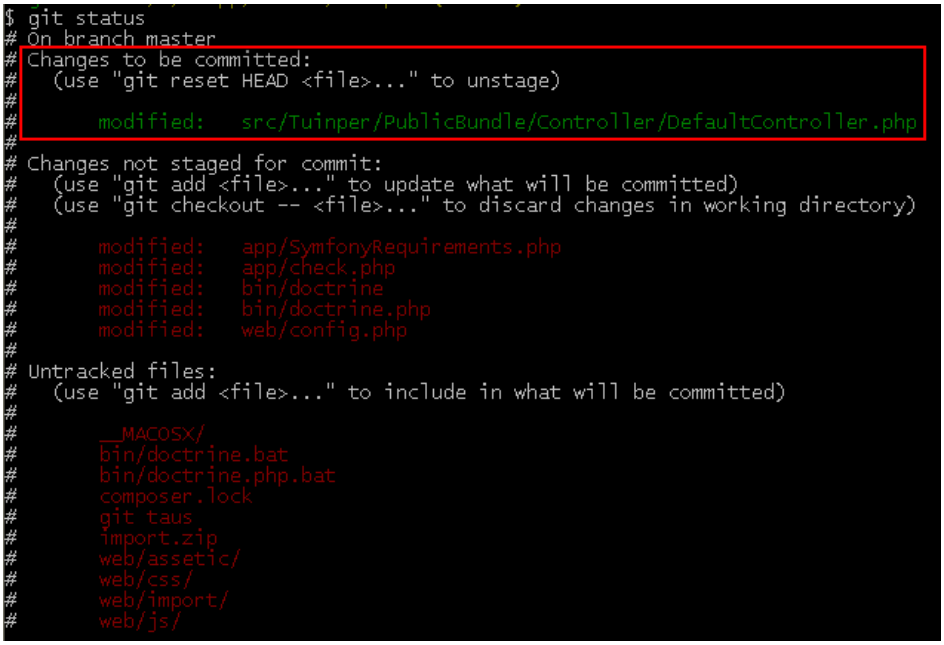
- **Changes not staged for commit**. Se corresponde con archivos ya existentes en el repositorio y que han sido modificados.
- **Untraked files**. En este caso corresponden a nuevos archivos que aún no han sido subidos al repositorio.

3.- 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 commited”.



4.- 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.

5.- 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': angelfcarrero
Password for 'https://angelfcarrero@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.