CURSO DE GIT Y GITHUB

# SISTEMA DE Control de versiones de proyectos

**Versión:** Estado en el que se encuentra el producto en un momento concreto del desarrollo, de esta forma podemos guardar y volver a recuperar cualquier versión que se van sucediendo a medida que el proyecto va madurando.

**Control de versiones :** Para poder volver a cualquier versión de un programa, es necesario disponer de un control de versiones y llevar una gestión de los cambios, para ello existen programas que n0s ayudan a realizar esta tarea.

**Sistemas de control de versiones:** Existen muchos sistemas de control de versiones como CVS, Mercurial, TortoiseSVN o Git. Todos ellos cuentan con las siguientes características son:

* Facilitan la gestión.
* Administran versiones.
* Almacenan información.
* Conservan registro.
* Son sistemas centralizados.

Estas son también algunas de las funcionalidades que nos ofrece un sistema de control de versiones:

* Modos de almacenamiento de los proyectos.
* Cuentan con la posibilidad de realizar cambios.
* Dependiendo del sistema, hay algunos que nos ofrecen la posibilidad de crear informes.
* Registro de un histórico en el que se pueden ver las diferentes versiones de los archivos.

Hay dos tipos de SCV:

|  |  |  |
| --- | --- | --- |
| **Centralizados**  Repositorio centralizado de todo el código con un único administrador, al tener un único administrador, todas las decisiones importantes las lleva a cabo ese responsable, como por ejemplo la creación de una nueva rama en un repositorio (CVS y Subversion). | | http://image.slidesharecdn.com/git-aprovecha-ventajas-control-versiones-distribuido-120216083822-phpapp02/95/git-aprovecha-las-ventajas-del-control-de-versiones-distribuido-5-728.jpg?cb=1336972365 |
| **Distribuidos**  Cada usuario tiene su propio repositorio, y los distintos repositorios pueden intercambiar archivos y versiones entre ellos (Mercurial, Git...) | http://image.slidesharecdn.com/git-aprovecha-ventajas-control-versiones-distribuido-120216083822-phpapp02/95/git-aprovecha-las-ventajas-del-control-de-versiones-distribuido-6-728.jpg?cb=1336972365 | |

## GIT

Git nace en el 2005, es un sistema de control de versiones distribuido, ofrece diferencias con respecto a otros sistemas de control de versiones, lo que hace que haya tomado mucha popularidad.

Está diseñado por LINUS TORVALDS.

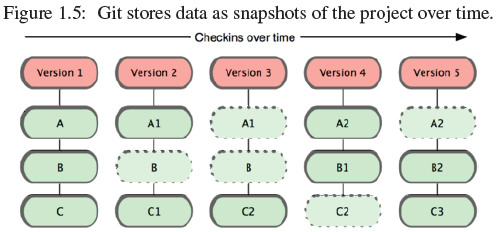
Git convierte cada uno de los ordenadores en un espejo en el que se puede recuperar cualquier versión del proyecto.

Este sistema distribuido hace que facilite muchísimo el trabajo entre varias personas, además de facilitar la gestión.

Git almacena y trata la información de una forma diferente a como lo hacen los sistemas de versiones centralizados, entre esas diferencias podemos destacar:

**DIFERENCIAS DE GIT CON RESPECTO A OTROS SISTEMAS**

* Realiza panorámicas completas no sólo diferencias entre versiones.
* Git piensa en los datos como si fuese un conjunto de fotos, cuando salvamos un estado del proyecto, Git almacena una visión global del proyecto, por lo que cada una de las versiones es un backup de todo el contenido, y no sólo de cada uno de los archivos que se han modificado como ocurre con otros sistemas.
* Todas las operaciones que se llevan a cabo son acciones locales, no es necesario información de otro ordenador para poder trabajar, en otros sistemas la información depende de la red. De esta forma cada una de las operaciones se realizan de forma instantanea.
* Tiene especial cuidado con el mantenimiento de integridad de datos, y el guardado de los datos, verificando que no hayan sido corrompidos, de esta forma, es imposible que se cambien los datos de un archivo sin que Git se entere.
* Las acciones de Git son modificables, cuando se lleva a cabo una acción es muy dificil que se puedan perder datos, después de realizar un commit, no se perderán los datos.



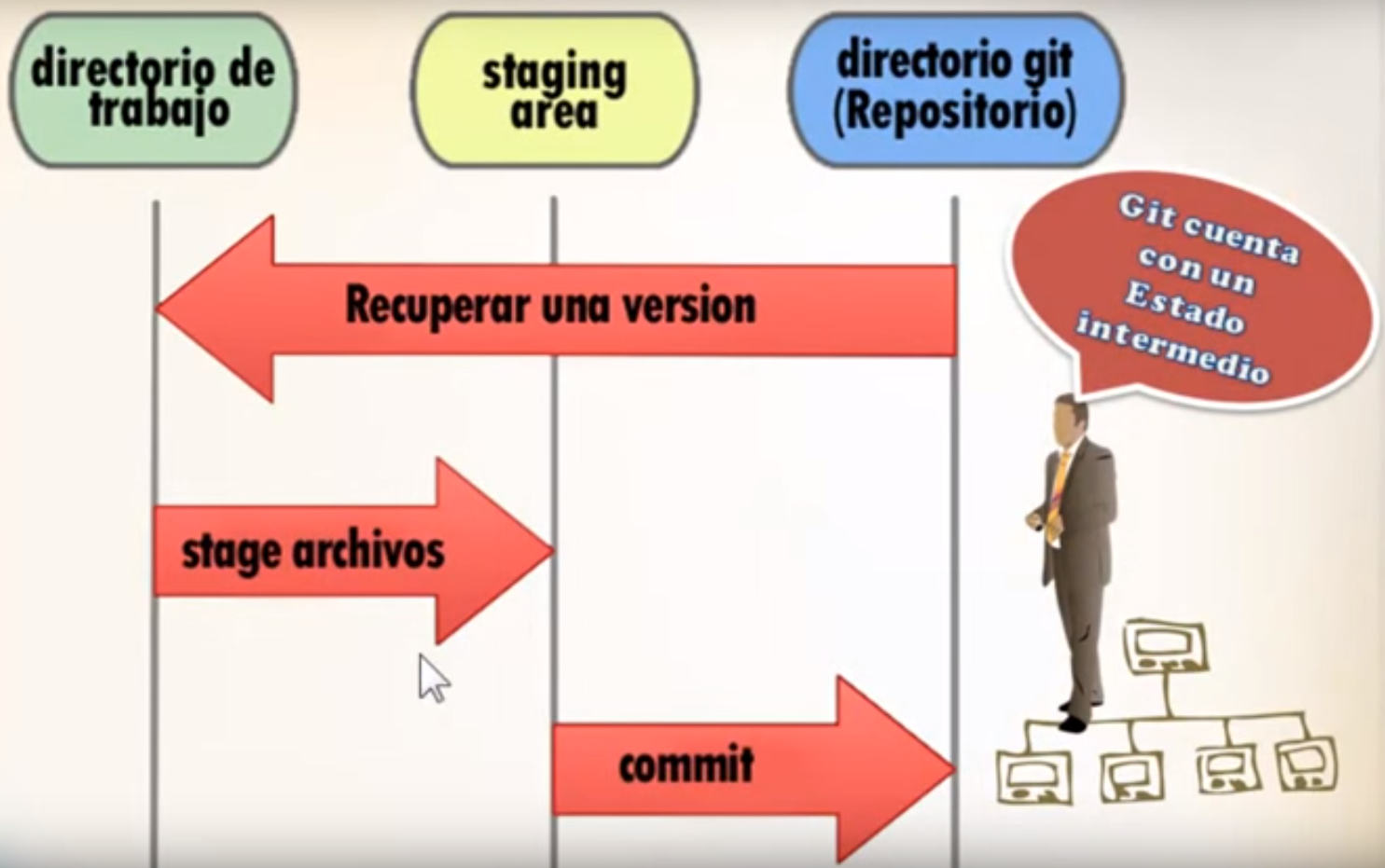
**LOS TRES ESTADOS DE GIT:**

**Commited:** Los datos están salvados en modo seguro en nuestra base de datos en local

**Modified:** Los datos están cambiados pero no los hemos salvado, por lo que los datos podrían perderse.

**Staging area:** Es como la terminal de embarque de un aeropuerto, en este estado podremos ir colocando archivos que queremos que se envien en el próximo commit, es como un estado intermedio entre commited y modified.

Esta área intermedia es un único archivo que almacena información de lo que irá en el próximo commit.



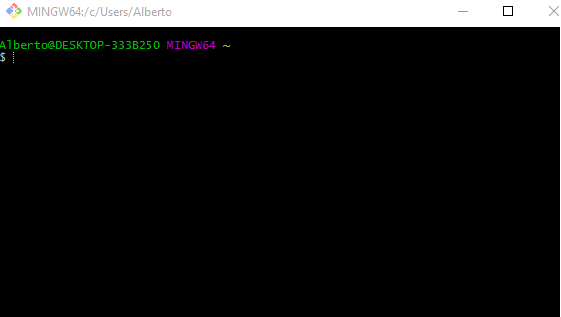
Si una versión particular se encuentra en el directorio git se dice que está commited.  
Si está modificado y ha sido añadido al área de Staging se dice que esta Staged  
Si todavía no ha sido añadido al área de Staging, entonces está modified.

**INSTALACIÓN DE GIT:**

<http://Git-scm.com>

Se puede descargar el programa desde la url superior, y se van siguiendo los pasos de instalación del programa.

Una vez hemos instalado Git, utilizaremos el Bush de Git, para ello lo ejecutaremos, y se nos abrirá un tipo consola de windows.



Lo primero que tenemos que hacer es instalar Github, para ello tenemos que hacernos con un SSH, o comprobar si ya tenemos una clave.

**Comprobar si ya tenemos un ssh:**

Escribimos cd ~/.ssh

Si ya tenemos una clave haremos un backup con el siguiente comando ls

Una vez realizado el backup, podremos crear un nuevo directorio con mkdir key\_backup

Una vez realizado directorio, haremos lo mismo con el contenido con cp id\_rsa+ key\_backup

**Crear un un nuevo ssh:**

Escribimos ssh-keygen -t rsa -C “alberto@adominguez.es”

Nos pedirá que insertemos una clave, la insertaremos dos veces, cuando la escribamos, aunque parezca que no se está escribiendo nada si que se está escribiendo, por lo que hay que tener cuidado con la contraseña

**CREACIÓN DE UNA CUENTA EN GITHUB:**

GitHub ofrece servicio en la nube, servicio de hosting, funcionalidades de redes sociales, sistema de Wikis, además de ofrecer estadísticas de los proyectos.

Ahora pasaremos a crear una cuenta en GitHub, para ello, accederemos a github.com, una vez creada la cuenta, vamos a introducir la clave SSH que hemos creado en el anterior paso.

Para ello iremos a la siguiente dirección:

<https://github.com/settings/keys>

y pulsaremos sobre New SSH key  


Para saber lo que tenemos que poner, iremos donde se nos ha creado la clave anteriormente, por defecto estará en la carpeta de usuario de nuestro ordenador, por ejemplo:  
C:\Users\Alberto\.ssh

Abriremos el archivo id\_rsa.pub con block de notas o cualquier editor.

Copiamos todo el contenido del archivo y lo pegamos en el navegador la parte de Key, le damos un título a la clave para poder acceder a ella.

Para comprobar que lo hemos hecho bien, volvemos a la consola y tecleamos

ssh [git@github.com](mailto:git@github.com)

Introducimos la contraseña que pusimos al inicio, y nos dirá que no tenemos acceso a github.com, sin embargo no hay problema, nos hemos autentificado, y vamos por el buen camino.

Ahora hay que actualizar la información personal en Git, ya que necesita un nombre de usuario y un email del responsable, para ello escribiremos:

git config - -global user.name “Alberto Dominguez”  
git config - -global user.email “alberto@adominguez”

Tenemos que configurar el Token de la API de Github, para ello iremos a la siguiente dirección:

<https://github.com/settings/tokens>

Y pulsamos sobre generate a personal access token, completamos los permisos que vamos a tener y ya tenemos nuestro Token creado.

Volvemos de nuevo a la consola y ponemos:

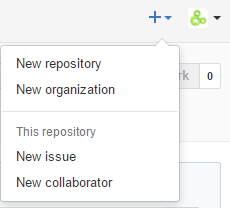
Git config --global github.token Escribir los caracteres del Token

IMPORTANTE: NO HAY ESPACIO ENTRE LAS BARRAS QUE HAY DELANTE DE GLOBAL

Con esto ya tendríamos configurado tanto GIT como GITHUB.

**Crear un repositorio en Github:**

Podemos crear un repositorio directamente desde GitHub



Y podemos crear una nueva carpeta en nuestro repositorio desde consola, y crear un archivo nuevo en la carpeta:

mkdir tutorial-git 🡪 Creamos la carpeta  
cd tutorial-git 🡪 Accedemos a la carpeta

echo "# tutorial-git" >> README.md

git init 🡪 Iniciamos Git

git add README.md 🡪 Añadimos el archivo README.md

git commit -m "first commit" 🡪 Realizamos el primer commit

git remote add origin git@github.com:ADominguezes/tutorial-git.git

git push -u origin master 🡪 Hacemos el Push

**Repositorios en GIT:**

Para crear un repositorio en nuestro ordenador, hay dos formas de poder hacerlo, desde la consola, o directamente desde el explorador de carpetas.

Para hacerlo desde la consola, abriremos git bash, y escribiremos

mkdir nombreCarpeta 🡪 Creamos la carpeta  
cd nombreCarpeta 🡪 Accedemos a la carpeta  
git init 🡪 Inicializamos el repositorio

Esto nos creará una carpeta oculta llamada .init que contiene información.

Para añadir todos los archivos utilizaremos el comando add de esta forma:

git add . 🡪 Añadimos todos los archivos para ser commiteados  
git add \*.html 🡪 Añadiremos todos los archivos con extensión html  
$ git commit -m "primera versión del proyecto" 🡪 Realizamos un commit del proyecto

También podemos utilizar un repositorio ya existente y clonarlo, por ejemplo si vamos a empezar un proyecto ya existente y necesitamos tener todos los archivos del proyecto. Para esto utilizaremos el comando $git clone y utilizando una dirección o url donde poder buscarlo.

git clone git://github.com/schacon/grit.git

Podemos utilizar el comando git status para conocer el estado de cada uno de los archivos

git status

## DICCIONARIO GIT

**Commit:** Convertir algo que es temporal en permanente, sería el acto de guardar, en el caso de GIT, sería crear una panorámica más.

**Modified:** Los datos están cambiados pero no los hemos salvado, por lo que los datos podrían perderse.

**Staging area:** Es como la terminal de embarque de un aeropuerto, en este estado podremos ir colocando archivos que queremos que se envien en el próximo commit, es como un estado intermedio entre commited y modified.

**Tracked:** Archivos que en algún momento han sido enviados en un commit

**Untracked:** Archivos que nunca se han enviado en un commit, por ejemplo archivos nuevos que creemos estarán en este estado.

**Unmodified:** Archivos que todavía no se han modificado.