Uso de Git en para control de Versiones en Automatización Industrial

Software Libre y Compromiso Social

Rafael Ernesto Escobar Zurita – i52eszur@uco.es

Indice:

1.- **Introducción.**

La informática tiene en la actualidad una gran influencia en la industria. El uso de aplicaciones informáticas permite mejorara el funcionamiento y el rendimiento de las actividades industriales. Incentivar a dichas empresas al uso de Software Libre es un reto que hoy en día hay.

En ramas como la Automatización Industrial mas aun si cabe, ya que todo el Software que se emplea es privativo. Donde empresas como Siemens que es una de las grandes en este sector con entornos para sus productos como Simatic 7 y Tia Portal, acaparan gran parte del mercado.

Como ex alumno, al haber acabado el Grado Superior de Automatismo y Robótica Industrial y estar cursando esta asignatura, he visto el gran potencial que puede tener usar herramientas de software libre, no solo en el transcurso de los dos años que el curso posee, si no aplicarlo en un futuro en la vida laboral.

Para ello es la razón de este trabajo, ya que en dicho ciclo en los dos años hay asignaturas de informática, y veo importante introducir en el Software Libre a los alumnos de primer curso para que puedan usar dichas herramientas.

Durante este trabajo vamos a describir el Software Libre así como a Richard Stallman y Linus Torvalds. Introduciremos Linux con sus distribuciones mas importantes así como los escritorios mas usados. Listaremos las aplicaciones mas importantes para la sustitución de software comercial. Terminando con la explicación de Git en un ejemplo real de una maqueta que se tiene que programar durante el ciclo.

2.- **Software libre. Richard Stallman y Linus Torvalds.**

El término Software Libre refiere el conjunto de software (programa informático) que por elección manifiesta de su autor cumple las siguientes libertades concebido por Richard Stallman:

1. Libertad para ejecutar el programa en cualquier sitio, con cualquier propósito y para siempre.
2. Libertad para estudiarlo y adaptarlo a nuestras necesidades. Exigiendo acceso al código fuente.
3. Libertad de redistribución, permitiendo colaboran con otras personas.
4. Libertad para mejorar el programa y publicar mejoras. Exigiendo el código fuente.

Un programa informático es software libre si otorga a los usuarios todas estas libertades de manera adecuada. De lo contrario no es Libre.

Licencias, es un mecanismos para garantizar, de acuerdo con la legalidad vigente lo siguiente:

- Las 4 libertades del SL.
- Permisos para que el receptor del programa pueda ejercer esas libertades.
- Restricciones
  - Dar crédito a los autores originales en caso de redistribución.
  - No pueden ir en contra de las 4 libertades del SL.

El termino Software Libre no significa que sea gratuito, ya que siendo software libre puede ser distribuido comercialmente o cobrarlo a precio de coste, con. Podemos obtener beneficios de las siguientes formas:

- Dar una cierta garantía.
- Servicio técnico.
- Personalización del software.

Tampoco debe confundirse con **Software de dominio publico**, el autor renuncia a todos sus derechos, pero tiene que estar declarado explícitamente en el programa, si no seria propietario. Las fuentes deben ser proporcionadas.

Otro termino relacionado es **Copyleft**, Software libre cuya licencia obliga a que las modificaciones que se distribuyan sean también libres, que posean como mínimo las mismas libertades que el software del que deriva.

Open source software, programas de fuente abierta, no es ni mas ni menos que Software Libre pero este se centra mas en la calidad del software, mientras que Software Libre hace referencia a las libertades.

Richard Stallman.

Richard Matthew Stallman nacido en Manhattan, Nueva York, 16 de marzo de 1953, con frecuencia abreviado como rms, programador estadounidense y fundador del movimiento por el software libre en el mundo. https://www.fsf.org/es.

Es el creador del editor de texto GNU Emacs, el compilador GCC y el depurador GBD bajo el proyecto GNU.

Enlaces referentes a Stallman.

https://es.wikipedia.org/wiki/Richard_Stallman

Conferencias en la Universidad de Córdoba.

http://www.webislam.com/videos/57218-charla_de_richard_stallman_en_la_universidad_de_cordoba.html

http://www.youtube.com/watch?v=zPt7LW8uk4I

http://www.ustream.tv/channel/stallman2010-cordoba

Linus Torvalds

Linus Bendedict Torvalds nacido el 28 de diciembre de 1969 en Helsinki, Finlandia., ingeniero de software conocido por iniciar y mantener el desarrollo del kernel de Linux, basándose en el sistema operativo libre Minix creado por Andrew S. Tanenbaum y herramientas y los compiladores del proyecto GNU.

Enlaces relacionados con Linus Torvalds

https://es.wikipedia.org/wiki/Linus_Torvalds

3.- **Linux. Distribuciones y escritorios.**

Linux termino mal empleado por muchas personas ya que cuando hacemos referencia a Linux solo hacemos referencia al núcleo y no a la distribución completa. Para hacer referencia a la distribución completa deberíamos hacerlo con GNU Linux.
Ya que GNU Linux, esta compuesto por el kernel Linux y un conjunto de programas como compiladores, entorno de escritorio, etc.

El Proyecto GNU, es un proyecto colaborativo de software libre con el objetivo de crear un sistema operativo completamente libre. Enlace web:

https://es.wikipedia.org/wiki/Proyecto_GNU

Las distribuciones que vamos a nombra a continuación se pueden descargar sus ISO desde la siguiente pagina web:

http://distrowatch.com/

En este punto vamos a definir como distribución padre la cual no deriva de otra:
- Debian.
- Suse.
- Red Hat.
- Fedora.
- Arch.

Después existen las que derivan de estas, las mas populares son Ubuntu que deriva de Debian y Linux Mint que deriva de Ubuntu.

Ademas tenemos un gran numero de distribuciones, en la pagina Distrowatch podemos hacer multitud de búsquedas dependiendo de la distribución Linux que busquemos acuerdo con nuestras necesidades. Por ejemplo:

- Para equipos con pocos recursos. https://www.azulweb.net/distribuciones-linux-ligeras-computadoras-viejas/
- Para diseño gráfico.
- Para pentesting.

En el siguiente enlace podemos ver una lista de distribuciones linux totalmente libres:
https://www.gnu.org/distros/free-distros.es.html
Podemos destacar Triskel al ser Gallega.

Unas de las diferencias con Windows o Mac OsX es que podemos customizar todo incluido el escritorio, o el entorno del usuario ya que en Gnu Linux disponemos de variedad de entornos de escritorio como por ejemplo:

- Kde/Plasma https://www.kde.org/plasma-desktop
- Gnome https://www.gnome.org/
- Unity https://unity.ubuntu.com/
- LXDE http://lxde.org/
- Xcfe https://www.xfce.org/

4.- **Aplicaciones de software Libre.**

Vamos ha emparejar aplicaciones comerciales con aplicaciones de Software Libre, así damos a conocer muchas de estas aplicaciones que para el usuario normal le son poco conocidas. En la siguiente tabla vamos a ver dichos programas.

Categoría	Software Comercial	Software Libre
Oficina.	Microsoft Office	LibreOffice, Latex.
	Microsoft Visio	Dia.
Imágenes.	Photoshop	Gimp.
	Adobe Illustrator, Macromedia Freehand, CorelDraw	Inkskape
Grabación CD/DVD	Nero	Brasero, IsoMaker
Multimedia	Maya, 3dsMax	Blender
	Windows Media Player, Power DVD	VLC.
	Adobe Premiere,	OpenShot
Navegador Web	Internet Explorer	Chromiun, Firefox
Pdf	Acrobat Reader	Sumatra Pdf.
Gestor de Correo	Outlook	Thunderbird
Compresores	Winrar	7zip

En la siguiente dirección podéis ver muchos mas software libre :

http://www.cdlibre.org/consultar/catalogo/index.html

5.- **Git.**

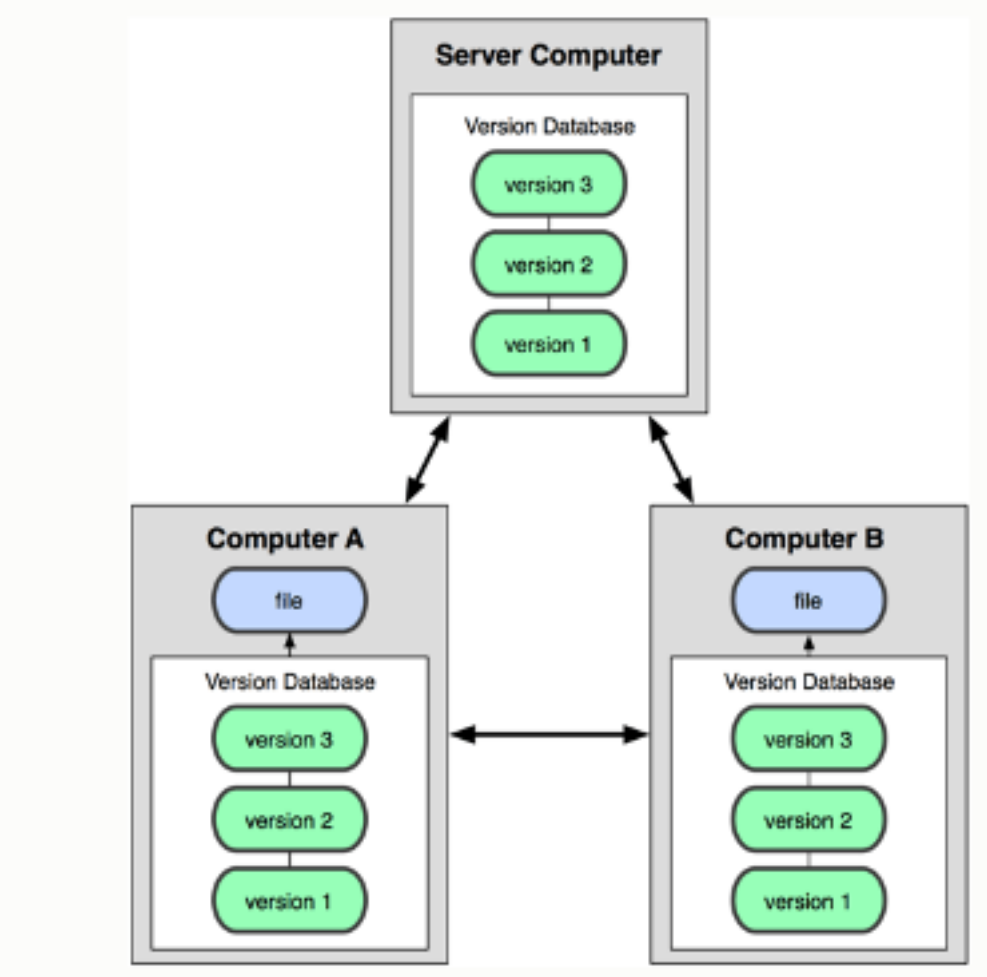
5.1.- **Historia.**

Git es un sistema de versiones distribuido. Linus Torvalds utilizaba un sistema de versiones propietario llamado Bitkeeper. Pero en 2005 diseña Git ante la ruptura de relaciones de la comunidad de Linux y Bitkeeper.

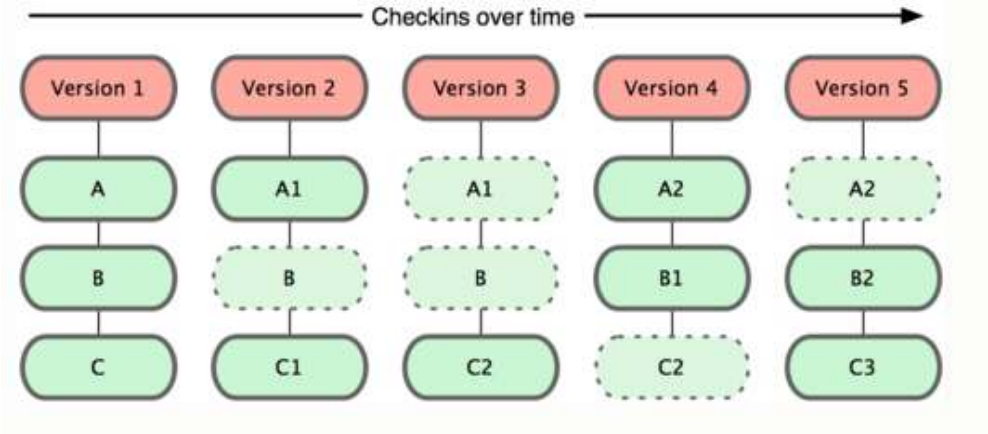
Se ideo con las siguientes características:

- Velocidad
- Diseño sencillo
- Desarrollo no lineal. Muchas ramas paralelas.
- Distribuido.
- Integrar grandes proyectos.
- Fácil de usar.

Las ventajas de los sistemas distribuidos es que tanto en el servidor como en los clientes existe una copia de los datos. Esto tiene la ventaja que si hay un problema de comunicación con el servidor no dejamos de funcionar al tener una copia en local, ademas si el servidor se rompe y se pierde la información podemos restaurarla desde las copias locales de los clientes, como puede apreciarse en la siguiente figura.



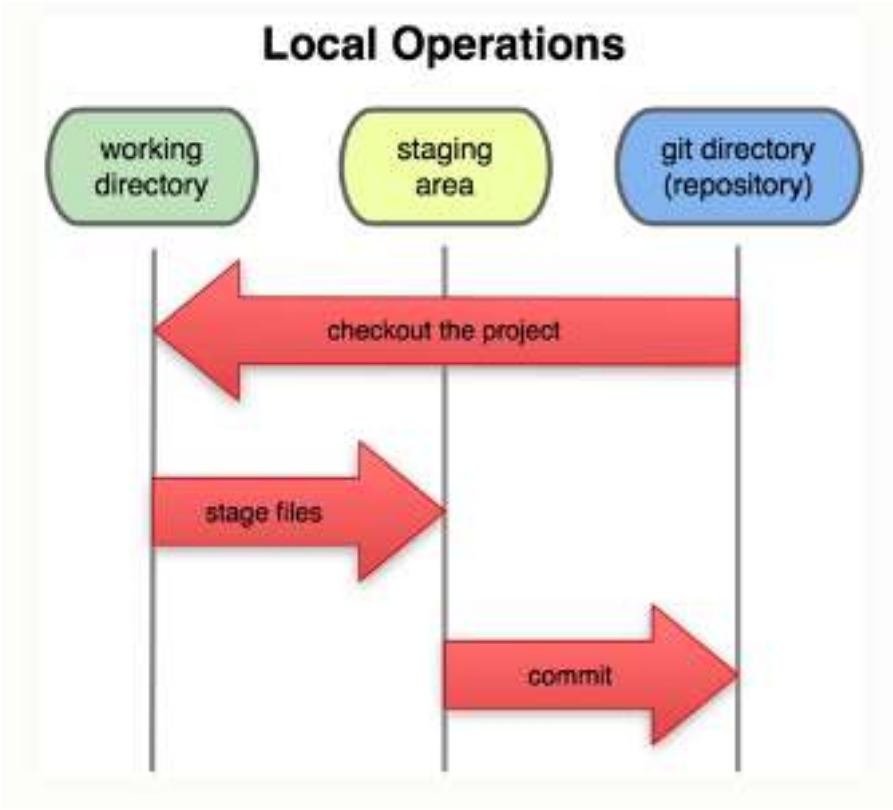
Git modela sus datos como un conjunto de instantáneas de un mini sistema de archivos. Cada vez que confirmas un cambio, o guardas el estado de tu proyecto en Git, él básicamente hace una foto del aspecto de todos tus archivos en ese momento, y guarda una referencia a esa instantánea. Para ser eficiente, si los archivos no se han modificado, Git no almacena el archivo de nuevo, sólo un enlace al archivo anterior idéntico que ya tiene almacenado. Git modela sus datos más como se aprecia en la siguiente figura.



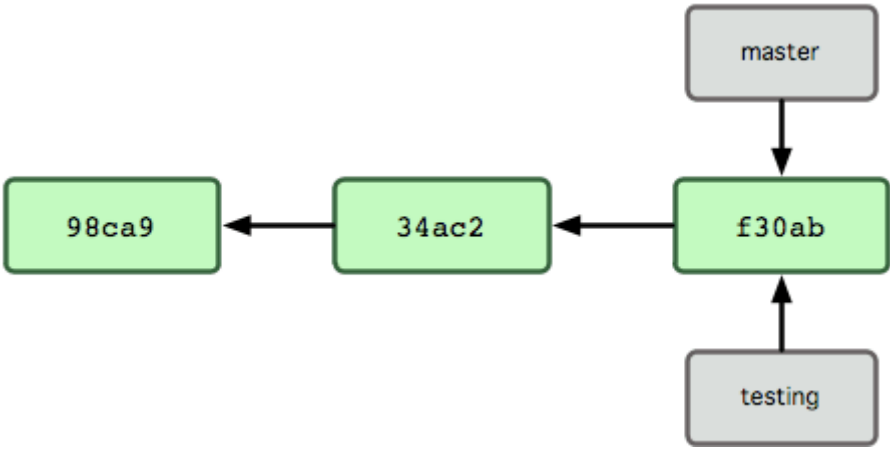
La mayoría de las operaciones en Git sólo necesitan archivos y recursos locales para operar. Por lo general no se necesita información de ningún otro ordenador de tu red. De esta forma no sobrecarga la red.

Cuando trabajamos con Git los ficheros están en tres estados:

- Modificado (Modified), fichero que se ha modificado en el directorio de trabajo.
- Preparado (Staged), archivos guardados y preparados para el próximo commit.
- Confirmado (Committed) Los datos se almacenan en el directorio de Git (.git), donde Git almacena los metadatos y la base de datos de objetos para tu proyecto.



Uso de ramas en Git. Por defecto se crea la rama **master** donde se van almacenando los commit. Al crear una rama crea un nuevo apuntador a la misma confirmación donde estés actualmente.



Instalación de Git:

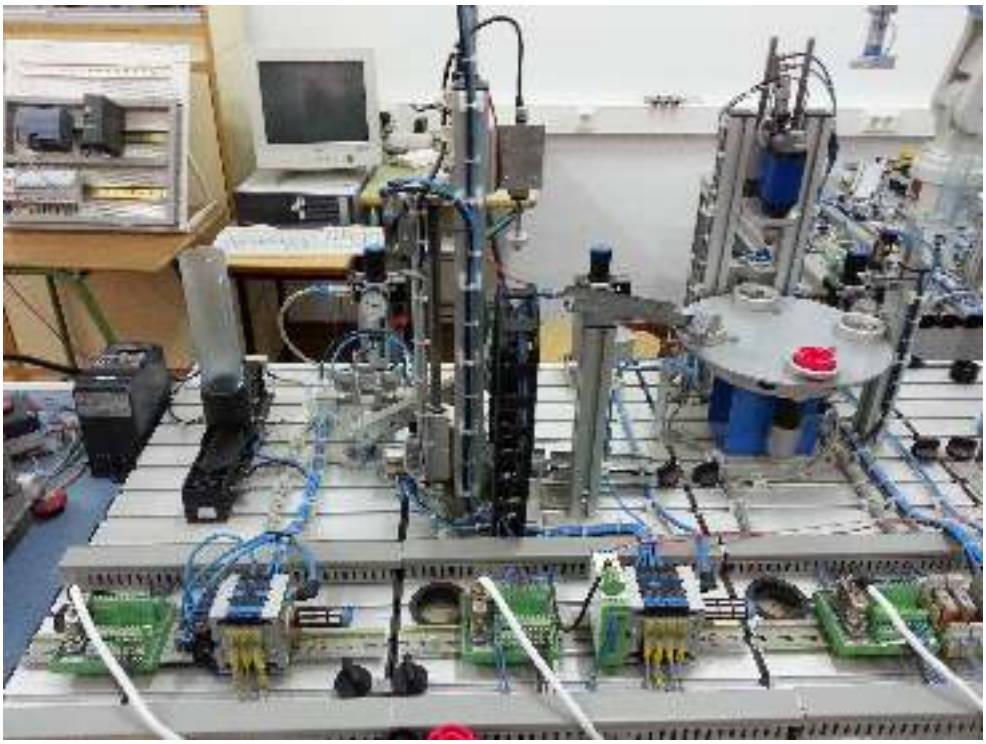
- **Linux**
  Si es una Debian apt-get install git
  Si es una Fedora o Suse yum install git-core

- **Mac**
  descargar en: http://code.google.com/p/git-osx-installer
  Si tienes MacPorts instalado (http://www.macports.org).
  $ sudo port install git-core +svn +doc +bash_completion +gitweb

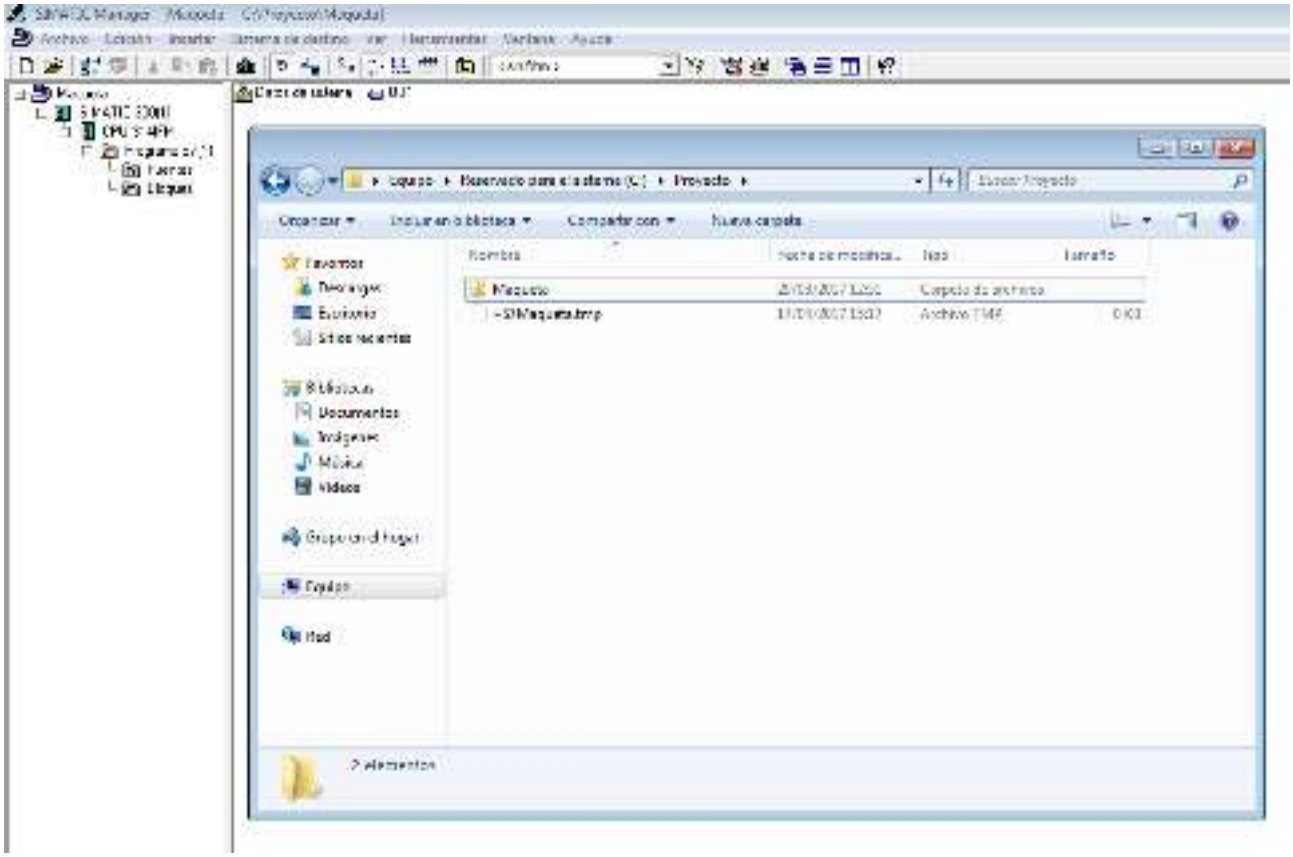
- **Windows**
  Descarga el archivo exe del instalador desde la página de GitHub, y ejecútalo: http://msysgit.github.com/

5.2.- **Uso en una maqueta del colegio.**

Para explicar como usar Git vamos a usar la maqueta de las 6 estaciones donde lo vamos a explicar usando las 2 primeras estaciones, mostrada en la siguiente imagen:



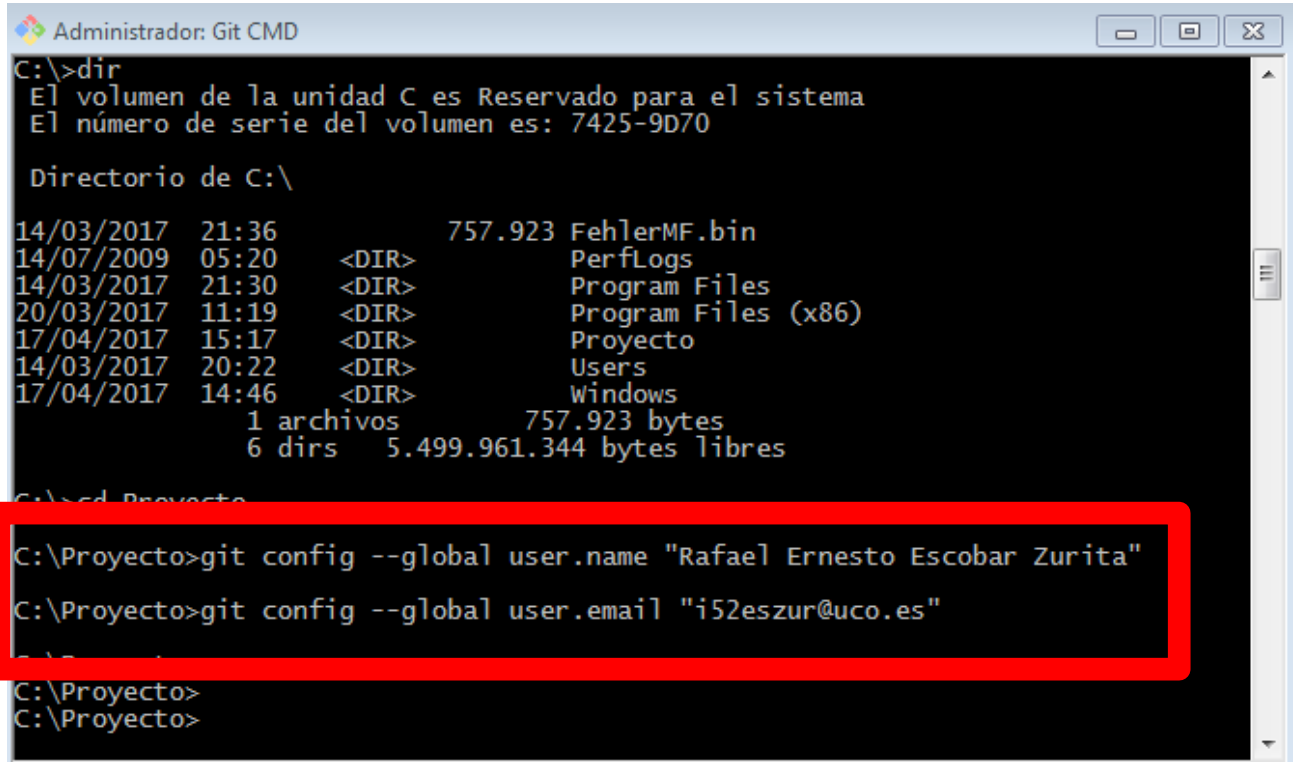
Comenzamos creando un proyecto en Simatic 7. Cuando lo tenemos configurado vamos a comenzar con Git.



La ruta de nuestro proyecto va hacer **c:\proyecto\maqueta**.
Lo primero es configurar la identidad nuestra, nuestro nombre y nuestra dirección de correo electrónico con las siguientes instrucciones:

git config –global user.name “Rafael Ernesto Escobar Zurita”
git config –global user.mail “i52eszur@uco.es”

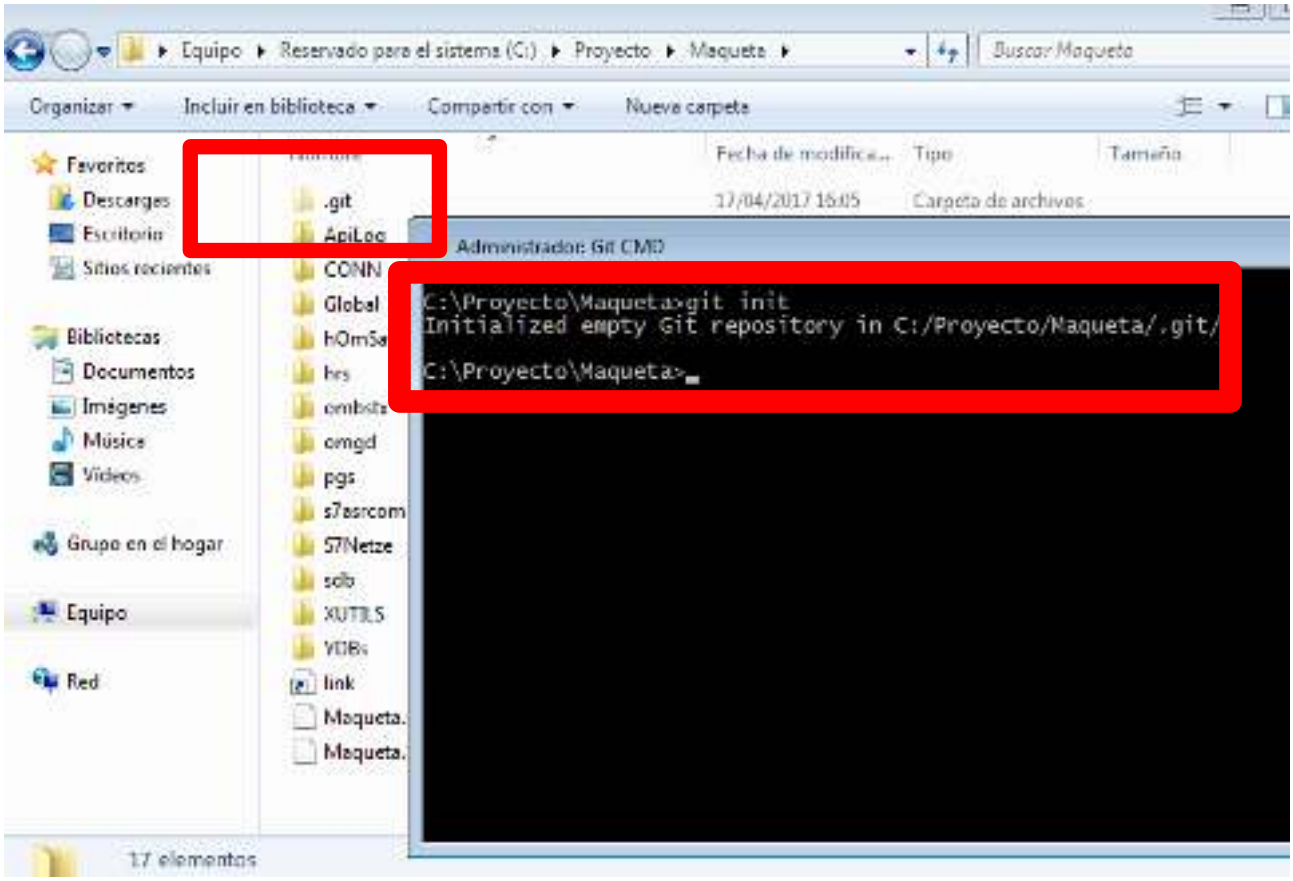
Como se ve en la siguiente captura:



Una vez configurado esto nos vamos al directorio donde estamos almacenando nuestro proyecto e inicial izamos el repositorio Git.

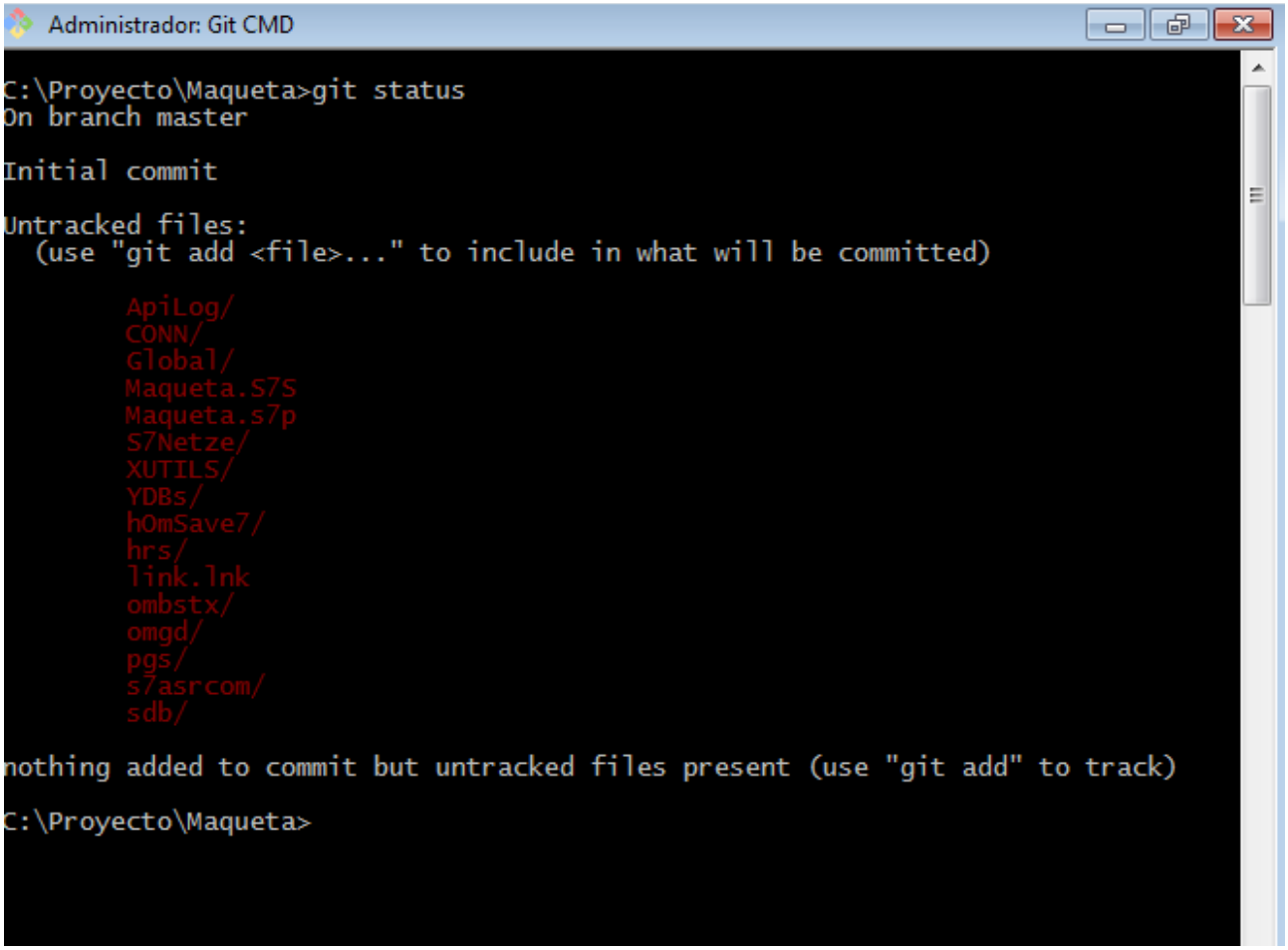
C:\Proyecto\maqueta\git init

En la siguiente captura vemos que se ha creado un nuevo repositorio y vemos como se ha creado automáticamente un directorio **.git**, normalmente esta oculto y es donde crea la base de datos de nuestro repositorio.



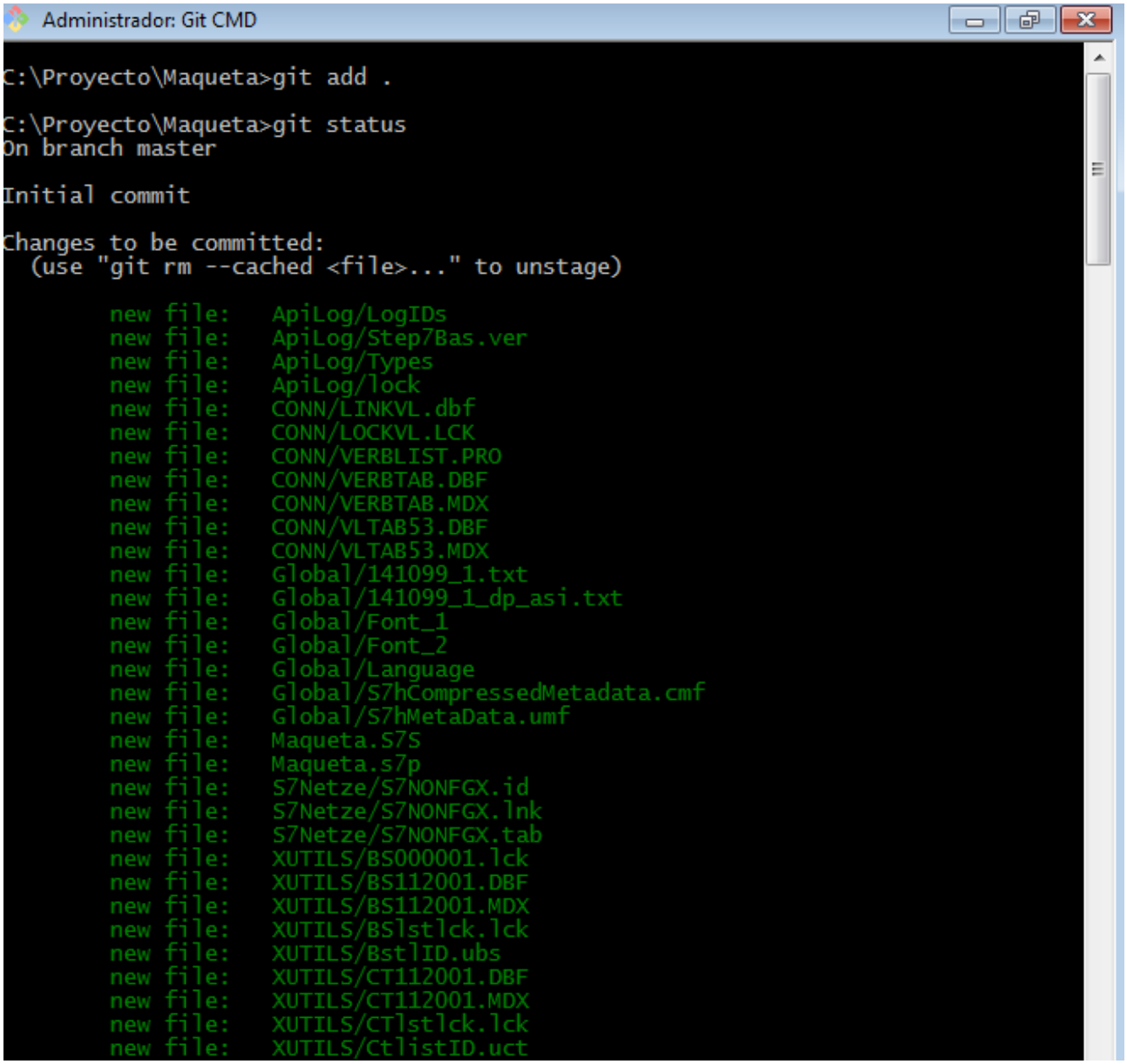
Una vez creado nuestro repositorio, esta preparado para ir añadiendo ficheros a nuestro repositorio. Ahora tendríamos que preparar nuestros ficheros y después hacer un commit para añadir los ficheros modificados al repositorio.

Si usamos la instrucción **git status**, vemos el estado de nuestro directorio de trabajo.

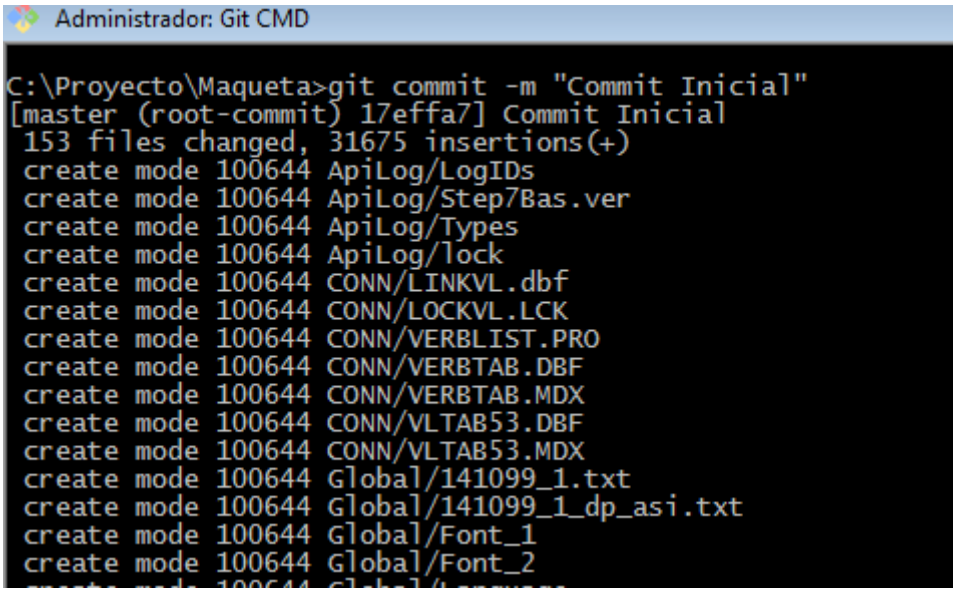


Como vemos todos los ficheros están untracked, ahora hay que prepararlos para realizar el Commit Inicial.

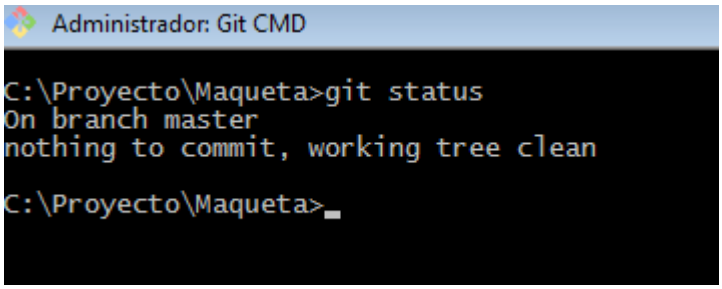
Preparamos los ficheros para el Commit, usamos **git add .** , para añadir todos los ficheros. Y vemos como los ficheros los hemos preparado para añadirlos.



Para realizar el commit **git commit -m “Commit Inicial”**.



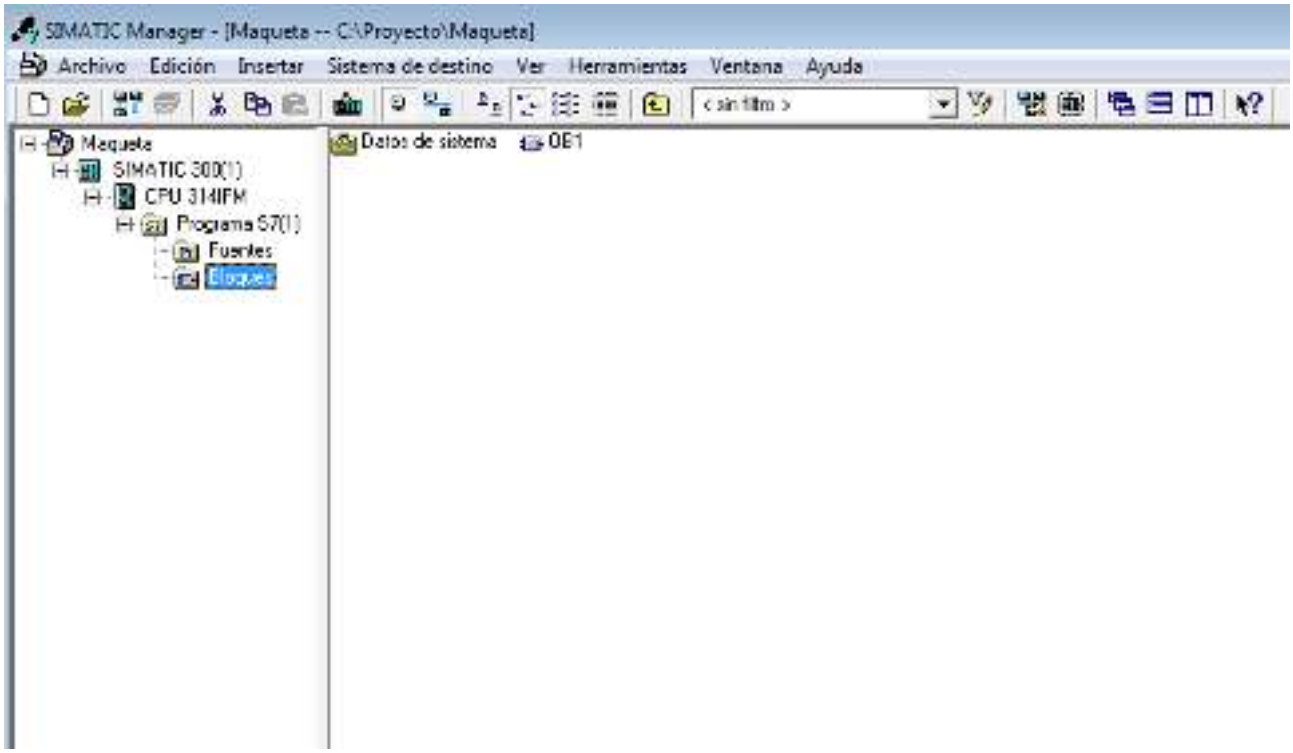
Si volvemos hacer un **git status** vemos que no tenemos nada para commit.



Cuando modificamos algo tendríamos los ficheros modificados untracked y volveríamos a prepararlos y hacer un nuevo commit.

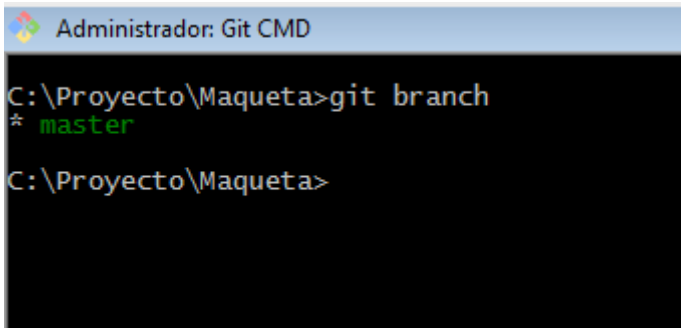
Podemos restaurar nuestros ficheros al ultimo commit pero esto restauraría todos los ficheros, con lo que lo mejor es el uso de las ramas.

Hemos terminado de programar nuestra primera estación, ahora vamos a comenzar con la segunda, no solo hay que hacer funcionar la segunda estación solamente sino que hay que hay que coordinar ambas estaciones. En dicho proceso podemos tener el problema de que lo que antes nos funcionaba correctamente nos deje de funcionar, para ello vamos a crearnos una rama donde vamos a comenzar a programar la segunda estación y después coordinarla con la primera.
Por defecto la primera rama que se crea al hacer un **git init** se denomina **master**, esta rama no puede cambiarse de nombre.



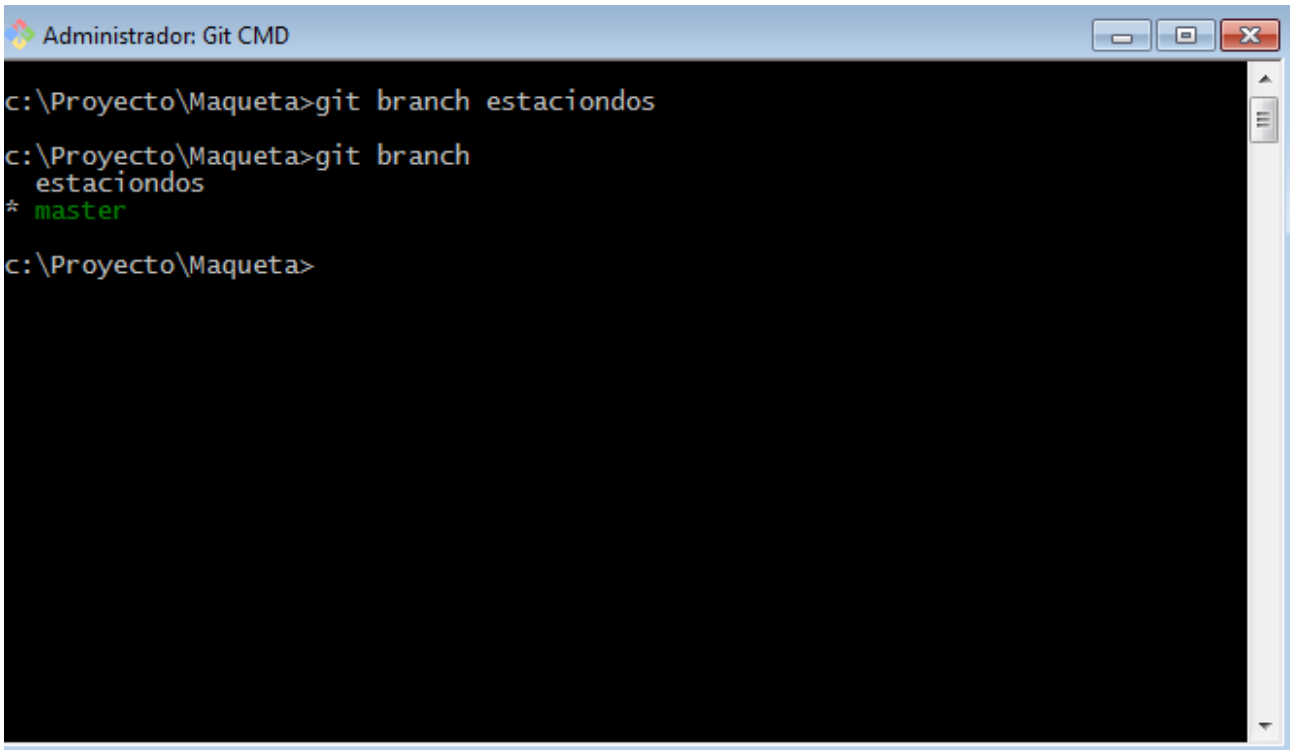
Nuestra rama master posee el siguiente contenido como vemos en la siguiente imagen. Donde OB1 almacenaría el código necesario para el funcionamiento de nuestra primera estación.

Para ver la rama activa usamos **git branch** y vemos como ahora mismo solo tenemos una rama y es la master la activa.



Ya tenemos nuestra primera estación completamente configurada a continuación vamos a crear una nueva rama, con la siguiente instrucción:
git branch segundarama.

Si hacemos git branch vemos como esta nuestra nueva rama creada ademas podemos ver la rama en la que nos encontramos, que aparece en verde, como podemos ver en la siguiente captura.

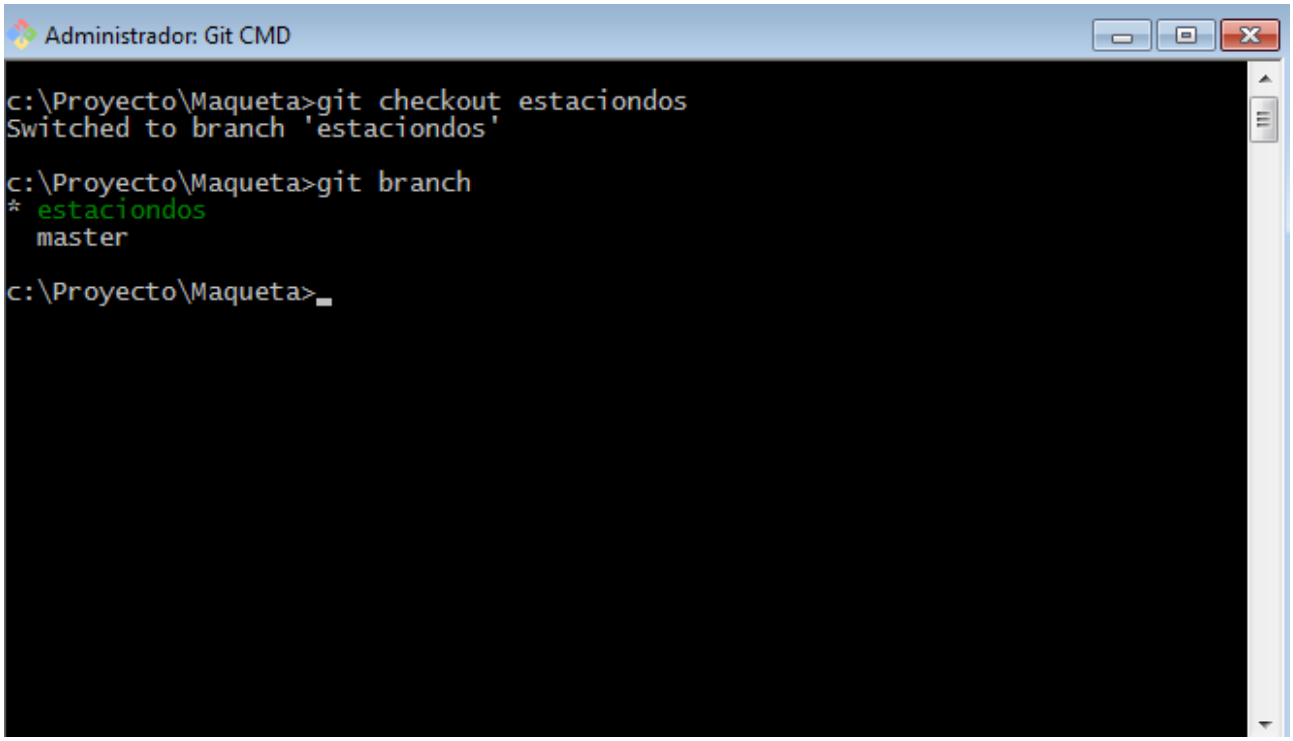


Esta rama es una copia exacta de la master, ahora podemos cambiarnos a estaciondos y a partir de ahí todos los cambios se realizaran es dicha rama. Quedando intacta la rama master y podemos volver cuando queramos a la rama master. Podemos tener cuantas ramas queramos.

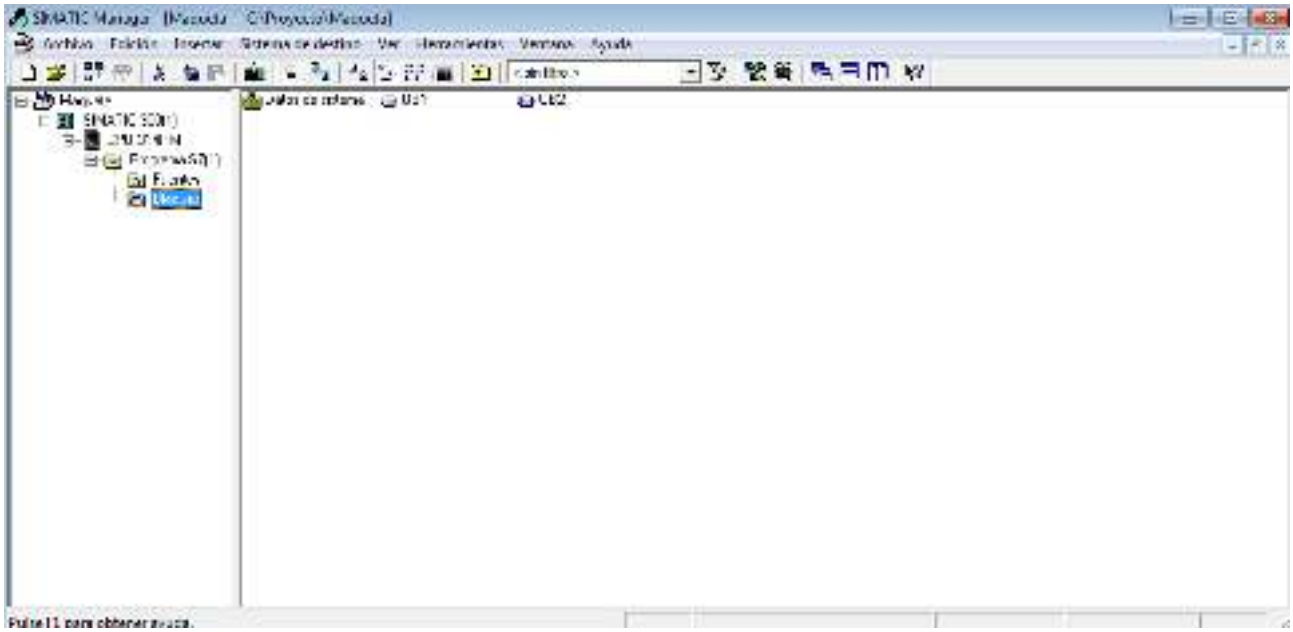
Para cambiarnos de rama usaremos:

git checkout estaciondos

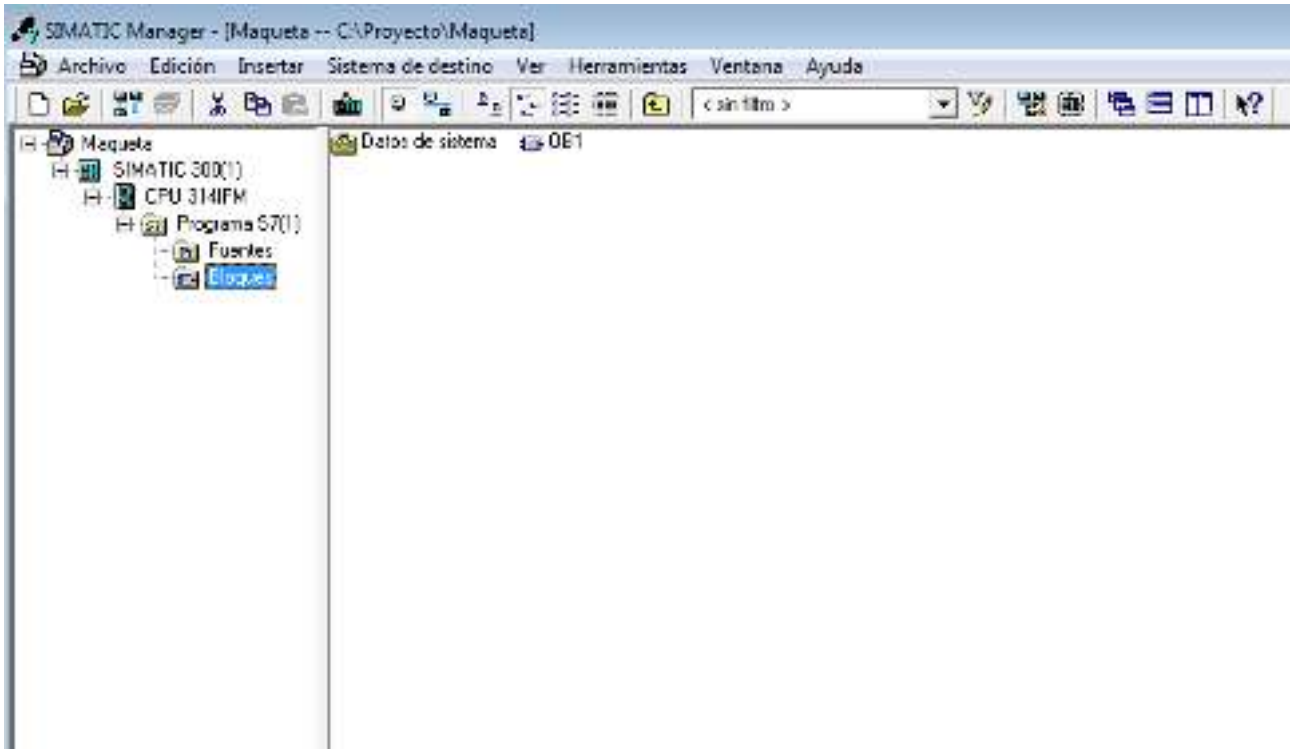
Vemos como la rama activa ahora es estaciondos, donde iremos haciendo todos los cambios necesarios manteniendo la rama master intacta, siempre pudiendo volver a ella si así nos hiciera falta.



Ahora añadimos la programación de la estación dos, y volvemos hacer los commit necesarios. Hemos añadido el OB2 donde tenemos la programación de la estacion.



Si volviésemos a la rama master con **git checkout master** podemos observar que volvemos a la versión de código donde solo tenemos el OB1

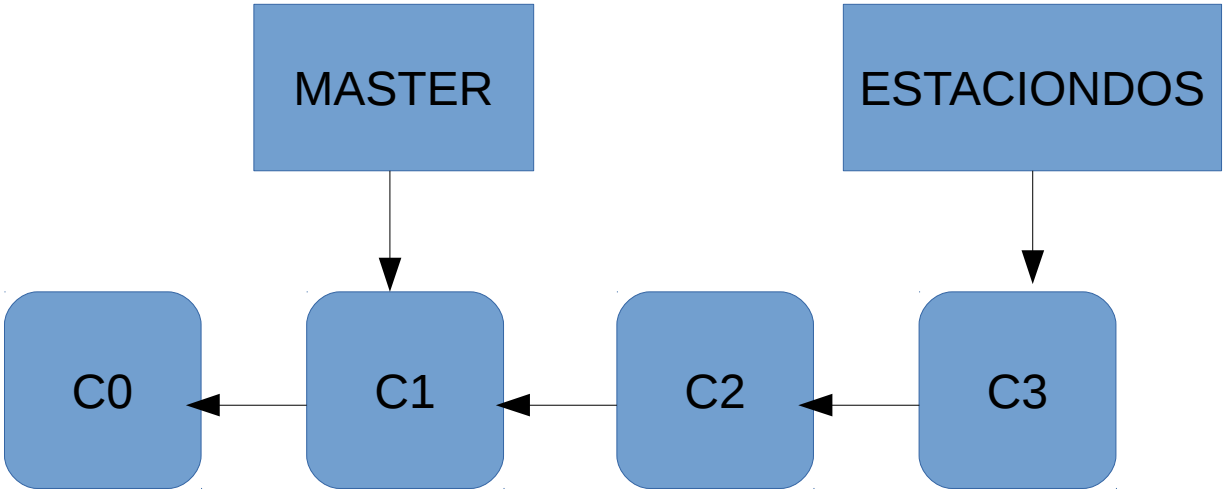


Podemos unificar ramas master con estaciondos seguimos el siguiente proceso:

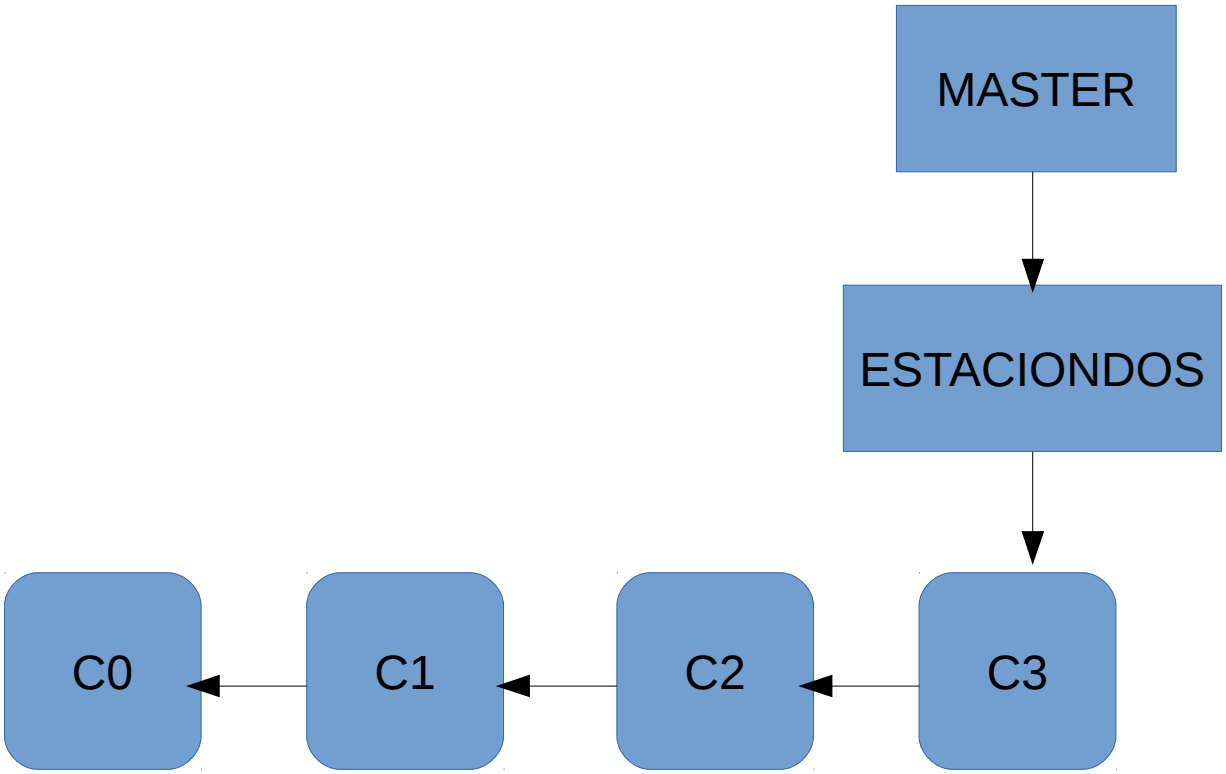
- Nos movemos a la rama master **git checkout master**.
- Unimos las ramas con **git merge estaciondos**.
- Borramos la rama estaciondos **git branch -d estaciondos**. Ya que tendríamos las dos ramas con el mismo contenido.

Con esto tenemos unificadas ambas ramas, también podemos crear una rama nueva desde estaciondos.

En las siguientes dos figuras vemos como se quedarían las ramas cuando las unimos.



Después de la unificación de las ramas.



5.3.- **Instrucciones mas importantes.**

Configuración del usuario y la dirección de correo, tu identidad.

git config –global user.name “Nombre”
git config –global user.mail “Dirección de correo”

Inicial izar Git.

git init

Ver estado del árbol de trabajo, el estado de nuestros ficheros.

git status

Añadir archivos

git add

Hacer commit

git commit -m “comentario”

Ver ramas que tenemos

git branch

Crear una rama.

git branch “nombre rama”

Unir ramas.

git merge “rama a unir”

Borrar una rama.

git branch -d “nombre rama”