

A lo largo de este curso vamos a utilizar el sistema de control de versiones de Git en la realización de todos los ejercicios prácticos, incluidos aquellos que se planteen en los exámenes. Para iniciarnos en el aprendizaje de este sistema de control de versiones, podemos encontrar en Internet documentación abundante, como por ejemplo el libro Pro Git. Este libro se puede consultar online o descargar en formato electrónico en varios idiomas, español incluido, desde <https://www.git-scm.com/book/en/v2>.

En lo que se refiere al presente documento, su propósito no es enseñar Git, sino únicamente describir varias formas de iniciar el control de versiones en un proyecto de Eclipse usando tanto repositorios locales como remotos.

Introducción a Git

Git es un software de control de versiones libre y opensource disponible para Linux, Mac y Windows. En el primer capítulo del libro Pro Git se explica cómo instalarlo en estos sistemas operativos. Si no deseas descargar el libro, puedes consultar este tema online en la dirección <https://git-scm.com/book/es/v2/Inicio---Sobre-el-Control-de-Versiones-Instalación-de-it>.

El aprendizaje de Git en su entorno nativo tendrá como objetivo conocer los comandos disponibles, su sintaxis y para qué se utilizan en el control de versiones.

No obstante, existen herramientas que facilitan el trabajo con Git a través de una interfaz gráfica de usuario. La versión de Git para Windows incorpora de serie dos herramientas de este tipo, **gitk** y **git-gui**, ambas con una funcionalidad muy limitada. Si se desea utilizar herramientas de este tipo con una funcionalidad más avanzada, existen en Internet algunas más completas, disponibles para los sistemas operativos de ordenador (Linux, Mac y Windows) y para los de dispositivos móviles (Android e IOS). Entre las más populares están **GitKraken**, disponible para Linux, Mac y Windows, o **SourceTree** disponible para Mac y Windows. Sin embargo, aunque estas aplicaciones facilitan mucho el uso de Git, la mejor forma de entender este sistema de control de versiones es realizar todo el proceso de aprendizaje desde la línea de comandos.

El control de versiones con Git gira en torno al concepto de repositorio. Para realizar el control de versiones de cualquier proyecto de programación se crea al menos un repositorio, que en la práctica se trata de una carpeta en la que se guardan:

- Los archivos y carpetas que forman parte del proyecto, entre los que probablemente algunos no estén sometidos al control de versiones. Esto es lo que en Git se conoce como el *working tree*.
- Archivos para uso interno de Git entre los que está la base de datos en la que se guardan los cambios que sufren los ficheros sometidos a control de versiones cuando se realizan las confirmaciones. Normalmente estos archivos se encuentran en una carpeta con el nombre *.git* en la carpeta raíz del repositorio.

Git es un sistema de control de versiones distribuido, pensado para que varios desarrolladores puedan colaborar en un mismo proyecto de desarrollo de software. Cada desarrollador tendrá su propia copia del proyecto en un repositorio local y tendrá la posibilidad de compartir sus cambios con el resto de participantes enviándoselos a sus repositorios. Git permite llevar a cabo este tipo de sincronización de forma descentralizada, es decir, los desarrolladores pueden enviarse los cambios los unos a los otros sin la intervención de un servidor central. Sin embargo, en la práctica se instala Git en una máquina a modo de servidor y en él se crean repositorios denominados remotos, cuyo cometido será el de permitir una sincronización centralizada entre los repositorios de los desarrolladores, haciendo que ésta resulte más sencilla y fiable.

En cualquier caso, ya sea de forma descentralizada, o con un servidor de Git, la sincronización se lleva a cabo ejecutando una serie de comandos específicos que harán uso de los protocolos de comunicaciones se describen en <https://git-scm.com/book/es/v1/Git-en-un-servidor-Los-Protocolos>.

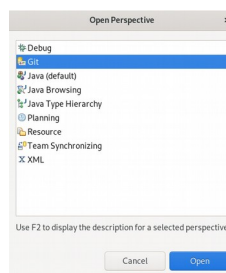
En la práctica no es necesario instalar un servidor de Git, ya que existen en Internet servidores públicos que proporcionan sus servicios a través de una aplicación Web que además suele proporcionar una serie de características adicionales como son, entre otras, la posibilidad de crear una [wiki](#) o una página web para cada proyecto o la posibilidad de usar recursos online que facilitan la colaboración. Entre los más populares están [GitHub](#), [GitLab](#) o [Bitbucket](#), que proporcionan un servicio básico gratuito y otros de pago con prestaciones superiores. Para poder utilizar estos servidores es necesario realizar un sencillo proceso de registro con el que obtenemos un nombre de usuario y una contraseña que usaremos para iniciar sesión y acceder a los servicios contratados, ya sean gratuitos o de pago.

Tanto Eclipse como IntelliJ Idea incorporan de serie los plugins necesarios para el trabajo con repositorios de Git locales y remotos, integrando el manejo de los mismos en la interfaz gráfica de usuario.

Creación de un proyecto de Eclipse en un repositorio local de Git

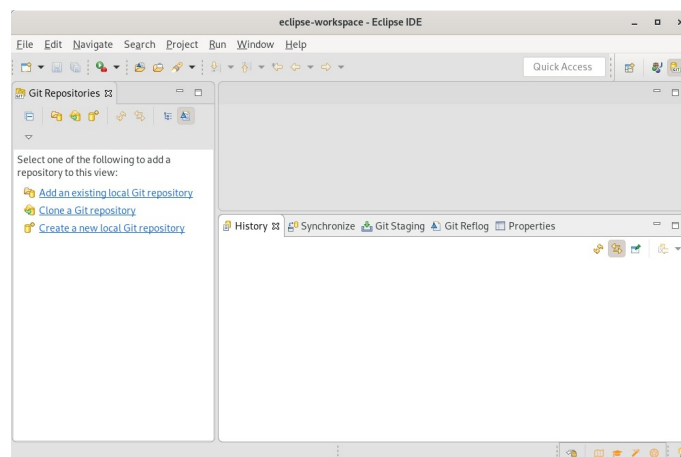
Existen varias formas de crear un proyecto de Eclipse dentro en un repositorio de Git, dependiendo de si el repositorio va a contener un único proyecto o varios, y de si se va sincronizar con un remoto o no.

En cualquier caso, resultará útil abrir la perspectiva de Git en Eclipse, aunque no es imprescindible. Esta se abre seleccionando en el menú principal la opción *Window* → *Perspective* → *Open Perspective* → *Other...*, que abre un cuadro de diálogo en el que hay que elegir la opción *Git* y pulsar el botón *Open* (el botón *Open Perspective* situado en la parte derecha de la barra de herramientas lo abre directamente):




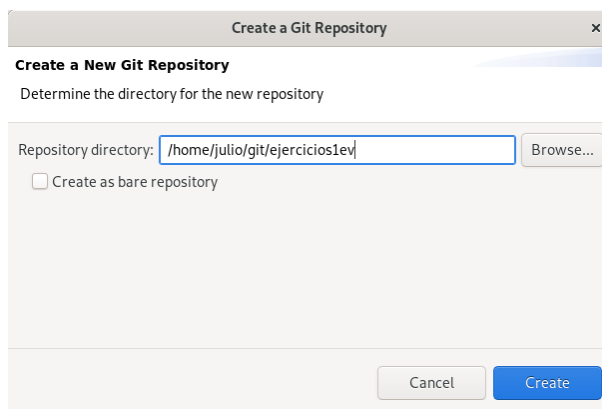
Cuando se abre por primera vez una perspectiva, se añade en la parte derecha de la barra de herramientas el correspondiente botón de acceso rápido (inicialmente sólo se encuentra disponible el botón de acceso rápido para la perspectiva de Java).

La perspectiva de Git contiene las vistas *Git Repositories*, *History*, *Synchronize*, *Git Staging*, *Git Reflog* y *Properties*, que proporcionan herramientas para la gestión de repositorios:



Antes de crear un proyecto nuevo de Eclipse dentro de un repositorio, se pueden dar dos situaciones:

- El repositorio ya esté creado, independientemente de si aparece en la vista *Git Repositories* o no. En este caso pasaremos directamente a la creación del proyecto.
- El repositorio no existe, lo creamos en el cuadro de diálogo que se abre pulsando sobre el botón *Create a new local Git repository*  de la de la vista *Git Repositories*:

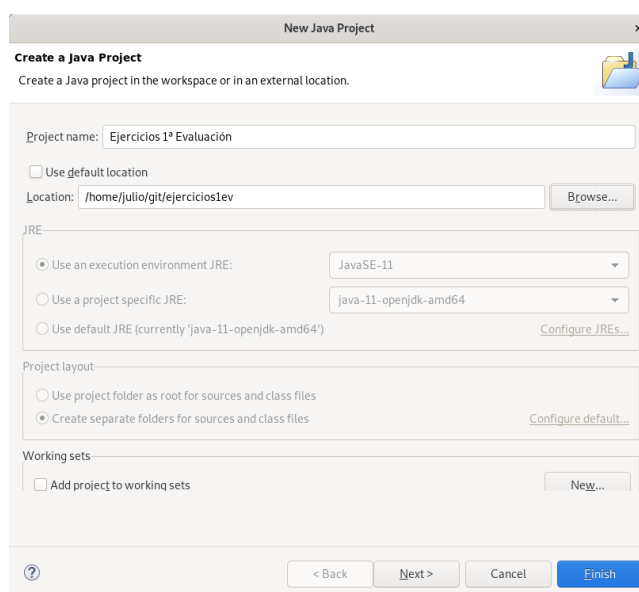


Seleccionamos una carpeta para el repositorio. Como se puede observar en la imagen, la nueva carpeta se sitúa por defecto dentro de otra llamada *git* que se encuentra, a su vez, en la carpeta personal del usuario que ha iniciado la sesión en el sistema operativo.

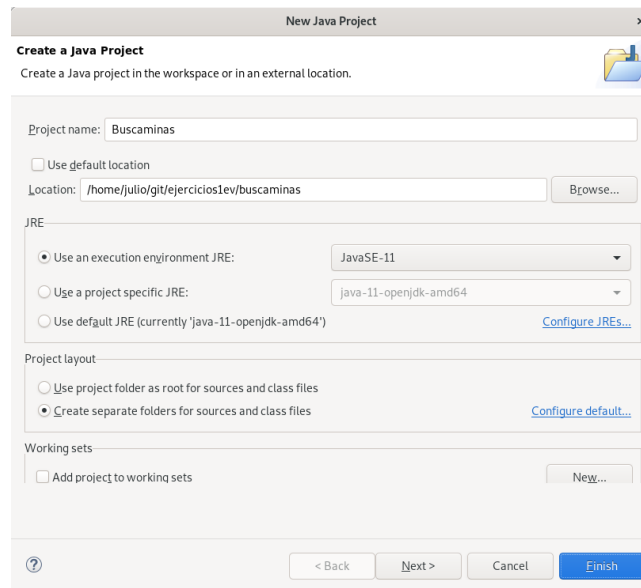
La opción *Create as bare repository* se utiliza normalmente para crear un repositorio que va a ser usado como remoto y, por tanto, normalmente no se marca.

El proceso finaliza al pulsar el botón *Create*.

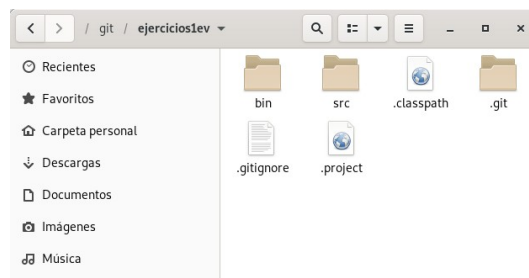
Una vez creado el repositorio, retornamos a la perspectiva de Java e iniciamos la creación de un proyecto nuevo. En el asistente de creación del proyecto asignamos un nombre al proyecto, desactivamos la casilla *Use default location* y en el cuadro *Location* utilizamos el botón *Browse* para seleccionar el repositorio que acabamos de crear como carpeta de proyecto. Si tenemos previsto que el repositorio contenga un único proyecto, elegimos como carpeta raíz del nuevo proyecto la carpeta del repositorio:



Si deseamos que el repositorio contenga varios proyectos, añadimos a la ruta especificada en el campo *Location* el nombre de una carpeta para el proyecto a continuación del nombre del repositorio:

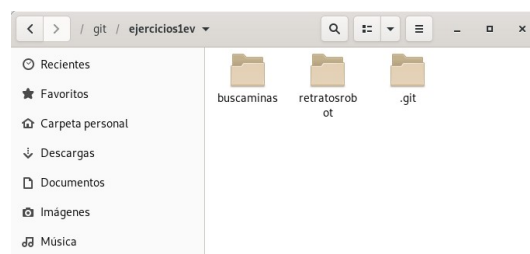


En cualquier caso, acabamos de crear un proyecto fuera del workspace configurado en Eclipse. Si echamos un vistazo con el explorador de archivos a la carpeta de un repositorio con un sólo proyecto veremos el contenido siguiente:



- La carpeta *.git* contiene la información que maneja Git internamente para llevar a cabo el control de versiones.
- El archivo *.gitignore* lo crea Eclipse de forma automática al detectar que hemos creado el proyecto en un repositorio. En la documentación de Git se explica el cometido de estos archivos.
- El resto de carpetas y archivos son los habituales en cualquier proyecto de Eclipse.

En el caso de que el repositorio contenga varios proyectos, veremos el contenido siguiente:



Sólo vemos la carpeta *.git* y las carpetas de los proyectos creados. Eclipse creará un archivo *.gitignore* dentro de cada carpeta de proyecto.

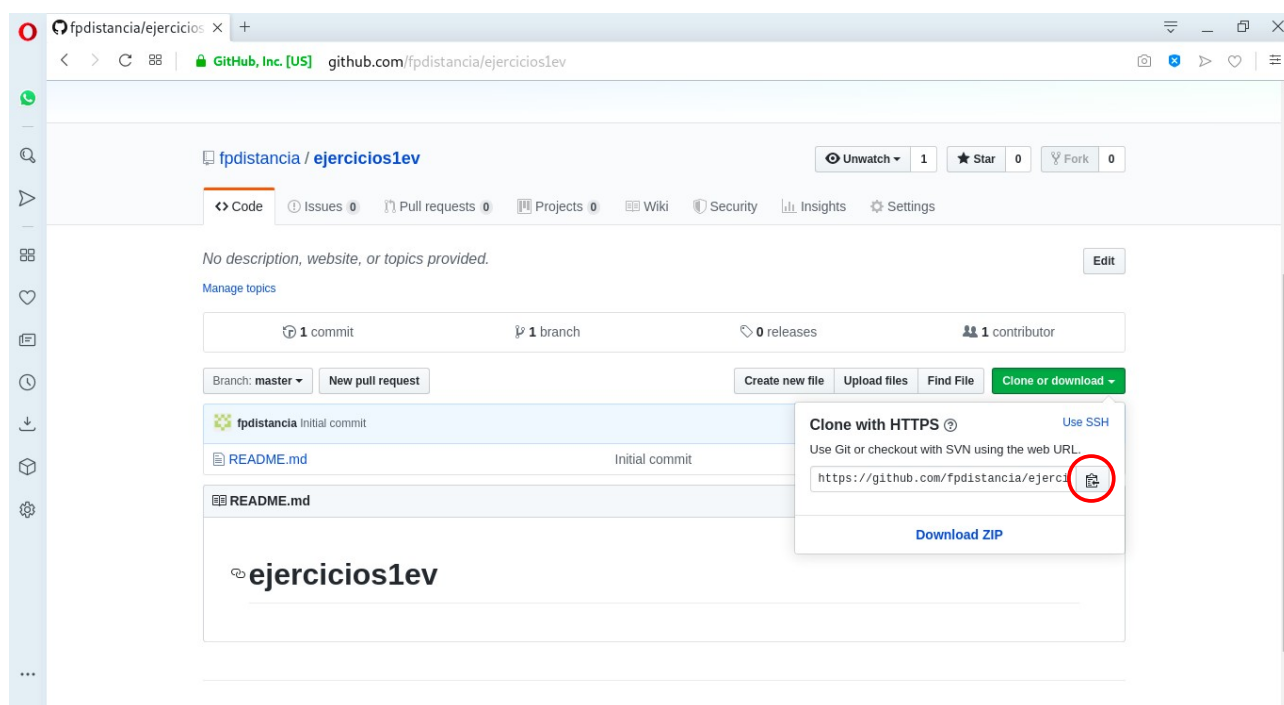
Creación de un proyecto de Eclipse en un repositorio local conectado con un remoto


Si necesitamos un repositorio remoto para un proyecto nuevo de Eclipse, ya sea porque vamos colaborar en él o simplemente por compartir nuestro código con otras personas, podemos partir de las situaciones siguientes:

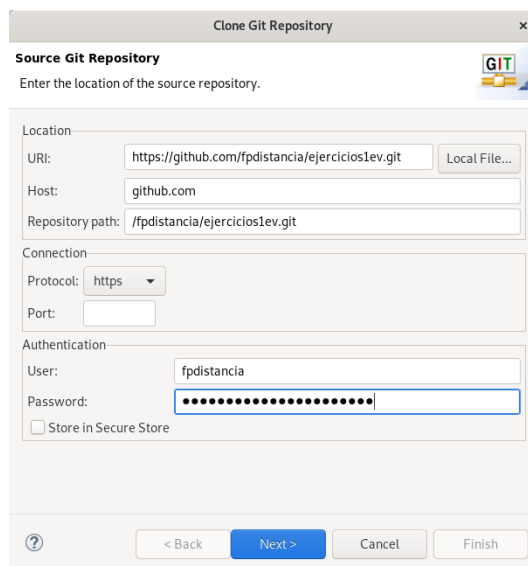
- Ya existe el repositorio local y tiene configurado un repositorio remoto. Este es el caso más sencillo, ya que lo único que tendremos que hacer será crear el proyecto.
- Sólo existe el repositorio remoto. Primero clonamos el remoto y después creamos el proyecto.
- Ya existe el repositorio local, pero no tiene configurado un repositorio remoto. Sin duda, este es el caso más complicado ya que tendremos que configurar el remoto en el repositorio local, antes o después de crear el proyecto.
- No existe ninguno de los dos. En este caso, creamos el repositorio remoto, lo clonamos y creamos el proyecto.

En todos los casos creamos el proyecto en el repositorio local como se indica en el apartado anterior.

Podemos clonar un repositorio remoto desde la línea de comando o desde Eclipse a partir de su URL. Veamos un ejemplo con GitHub (en GitLab y Bitbucket la forma de proceder va a ser muy parecida). Iniciamos sesión en GitHub, y creamos un repositorio (aunque este ejemplo lo creamos para que inicialmente ya contenga un archivo *README.md*, también podríamos crear un repositorio vacío). En la página principal del repositorio vamos a ver un botón llamado *Clone or Download*. Al pulsar sobre él, se despliega una pequeña ventana donde se muestran las direcciones HTTPS y SSH del repositorio:



A la derecha de la URL (rodeado con el círculo rojo en la imagen) podemos ver un botón para copiarla en el portapapeles. Una vez copiada, nos vamos a Eclipse y activamos la perspectiva de Git. En la vista *Git Repositories* pulsamos el botón *Clone a Git repository and add the clone to this view*  para abrir el asistente de clonación:

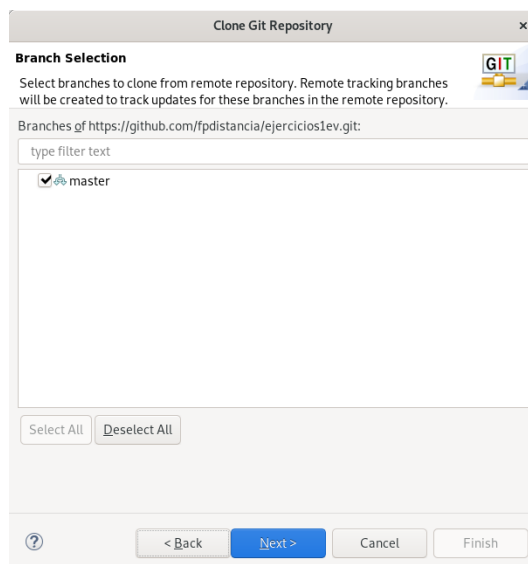


The dialog box is titled "Clone Git Repository". It contains the following fields and options:

- Location:**
 - URI: (A red circle highlights the copy icon to the right of the URI field.)
 - Host:
 - Repository path:
- Connection:**
 - Protocol:
 - Port:
- Authentication:**
 - User:
 - Password:
 - ☐ Store in Secure Store

At the bottom, there are buttons: "< Back", "Next >", "Cancel", and "Finish".

Comenzamos pegando la dirección que hemos copiado en el campo *URI* y se rellenarán automáticamente los campos *Host*, *Repository path* y *Protocol*. El campo *Port* normalmente quedará en blanco para utilizar el puerto por defecto y rellenamos los campos *User* y *Password* con nuestras credenciales de inicio de sesión en el servidor. Pulsamos el botón *Next* para ir realizar el siguiente paso del asistente:

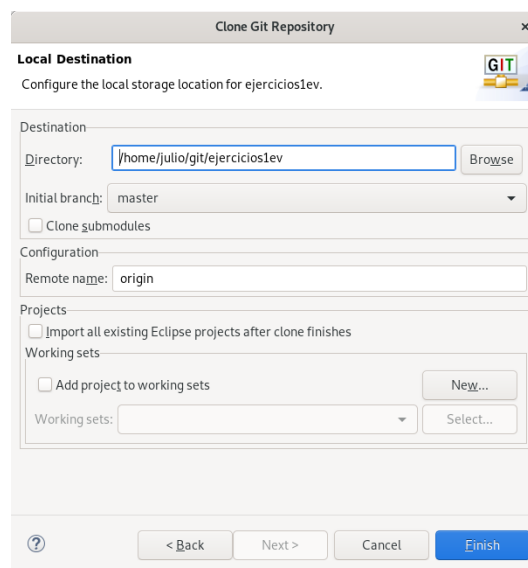


The dialog box is titled "Clone Git Repository" and shows the "Branch Selection" step. It contains the following elements:

- Branch Selection:**
 - Text: "Select branches to clone from remote repository. Remote tracking branches will be created to track updates for these branches in the remote repository."
 - Text: "Branches of https://github.com/fpdistancia/ejercicios1ev.git:"
 - Filter:
 - List: A list box containing one item: "✓ master" (with a branch icon).
 - Buttons: "Select All" and "Deselect All".

At the bottom, there are buttons: "< Back", "Next >", "Cancel", and "Finish".

Seleccionamos las ramas que vamos a clonar (al tratarse de un repositorio que acabamos de crear solo veremos una rama llamada *master* que ya se encuentra seleccionada) y pasamos al siguiente y último paso. En él especificamos el directorio de destino, la rama inicial en la que se va a situar y el nombre con el que se identificará el remoto. Normalmente nos servirán los valores por defecto que nos propone el asistente:

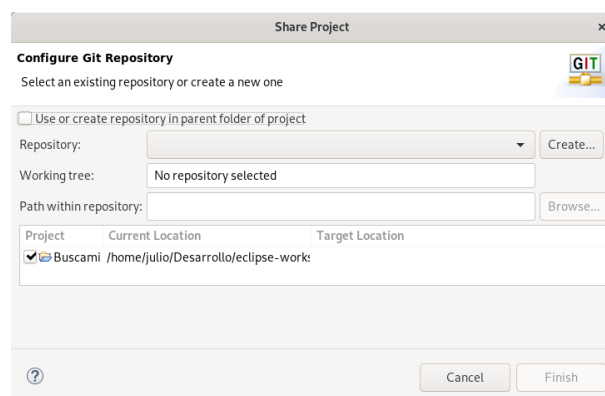


La opción *Import all existing Eclipse projects after clone finishes* se utiliza para importar en Eclipse todos los proyectos que estén en el repositorio que se va a clonar, si es que contiene alguno, aunque esto también lo podremos hacer a posteriori. Como en este ejemplo estamos clonando un repositorio remoto que sólo contiene el archivo *README.md*, no tiene sentido activar esta opción.

Pulsamos el botón *Finish* y Eclipse procederá a la descarga de los archivos desde el servidor remoto para completar la clonación. Una vez completada, podremos crear un proyecto de eclipse en el repositorio clonado como se indica en el apartado anterior.

Cómo iniciar el control de versiones en un proyecto de Eclipse que ya existe en el workspace

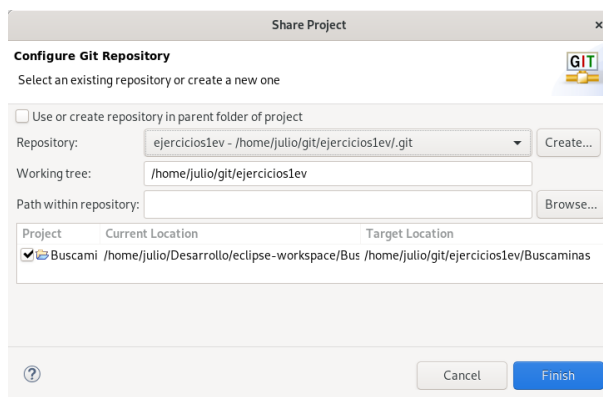
Para iniciar el control de versiones de un proyecto que ya exista en el workspace de Eclipse, nos vamos a la vista *Package Explorer* de la perspectiva de Java y pulsamos sobre el nombre del proyecto con el botón derecho del ratón. En el menú contextual seleccionamos la opción *Team → Share project...* para abrir el cuadro de diálogo siguiente:



Si activamos la opción *Use or create repository in parent folder of project*, podremos crear el repositorio directamente en la carpeta del proyecto. Con esta opción también podríamos convertir todo el workspace en un repositorio de Git, iniciando de esta forma el control de versiones en todos los proyectos que contiene. En cualquier caso, Eclipse desaconseja la creación de repositorios dentro del workspace.

Si no activamos la opción *Use or create repository in parent folder of project*, tenemos que especificar el repositorio de destino. Esto lo podemos hacer en el desplegable que se muestra en el campo *Repository*. Si la lista está vacía o si deseamos crear un repositorio nuevo, pulsamos el botón *Create*.

El campo *Working tree* no es editable y se rellena automáticamente. En el cuadro situado bajo el apartado *Path within repository* podemos observar que el proyecto se moverá del workspace a una carpeta con el nombre del proyecto dentro del repositorio:

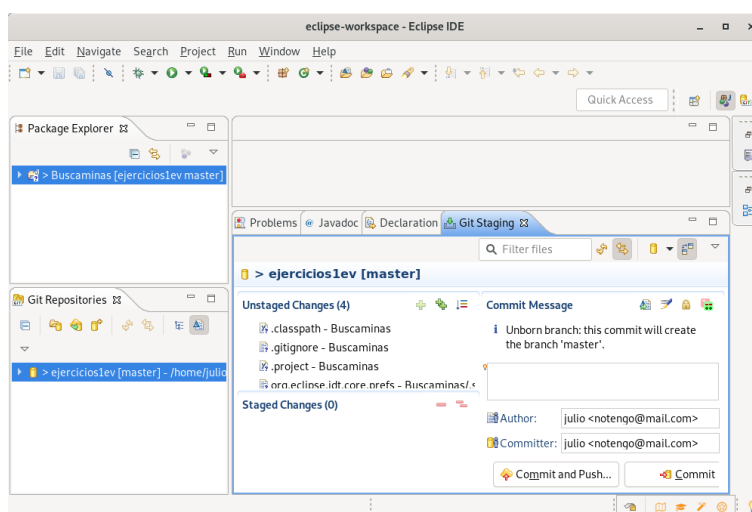



Con el campo *Path within repository* se pueden añadir más niveles de profundidad a la ruta del working tree. El asistente creará automáticamente las carpetas intermedias que especifiquemos en una ruta relativa entre la raíz del repositorio y la carpeta del proyecto. Normalmente dejaremos este campo en blanco.

Al botón *Finish*, habremos iniciado el control de versiones en el proyecto.

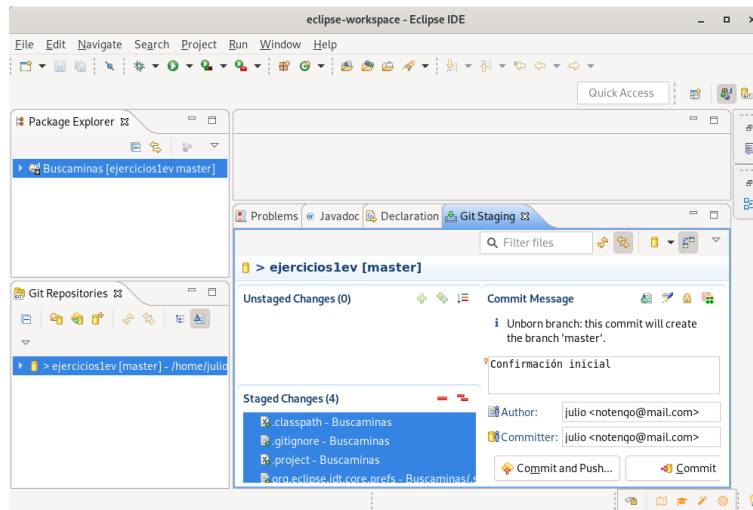
Primera confirmación de un proyecto desde Eclipse

Una vez que tenemos un proyecto bajo control de versiones, podemos comenzar a realizar confirmaciones, bien sea desde la línea de comando o desde el propio entorno de desarrollo. A continuación veremos como realizar la primera confirmación en un proyecto, antes incluso de comenzar a agregar código fuente o cualquier otro contenido. En el caso de Eclipse, las confirmaciones se pueden hacer desde la perspectiva de Java o desde la Perspectiva de Git. Desde la perspectiva de Java, en la vista *Package Explorer*, hacemos clic con el botón derecho del ratón sobre el nombre del proyecto, elegimos la opción *Team → Commit...* y Eclipse abre la vista *Git Staging*:



El primer paso antes de realizar cualquier confirmación consiste en pasar los archivos desde el área denominada *Unstaged Changes* al área *Staged Changes* pulsando sobre el botón .

A continuación escribimos un mensaje de confirmación en el cuadro de texto etiquetado como *Commit Message*. Los campos *Author* y *Committer* se rellenan automáticamente con los valores que hayamos configurado en Git (ver [Configurando Git por primera vez](#)), aunque tenemos la posibilidad de cambiarlos:



Finalmente, realizamos la confirmación pulsando el botón *Commit* o el botón *Commit and Push*. La segunda opción la utilizamos cuando tenemos configurado un remoto y queremos realizar un *push* (subir la confirmación al servidor).