Version Control System for Gis (VCSGis) Documentación de usuario

Borrador preliminar

(v1.0.0)

Sumario

Control de cambios

Versión	fecha	responsable	organización	estado	descripción
1.0.0	07/12/2020	Jose Olivas	Asociación gvSIG	Primera revisión	Borrador preliminar

Introducción

Este documento presenta una introducción resumida al control de versiones **VCSGis** o **Version Control System Gis**. En el se tratan los conceptos generales de un sistema de control de versiones genérico y estos mismos en el caso concreto de *VCSGis*. Además el documento pretende que tras su lectura el usuario no solo conozca las partes o elementos que componen este tipo de sistemas sino que pueda utilizarlo o trabajar con *VCSGis*. Para llevar a cabo la segunda idea esta documentación presenta una guía de uso más ejemplos básicos.

Antes de empezar a describir las partes que componen un sistema de control de versiones hay que definir claramente que son estos y por tanto que es *VCSGis*. Un sistema de control de versiones es un programa o software basado en la centralización de informacion para compartir entre usuarios que a diferencia de un servidor normal, recuerda los cambios que hayan sido realizados en sus datos. Cabe destacar de nuevo que no es un servidor al uso, ya que no solo almacena información, sino que almacena la información así como las modificaciones que los usuarios realizan sobre ella, siendo igual de importante las gestión de esos cambios como la propia información.

También hay que destacar que los sistemas de control de versiones no se centran en ningún tipo de dato en concreto, pudiendo existir sistemas que controlan archivos fuente, imágenes… dependiendo de la necesidad de los usuarios. En el caso de VCSGis al estar orientado a un perfil técnico especialidado en informacion geografica este gestiona información sobre tablas y capas (tablas con información geométrica).

Sin mas dilación comenzamos a detallar las diferentes partes que componen un un sistema de control de versiones tomando como ejemplo a *Version Control System Gis* (*VCSGis*).

Empezando

Instalación de VCSGis

TODO

Pendiente de realizar documentacion

Requerimientos del sistema

TODO

Pendiente de realizar documentacion

Conceptos básicos del control de versiones

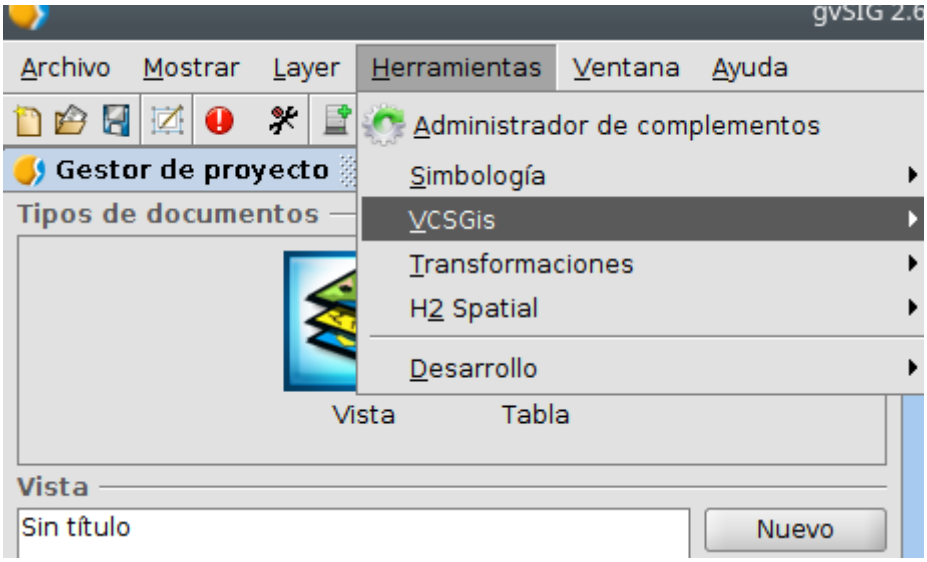
Antes de que entremos a ver como funciona *VCSGis* es importante tener una visión general de cómo funciona un *Sistema de Control de Version* (*VCS*) y los términos que se utilizan.

- El **repositorio** VCSGis usa una base de datos central que contiene todos los datos cuyas versiones se controlan y sus respectivas historias. Ésta base de datos se conoce como el **repositorio**. El repositorio normalmente esta en un servidor de base de datos, que provee a pedido el contenido a los clientes de *VCSGis*. Si solo puede hacer una copia de seguridad de una sola cosa, hágala del repositorio, ya que es la copia maestra de toda su información.
- **Copia de trabajo**

  Aquí es donde se realiza el trabajo en serio. Cada usuario tiene su propia copia de trabajo, comunemente conocida como *caja de arena* en su ordenador local. Usted puede obtener la última versión del *repositorio*, trabajar en ella localmente sin perjudicar a nadie, y cuando haya terminado con los cambios que ha realizado puede *confirmar* (commit) sus cambios en el *repositorio*.

  Una *copia de trabajo* de *VCSGis* no contiene la historia de los datos, pero sí contiene una copia de los datos que existían en el repositorio antes que comience a hacer cambios. Esto significa que es fácil verificar qué cambios ha realizado.

También necesita saber donde encontrar *VCSGis* dado que no hay mucho para ver en los menus y herramientas de *gvSIG desktop*. Esto se debe a que *VCSGis* es una complemento de *gvSIG desktop*, así que primero inicie el *gvSIG dektop*. Hága click en el menu *"Herramientas"* y debería ver una entrada nueva **VCSGis**:


Archivo
Mostrar
Layer
Herramientas
Ventana
Ayuda
Administrador de complementos
Simbología
VCSGis
Transformaciones
H2 Spatial
Desarrollo
Gestor de proyecto
Tipos de documentos
Vista
Tabla
Vista
Sin título
Nuevo

Repositorio

Como se dijo en el apartado anterior *VCSGis* es un sistema centralizado para compartir información, estando en su núcleo un **repositorio**, es decir un almacén central de datos, que salva la información en forma de tablas. Este almacén no es más que un sistema de base de datos que tiene tanto la información que modifican los usuarios del software, en el caso de *VCSGis* tablas o capas, como las tablas que almacenan los cambios producidos en los anteriores.

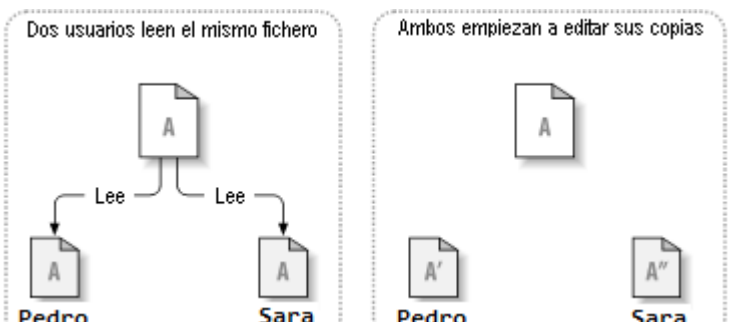
Cada tabla del usuario mantendra su propio control de versiones de forma independiente a las demas tablas del repositorio, llevando un control de los registros que se van insertando, modificando o borrando en ella. La unidad minima de informacion que sigue o controla *VCSGis* es el registro de una tabla, sin hacer seguimiento de los cambios en los campos de cada registro.

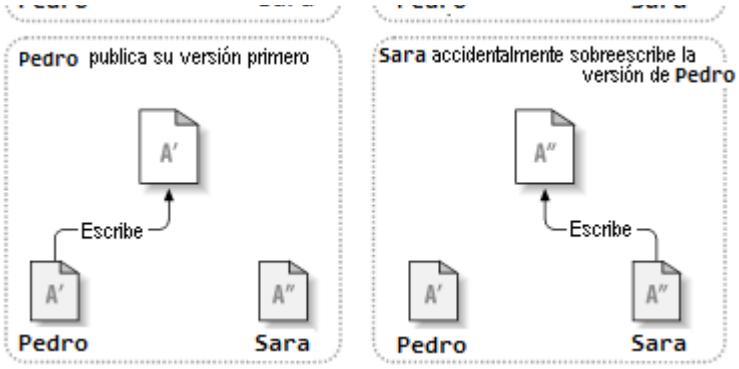
Cuando un cliente accede a los datos del repositorio en *VCSGis*, normalmente ve la última versión de este, es decir la informacion actual con las capas y tablas . Sin embargo el cliente también tiene la posibilidad de ver los estados previos del repositorio. Por ejemplo, un cliente podría preguntarse *¿Qué contenía esta tabla este miércoles?* o *¿Quién fue la última persona en editar esta capa y qué cambios realizó?* Estas son el tipo de posibilidades que diferencian a los sistemas de control de versiones: sistemas diseñados para guardar y registrar las modificaciones de los datos a lo largo del tiempo.

Modelos de versionado

Tras la definición de los sistemas de control de versiones y más concretamente *VCSGis*, así como el núcleo de estos o repositorio, puede intuirse la existencia de un punto crítico en este tipo de software. Este punto no es otro que el intercambio de información actualizada entre usuarios evitando que los cambios de estos no se pisen unos a otros, ya que parece sencillo pensar que los usuarios de forma accidental pueden sobrescribir cambios de los demás al almacenar la información de los suyos en el repositorio.

Para entender mejor lo anterior consideramos el siguiente escenario. Suponga que tiene dos compañeros de trabajo, Pedro y Sara. Cada uno decide editar la misma tabla del repositorio a la vez. Si Pedro guarda sus cambios en el repositorio primero, es posible que, unos momentos después, Sara pueda accidentalmente sobreescribirlos con su propia versión. Mientras que la versión de la tabla de Pedro no se ha perdido para siempre, porque el sistema recuerda cada cambio, cualquier cambio que Pedro hizo no estará en la versión nueva de la tabla de Sara, porque ella nunca vio los cambios de Pedro. De lo anterior podemos decir que el trabajo de Pedro está efectivamente perdido, o al menos falta en la última versión de la tabla.



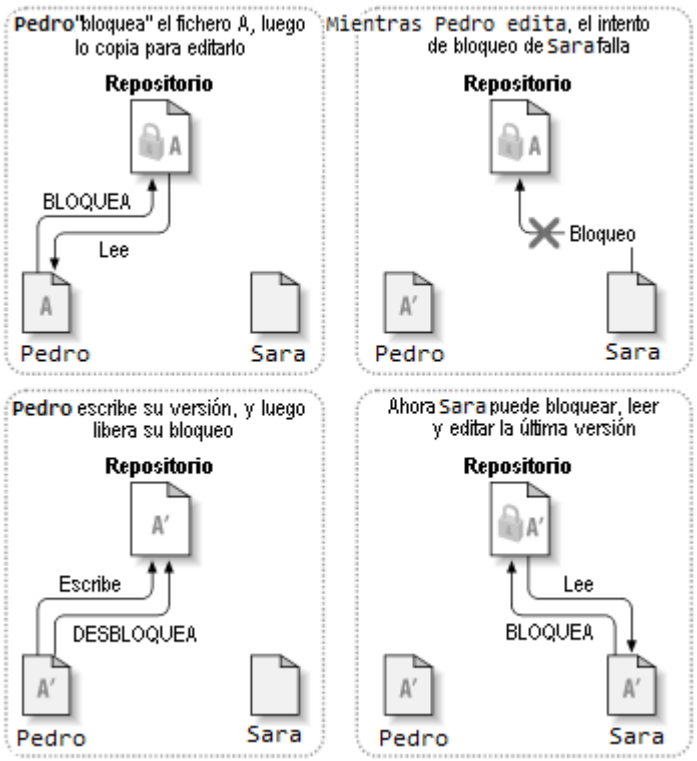


Para solucionar ese problema existen dos modelos;

- Bloquear-Modificar-Desbloquear
- Copiar-Modificar-Fusionar

Bloquear-Modificar-Bloquear

Muchos sistemas de control de versiones utilizan un modelo Bloquear-Modificar-Desbloquear para enfrentarse al problema anterior, lo cual es una solución muy simple. En estos sistemas, el repositorio sólo permite que una persona cambie un archivo. Pedro primero debe bloquear el archivo antes que pueda empezar a hacer cambios en él. Si Pedro ha bloqueado un archivo, entonces Sara no puede hacer ningún cambio en él. Si ella intenta bloquear el archivo, el repositorio le denegará la petición. Todo lo que ella puede hacer es leer el archivo, y esperar a que Pedro termine sus cambios y libere el bloqueo. Después que Pedro desbloquee el archivo es el turno de Sara para bloquear y editar.



El problema con el modelo bloquear-modificar-desbloquear es que es un poco restrictivo, y a menudo se convierte en una molestia para los usuarios:

- *El bloqueo causa muchos problemas administrativos*. A veces los usuarios bloquean archivos y se olvidan de desbloquearlos, dejando sin acceso a la

edición de estos por parte de los demás usuarios. Para solucionar esto se tiene que gestionar el desbloqueo por parte de un administrador lo cual hace que la situación cause un montón de retraso y pérdidas de tiempo innecesarias.

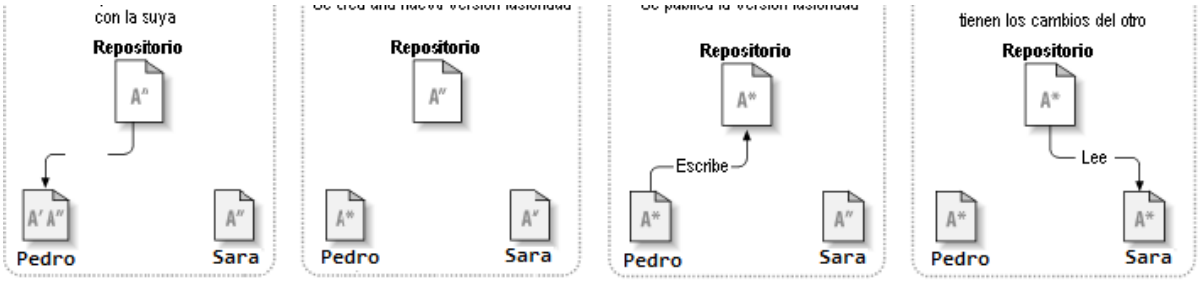
- *El bloqueo puede causar procesos en serie innecesarios*. ¿Qué ocurre si un usuario está editando el inicio de un archivo de texto, y otro simplemente quiere cambiar la parte final del mismo archivo? Esos cambios no se superponen en absoluto y por tanto se podría editar el archivo de forma simultánea, no existiendo la necesidad de tomar turnos en este tipo de situaciones.

- *El bloqueo puede causar una falsa sensación de seguridad*. Imagine que un usuario bloquea y edita el archivo A, mientras otro usuario distinto simultáneamente bloquea y edita el archivo B. Pero supongamos también que los archivos A y B dependen uno del otro, y que los cambios hechos a cada uno son incompatibles. De repente A y B ya no funcionan juntos y el sistema de bloqueo no tiene forma de prevenir este problema, sin embargo, de alguna forma este sistema dio una sensación de falsa seguridad. Es fácil para los usuarios imaginar que al bloquear los archivos están realizando una tarea segura y aislada, y la realidad es que no, inhibiendoles de discutir si sus cambios son compatibles entre si.

Copiar-Modificar-Fusionar

VCSGis y otros sistemas de control de versiones usan el modelo Copia-Modificación-Fusión como alternativa al modelo de bloqueo. En este modelo, cada cliente de los usuarios lee el repositorio y crea una **copia de trabajo** personal de las capas o tablas de modo que los usuarios trabajan en paralelo, modificando sus copias privadas. Finalmente, las copias privadas son unificadas conjuntamente en una nueva versión final mediante el asesoramiento del sistema de control de versiones, pero es un humano en ultima instancia el responsable de hacer esta acción de manera correcta.

Para entender mejor lo anterior consideramos el siguiente escenario; Digamos que tanto Pedro como Sara crean copias de trabajo de la misma capa, copiadas del repositorio. Ellos trabajan simultáneamente, y hacen cambios al mismo archivo A dentro de sus copias. Sara es la primera en grabar sus cambios en el repositorio. Cuando Pedro intenta grabar sus cambios más tarde, el repositorio le informa que su archivo A está desactualizado. En otras palabras, que el archivo A en el repositorio ha cambiado de alguna forma desde la última vez que lo copió. Por lo que Pedro le pide a su cliente que fusione cualquier nuevo cambio del repositorio dentro de su copia de trabajo del archivo A. Lo más seguro es que los cambios de Sara no se superpongan a los suyos; por lo que una vez que ambos conjuntos de cambios se han integrado, él graba su copia de trabajo de nuevo en el repositorio.



¿Pero qué ocurre si los cambios de Sara se superponen a los cambios de Pedro? ¿Qué hacemos entonces? La situación se denomina un **conflicto**, y normalmente no es problema. Cuando Pedro le pide a su cliente que fusione los últimos cambios del repositorio en su copia de trabajo A, se da el caso anterior, la copia de trabajo de Pedro marca que está en un estado de conflicto. En ese estado de conflicto Pedro es capaz de ver ambos conjuntos de cambios conflictivos, y manualmente podrá elegir entre ellos cual es el más acertado. Tenga en cuenta que el software no puede resolver conflictos automáticamente, ya que sólo los humanos son capaces de entender y hacer las elecciones necesarias de forma correcta. Una vez que Pedro haya resuelto manualmente los cambios que se superponían, puede guardar de forma segura el archivo fusionado al repositorio.

El modelo Copiar-Modificar-Fusionar puede parecer un poco caótico, pero en la práctica, funciona extremadamente bien. Los usuarios pueden trabajar en paralelo, sin que tengan que esperar nunca uno por otro, y cuando trabajan en los mismos archivos, resulta que la mayoría de los cambios concurrentes no se superponen en absoluto, siendo los conflictos muy poco frecuentes. Además, el tiempo que lleva resolver conflictos es mucho menor que el tiempo perdido por un sistema que implementa el modelo de bloqueo.

Como curiosidad decir que hay una situación donde el modelo Bloquear-Modificar-Desbloquear resulta mejor que este, y es cuando se tiene archivos no-fusionables, por ejemplo imágenes. Si dos personas cambian una imagen a la vez, no hay forma de fusionar esos cambios, y uno de ellos perderá sus cambios.

Copias de trabajo

A modo de recordatorio decir que el sistema de control de versiones *VCSGis* utiliza el modelo Copia-Modificación-Fusión como modelo de versionado, por lo que utiliza **copias de trabajo**.

Una copia de trabajo en *VCSGis* esta formada por una base de datos en su sistema local, la cual contiene una colección de tablas o capas que pueden ser modificados sin ningún tipo de problema. Su copia de trabajo es su área de trabajo privada y *VCSGis* nunca incorporará los cambios de otra gente, ni hará que sus cambios estén disponibles para los demás, a menos que se lo pida expresamente. Básicamente la copia de trabajo es una captura/fotografía/snapshot del repositorio en un instante concreto.

Una copia de trabajo de *VCSGis* también contiene algunas tablas extra, creados y mantenidos por *VCSGis*, para ayudar a llevar a cabo la gestión de cambios. En estos se almacenan los numero de revisión, un sistema de numeración única que identifica de manera inequívoca los diferentes cambios registrados en el repositorio. Estos codigos de revision, únicos para cada capa o tabla, ayudan al software a reconocer qué capas o tablas contienen cambios no publicados o qué capas o tablas contienen contenidos desfasados respecto a la información del repositorio.

Flujo de trabajo (Modelo Copiar-Modificar-Fusionar)

Este apartado define la estructura o forma de trabajar teórica con un sistema de control de versiones que presenta el modelo Copiar-Modificar-Fusionar como podría ser *VCSGis*. Especificar que estos mismos procesos serán explicados desde un punto de vista práctico para *VCSGis* en el apartado siguiente.

Paso 1: Creación de repositorio. Se lleva a cabo por el administrador y con el se da inicio al proceso.

Paso 2: Creación de copia de trabajo. Proceso realizado por los diferentes usuarios o clientes del repositorio y como se dijo en el apartado anterior consiste en la captura del repositorio en un instante determinado. Esa captura completamente funcional permite cualquier acción de edición o modificación y los cambios realizados en esta serán propuestos por el usuario para formar parte del repositorio.

Paso 3: Realización de cambios en la informacion de su copia de trabajo y verificación que funcionan correctamente

Paso 4: Publicación de cambios. El sistema de control de versiones *VCSGis* en este caso provee al usuario de comandos para publicar sus cambios en el repositorio por tanto estar estos disponibles para todos los usuarios registrado es en ese repositorio. En el caso de que los demás usuarios hayan publicado antes sus propios cambios, el software le provee de comandos para fusionar esos cambios dentro de su copia de trabajo tras leer el repositorio. La fusión puede presentar dos caso bien diferenciados:

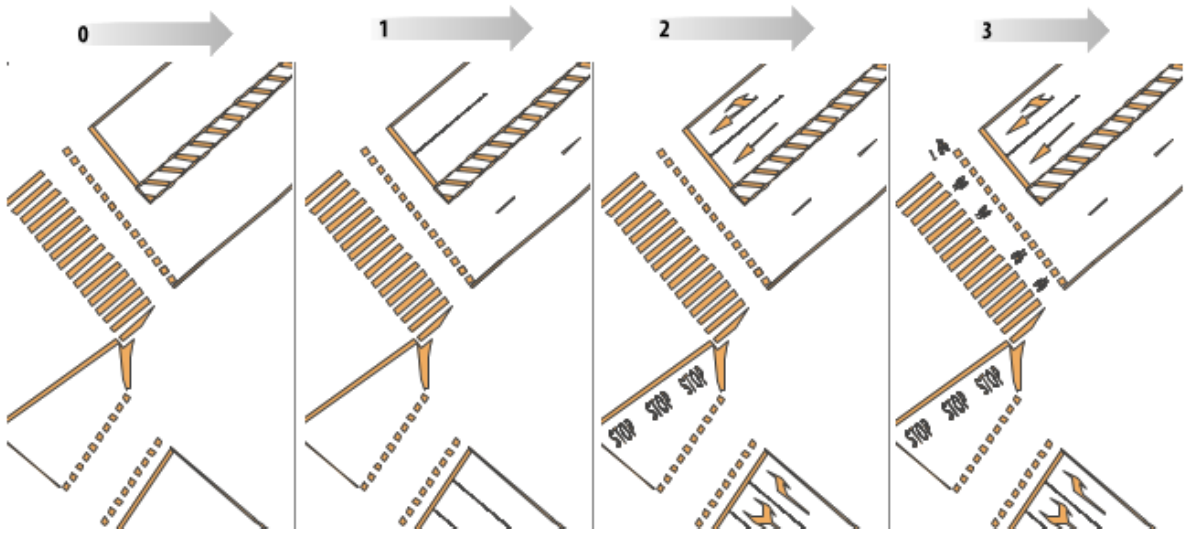
- Los cambios del usuario afectan a elementos no modificados anteriormente por los demás usuarios y por tanto se mezclan y fusionan las tablas.
- El usuario ha realizado cambios sobre elementos modificados anteriormente por otros usuarios y ya registrados en el repositorio, en ese caso se entra en conflicto y el usuario decide que cambios son los que se quedan finalmente en el repositorio los propios o los ya presentes.

Al acto de publicar los datos en el servidor se le denomina hacer **confirmar**, *commit*, y al proceso de actualizar la copia de trabajo con la información del repositorio se le denomina **actualizar**, *update*.

Revisiones

Volviendo al proceso de publicar o *confirmar* los datos en el servidor, hacer *commit*, este nos genera como resultado una **revisión** a la cual esta ligada cualquier cambio realizado en la tabla de la copia de trabajo modificada; cambiar el contenido de las tablas, crear, borrar, renombrar y copiar registros o geometrias... Siendo solo registrada en el repositorio si todos los cambios de esa tabla pueden realizarse (transacción atómica). En el caso de que algún cambio de una tabla no pueda hacerse, ninguno de los demás cambios de la tabla referentes a esa revisión se realizará. Las revisiones presentan un identificador único de la revisión y en el caso de VCSGis cada tabla de la copia de trabajo presentara un control de la revisión individual con identificadores únicos propios.

Para ilustrar lo anterior se presenta el siguiente esquema del funcionamiento de las revisiones;



En la primera revisión, revisión 0, se han creado las lineas base de la calzada, en la siguiente revisión, revisión 1, se añaden las separaciones entre carriles y medaneras, en la revisión 2 se añaden las feclas indicadores del sentido y señales de stop, y por ultimo, en la revisión número 3 se han añadido las señales del carril bici.

Tras la explicación anterior parece lógico pensar que las revisiones y su número o identificador único son el mecanismo que utilizan los sistemas de control de versiones y por tanto *VCSGis* para gestionar el estado de la copia de trabajo con respecto al repositorio y viceversa.

Estados de tablas y registros

Los estados que puede presentar la copia de trabajo y las diferentes capas y tablas que los componen son las siguientes;

- *Sin cambios y actualizado.* La tabla no se ha cambiado en la copia de trabajo y no se han confirmado cambios a esa tabla en el repositorio desde su revisión de trabajo. Una acción *commit* de esa tabla no hará nada, y una actualización de ella tampoco.

- *Cambiado localmente y actualizado.* La tabla ha sido cambiado en la copia de trabajo y no se ha confirmado ningún cambio a esa tabla en el repositorio desde su revisión base. Hay cambios locales que no se han confirmado en el repositorio, por lo que al hacer commit la tabla se confirmaran sus cambios actualizando el repositorio. Si se procede a hacer una actualización o update de la tabla no se realizarà ya que la copia de trabajo actual posee la versión mas moderna de esta presente en el repositorio.

- *Sin cambios y desactualizado.* La tabla no ha sido cambiado en la copia de trabajo, pero ha sido cambiado en el repositorio. La tabla deberá ser actualizado en algún momento para presentar el mismo contenido que la versión del repositorio. Un comando commit sobre la tabla no hará nada ya que no hay cambios en local y el comando actualizar o update traerá los últimos cambios del repositorio a su copia de trabajo.

- *Cambiado localmente y desactualizado.* La tabla se ha cambiado tanto en la copia de trabajo como en el repositorio. No podra ejecutar un *commit* sobre la tabla ya que necesita actualizar en primer instancia la copia de trabajo con el repositorio. Para ello necesita realizar una actualización o update de la copia de trabajo que intentará fusionar los cambios del repositorio con los cambios locales. Si *VCSGis* no puede completar la fusión de una forma plausible automáticamente, le dejará al usuario la tarea de resolver los conflictos. Por ultimo, una vez actualizado, se debera realizar una petición commit para registra los cambios locales en el repositorio y terminará el proceso.

Conexión con el repositorio

El sistema de control de versiones VCSGis dispone de dos opciones a la hora de realizar la conexión con el repositorio;

- Conexión a una BBDD de nuestra red local.
- Conexión a un servidor VCSGis.

La primera opción y más usual consiste realizar la conexión a un repositorio situado en un servidor local y se realiza identificando el fichero de la BBDD en la estructura de carpetas. La segunda opción al ser online permite la conexión al repositorio mediante URL, ampliando la operabilidad del software.

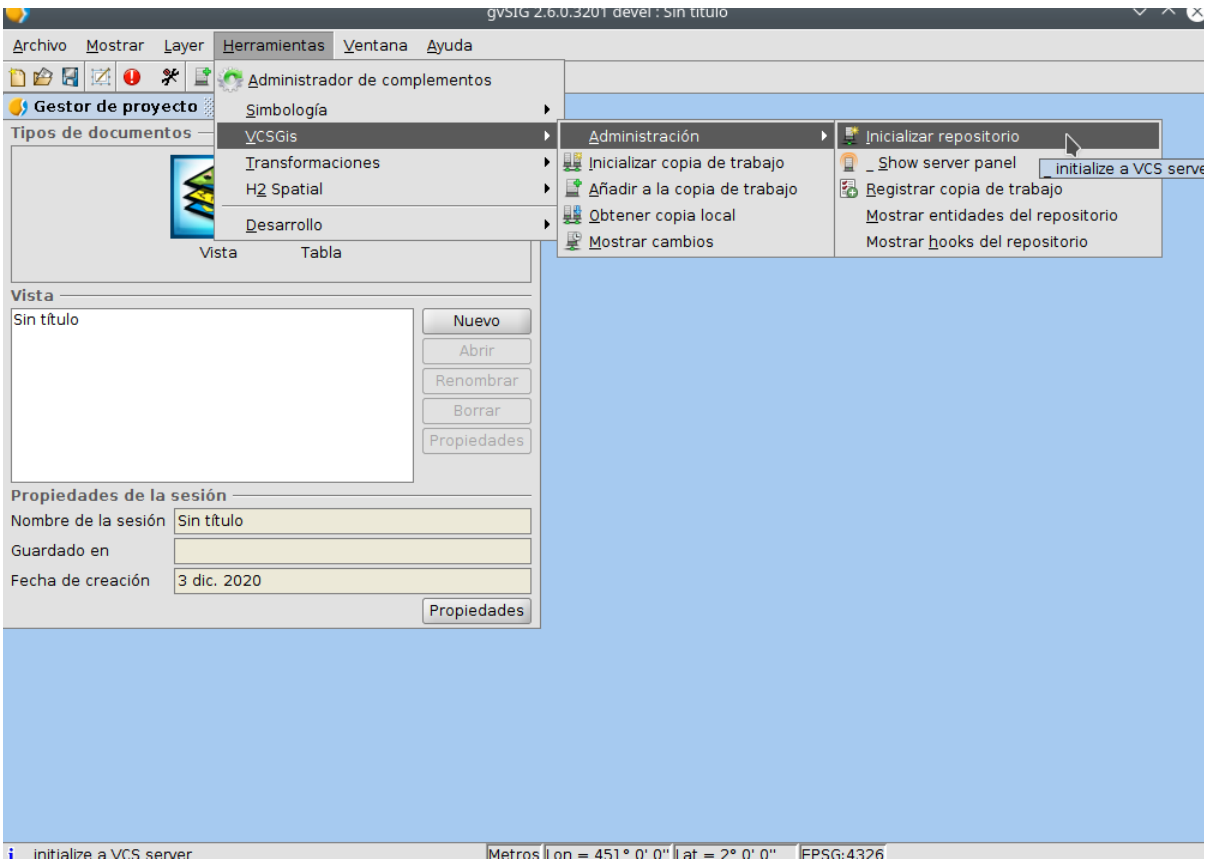
Utilización básica

A continuación, como se mencionó en el Apartado *Flujo de trabajo (Modelo Copiar-Modificar-Fusionar)* se realiza un aproximación más práctica a la manera o forma de trabajar que tiene el software VCSGis. Para llevar a cabo lo anterior se seguir el flujo de trabajo indicado en el apartado antes mencionado´adaptando los puntos a nuestro sistema de control de versiones.

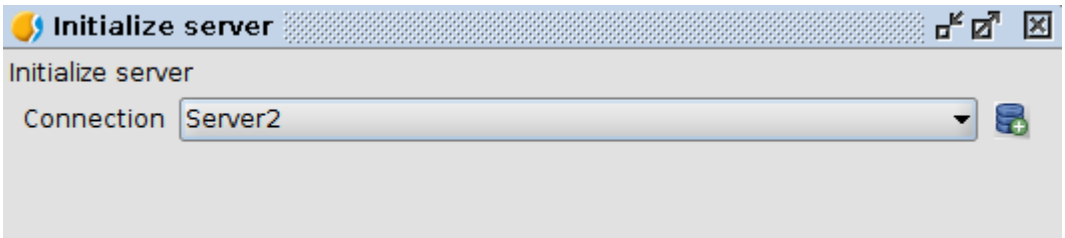
Creación de un repositorio

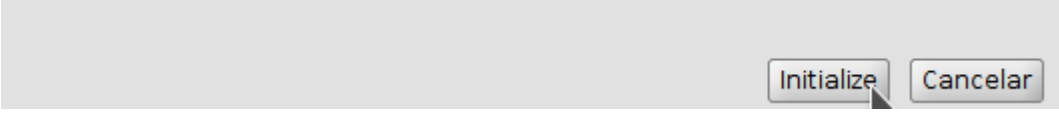
Para la creación del repositorio tal y como se destaca anteriormente en el documento disponemos de dos opciones, conexión local a un BBDD como repositorio o conexión a un servidor o repositorio remoto (online). En esta guía nos inclinaremos por la primera opción o conexión local.

Para inicial la conexión al repositorio hay que dirigirse a la pestaña *Herramientas*, desplegarla y buscar la opción *VCSGis*. Una vez dentro seleccionaremos la opción *Administración* y dentro de esta *Inicializar repositorio*. Puede ver gráficamente lo mencionado anteriormente en la siguiente ilustración.

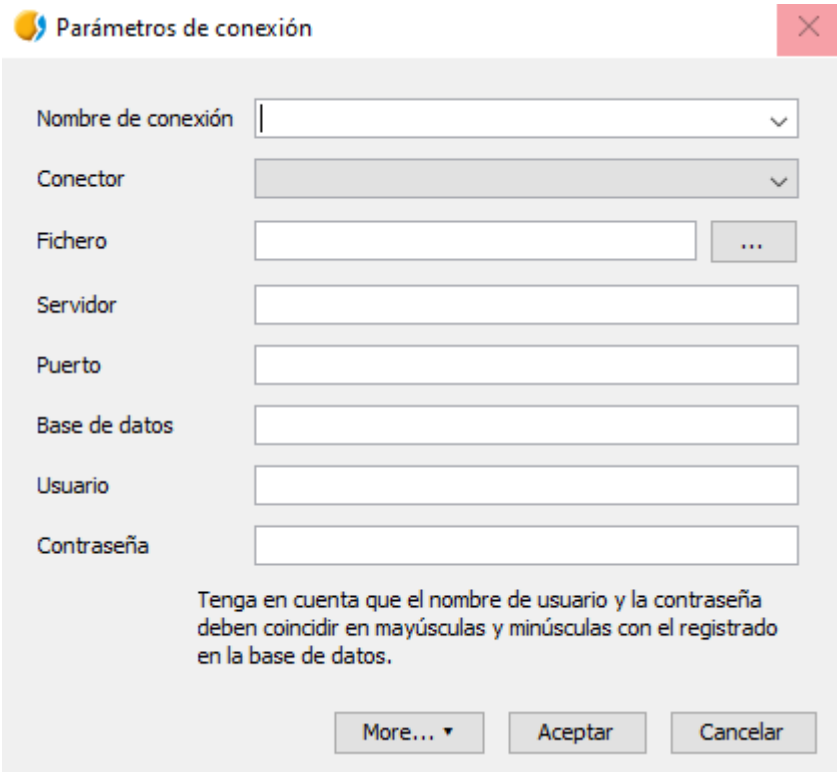


Tras pulsar se obtiene la siguiente venta la cual nos permite seleccionar una conexión a un a BBDD ya existente o generar una nueva.





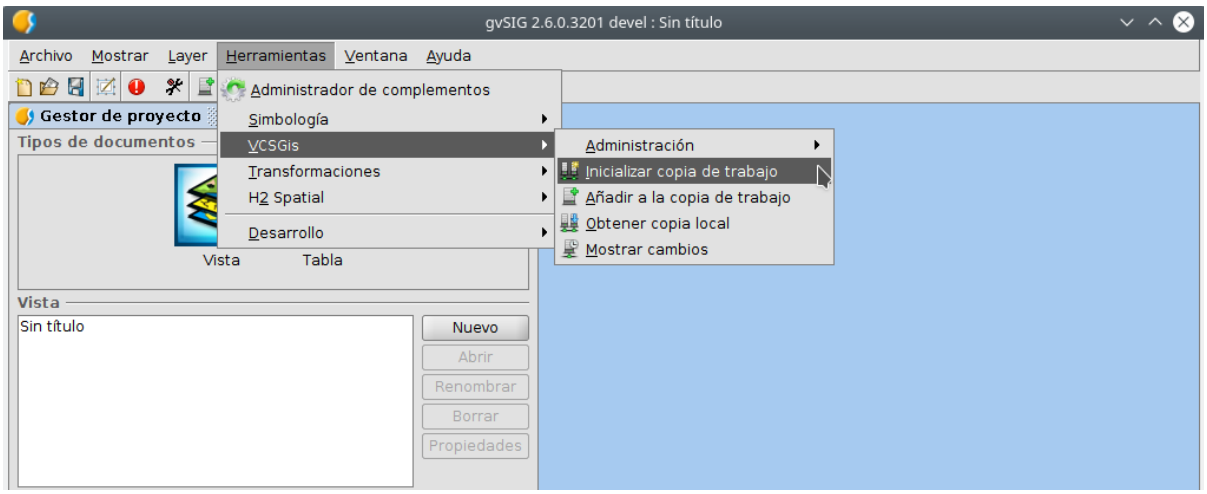
En el caso de que se busque crear una nueva conexión hay que hacer clic sobre el botón situado a la derecha del desplegable en la ventana anterior, acción que iniciará el proceso de creación de una nueva conexión a una BBDD. Esta nueva conexión se realiza mediante la ventana genérica de conexiones a este tipo de datos de gvSIG Desktop, ver la figura siguiente.

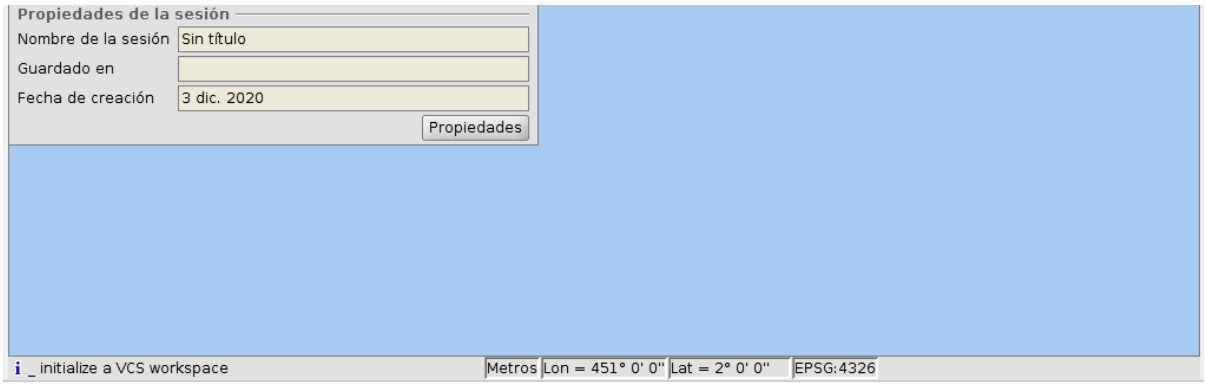


Como se detallo en la explicación teórica, la creación es un proceso destinado a ser ejecutado por el administrador, un usuario regular no debería llevar a cabo este proceso.

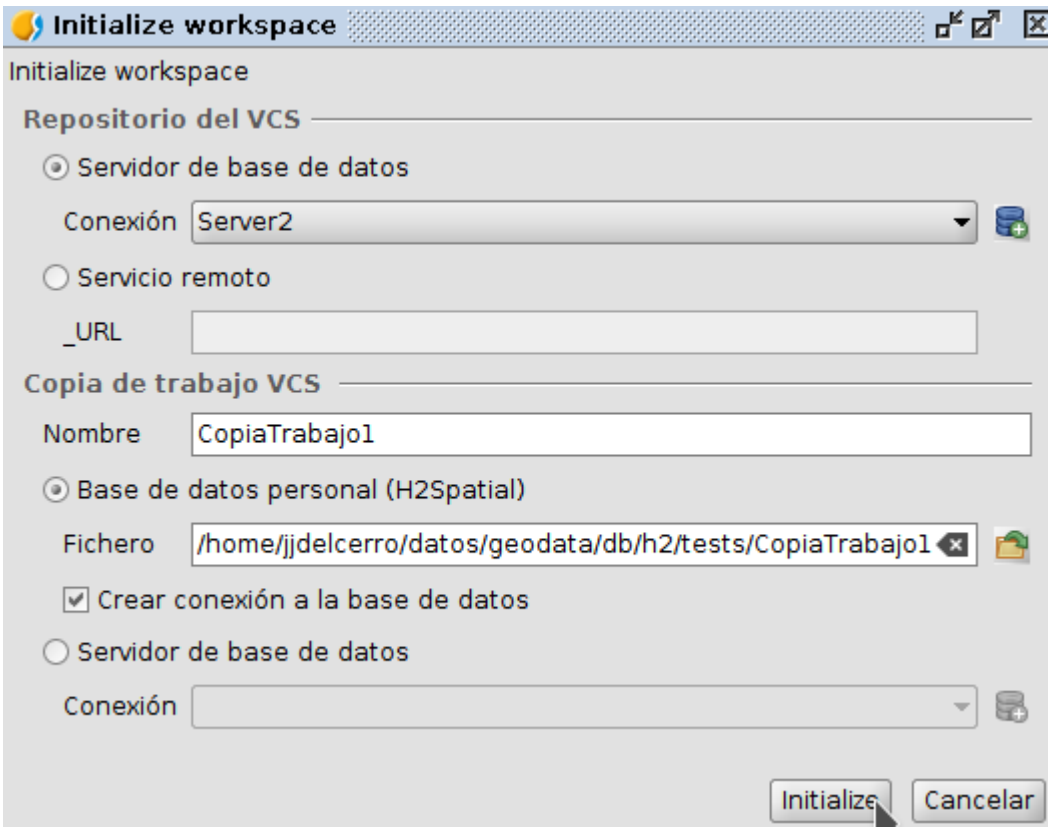
Creación de una copia de trabajo

El siguiente paso es crear la copia de trabajo, proceso realizad al ejecutar el comando *Inicializar copia de trabajo* dentro de las pestaña *VCSGis* situada en la opción *Herramientas* de gvSIG Desktop.





Una vez ejecutado lo anterior se obtiene la siguiente venta cuyos componentes se dividen en dos apartados, *Repositorio del VCS* y *Copia del VCS*.

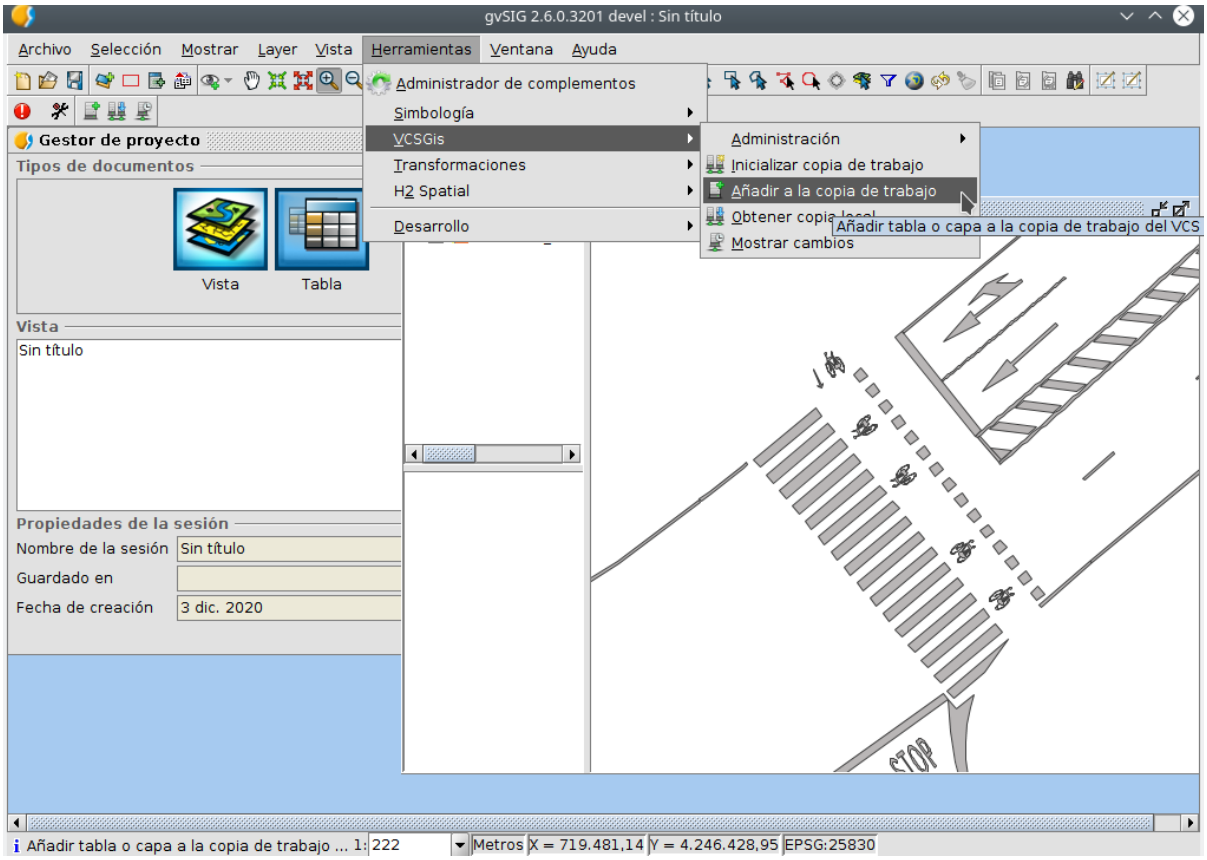


En el apartado *Repositorio del VCS* se selecciona la conexión al repositorio, pudiendo ser local u online, mientras que el apartado *Copia del trabajo VCS* se introducen los parámetros para generar la cueva copia de trabajo. Los parámetros necesarios son el nombre de la copia y la situación de esta, la cual puede ser local, genera un BBDD personal en formato H2Spatial, u online. Si se selecciona la opción local, marcar la opción *Crear conexión a la base de datos* como recomendación.

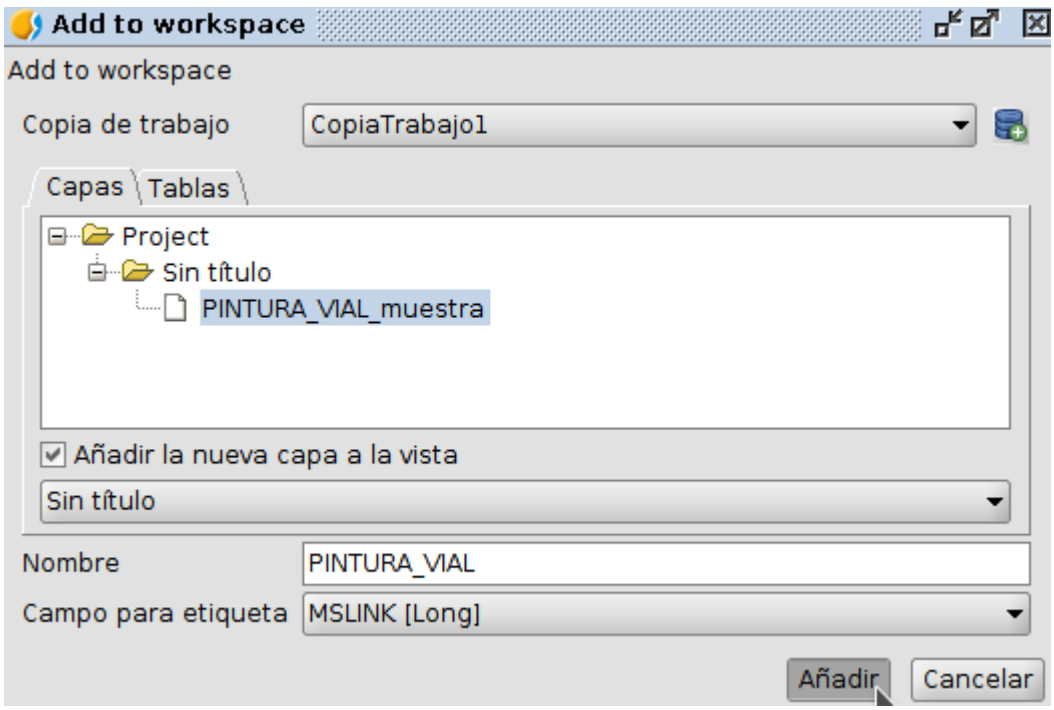
Añadir una capa al repositorio

Tras cerciorarnos de la existencia de un repositorio y de una copia de trabajo se procede a actualizar este con nueva información. Para realizarlo hay que cumplir dos pasos.

El paso 1 consiste en cargar dicha nueva información o capa en la vista. El segundo paso se basa en ir a la opción *Herramientas* del menú de gvSIG Desktop, pestaña *VCSGgis* y pestaña *Añadir a la copia de trabajo*.



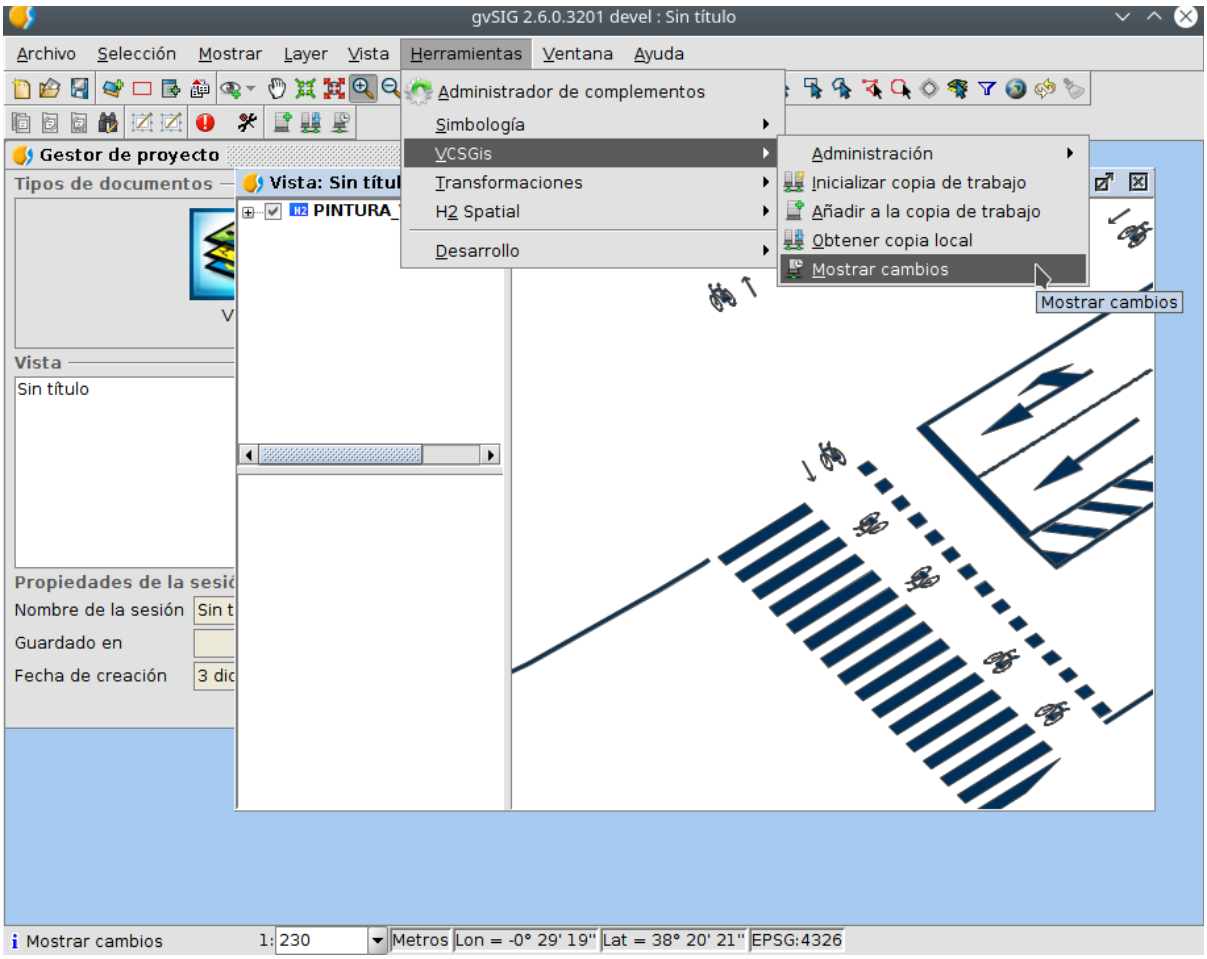
Tras pulsar el comando *Añadir a la copia de trabajo* se desplegá la ventana siguiente donde se selecciona la copia de trabajo donde queremos añadir la capa, la estructura de carpetas actual de gvSIG donde se selecciona la información a añadir y una serie de opciones.



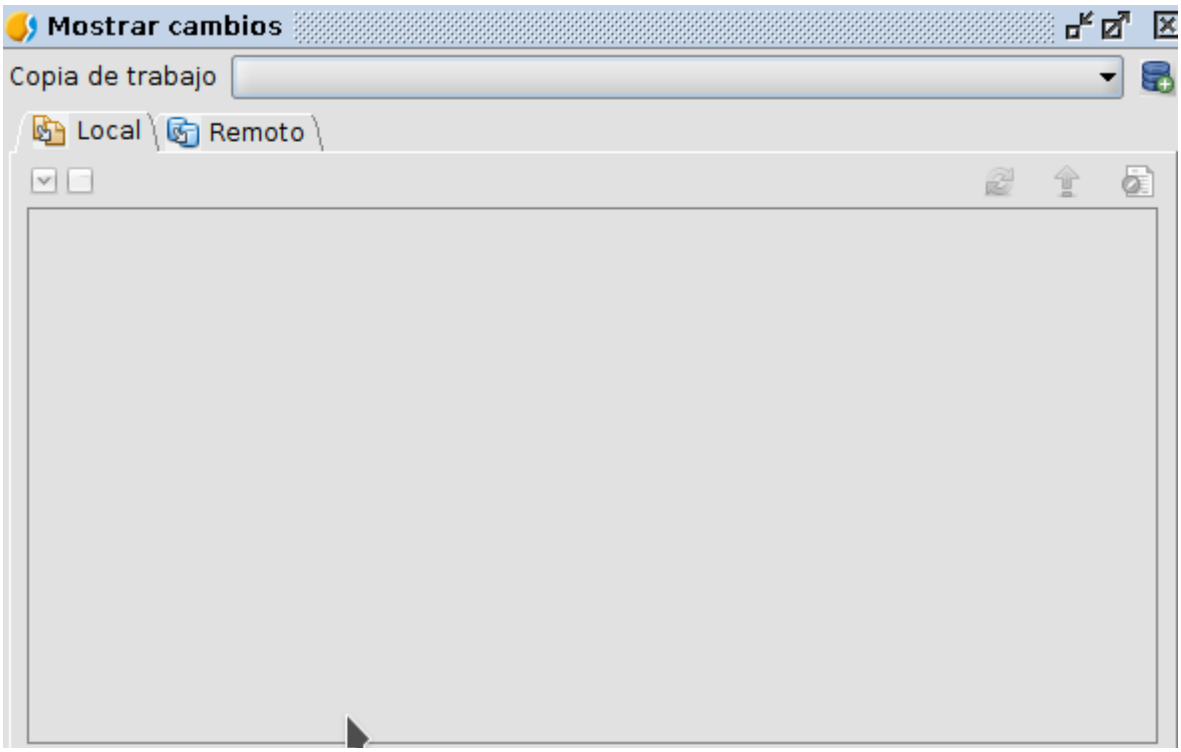
Entre las opciones destaca *Añadir esa capa a la vista*, opción recomendada para iniciar el proceso de edición utilizando el control de versiones. Hay que hacer incapie en la idea anterior, la capa añadida recientemente es sobre la que hay que trabajar ya que es la que tiene un control de versiones asociado, la capa inicial usada para introducir el dato puede eliminarse de la vista ya que los cambios sobre ella no se registan en VCSGis. Las otras dos opciones restantes son nombre y

campo para etiquetas que permiten renombrar la capa y seleccionar que campo queremos que sea representado como campo de etiquetas respectivamente.

El proceso de añadir la nueva capa al repositorio termina cuando tras realizar la carga de información anterior se ejecuta la el comando *Mostrar cambios* situado en la pestaña *VCSGis* dentro de la opción *Herramientas* de del software.



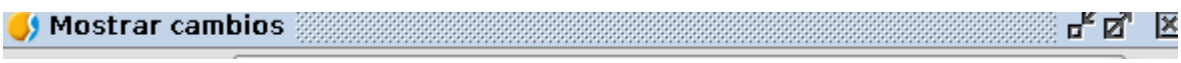
La ejecución de lo anterior genera la siguiente ventana.

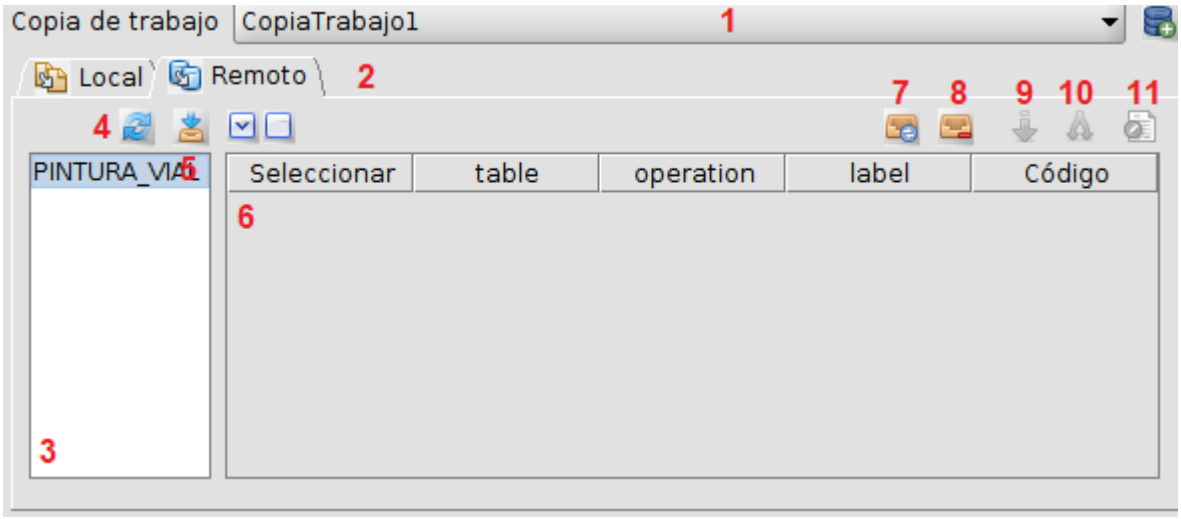


La ventana anterior o ventana *Mostrar cambios* es una de las ventanas más importante de VCSGis y es la encargada como su propio nombre indica de mostrar los cambios entre la copia de trabajo actual y el repositorio. Los cambios entre información se detectan seleccionando la copia de trabajo en cuestión y mediante la gestión de las pestañas *Local* y *Remoto*. Al seleccionar la pestaña ***Local*** se muestran los ***cambios existentes en la copia de trabajo frente al repositorio***. Si por el contrario se selecciona la opción ***Remoto*** se muestran los ***cambios del repositorio frente a la copia de trabajo***. Además de las pestañas, la ventana permite identificar la *Fecha de entrada en vigor* así como un apartado *Comentarios* asociados a los cambios que se van a enviar al repositorio.

Los componentes seleccionada la opción *Local* de esta se listan a continuación:

1. Desplegable para selección de la *Copia de trabajo* sobre la que ver los cambios.
2. Pestañas Local/Remoto.
3. Área de visualización de cambios.
4. Botón de refrescar el área de visualización.
5. Botón para enviar o *confirmar* los cambios locales al repositorio (commit).
6. Botón que muestra un folmulario con los datos del registro seleccionado.
7. Campo para indicar la fecha de entrada en vigor de los cambios que van a ser enviados al repositorio.
8. Campo para introducir un comentario a los cambios que van a ser enviados al repositorio.





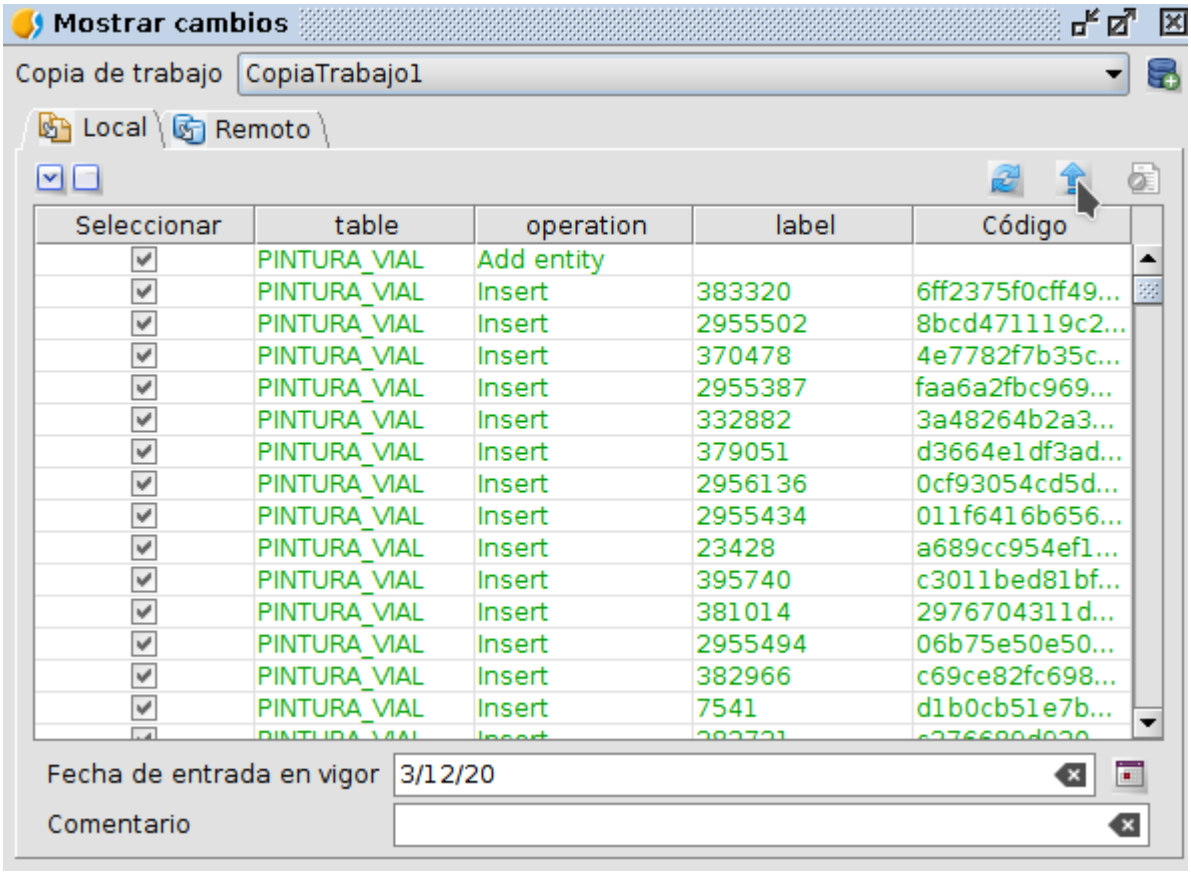
Los componentes seleccionada la opción *Remoto* de esta se listan a continuación:

1. Desplegable para selección de la *Copia de trabajo* sobre la que ver los cambios.
2. Pestañas Local/Remoto.
3. Área con lista de capas de la *Copia de trabajo*.
4. Botón de refrescar la lista de capas de la *Copia de trabajo*.
5. Botón para descargar los cambios que han habido en la capa seleccionada en el repositorio desde la ultima vez que se actualizó esta en la *Copia de trabajo*. Esta operacion puede ser pesada dependiendo de la cantidad de cambios que hayan en el repositorio desde la ultima actualización de la *Copia de trabajo*.
6. Área de visualización de cambios. Muestra los cambios que han habido en el repositorio en relacion a las capas de la *Copia de trabajo*.
7. Actualiza el área de visualización de cambios, releyendo estos de la informacion almacenada en la *Copia de trabajo*. No accede al repositorio para actualizarlos.
8. Elimina la lista de cambios de la *Copia de trabajo*. El usuario debera descargarlos de nuevo del repositorio en caso de que desee verlos.
9. Actualizar las capas del usuario de la *Copia de trabajo* con los cambios mostrados (update). Esta opcion no esta disponible si existen conflictos entre los cambios del repositorio y los cambios realizados en la *Copia de trabajo*.
10. Se mezclan o fusionan las capas del usuario de la *Copia de trabajo* con los cambios del repositorio. Se actualizan los datos que no tengan conflicto con la *Copia de trabajo*, y se marcan para actualizar en el repositorio los cambios realizados en la *Copia de trabajo*. Esta opcion mantiene las modificaciones de la *Copia de trabajo* frente a los cambios que se hayan realizado en el repositorio. Solo estara activa si se han detectado conflictos entre la *Copia de trabajo* y el repositorio.
11. Botón que muestra un folmulario con los datos del registro seleccionado.

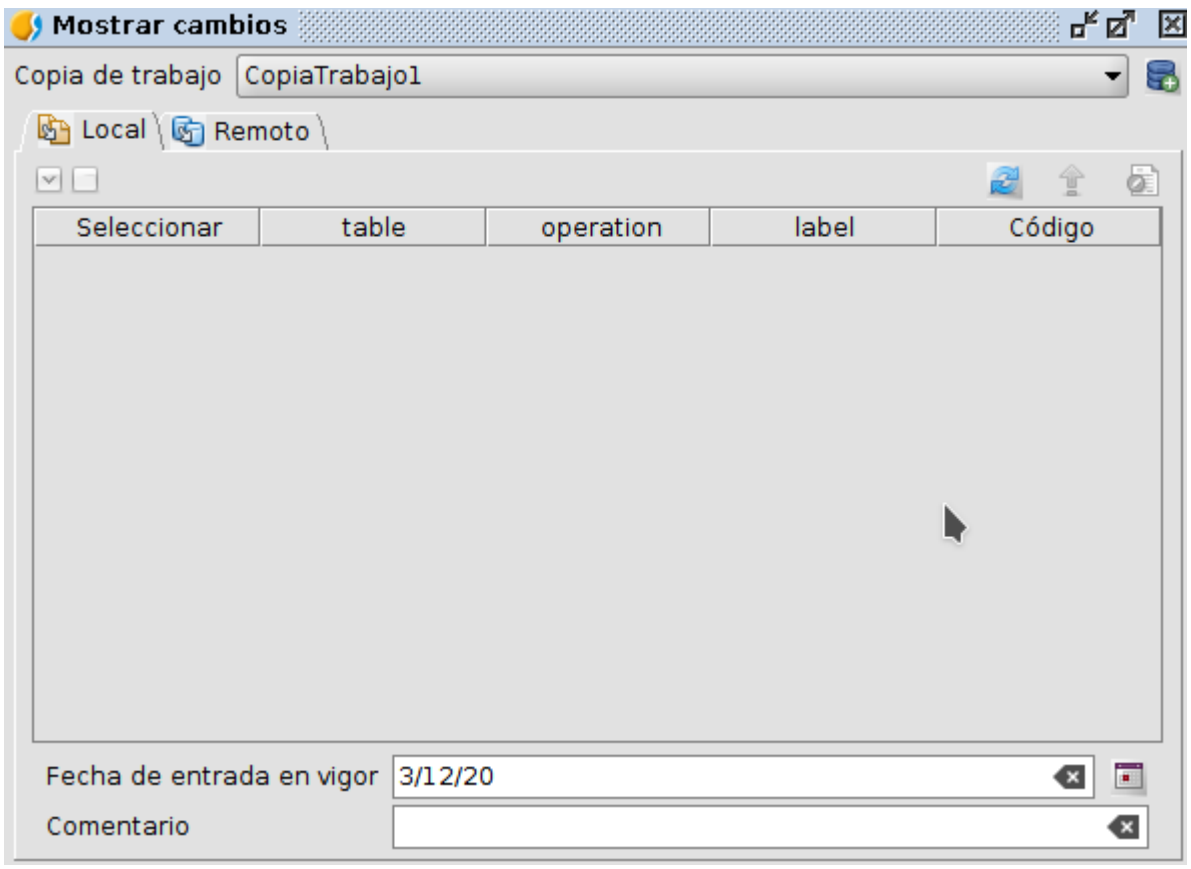
Esta opción de la ventana se explicará con mayor detalle en los apartados siguientes mediante ejemplos.

Volviendo al caso en cuestión tras ejecutar el comando *Mostrar cambios* y al pulsar la opción *Local* en el área destinada a estos aparecen registros correspondientes a los elementos de la capa a añadir, ver siguiente imagen. Esto se debe a que hay diferencias entre la copia de trabajo y el repositorio, presentando la copia de trabajo una serie de nuevos elementos, una capa, que el repositorio carece. El proceso de añadir una nueva capa al repositorio finaliza si seleccionamos esos

registros y pulsamos el botón que realiza un *commit*, componente 5 de la ventana. Con esa acción el repositorio se actualiza con la información de la copia de trabajo y tendría por tanto la nueva capa a su disposición.

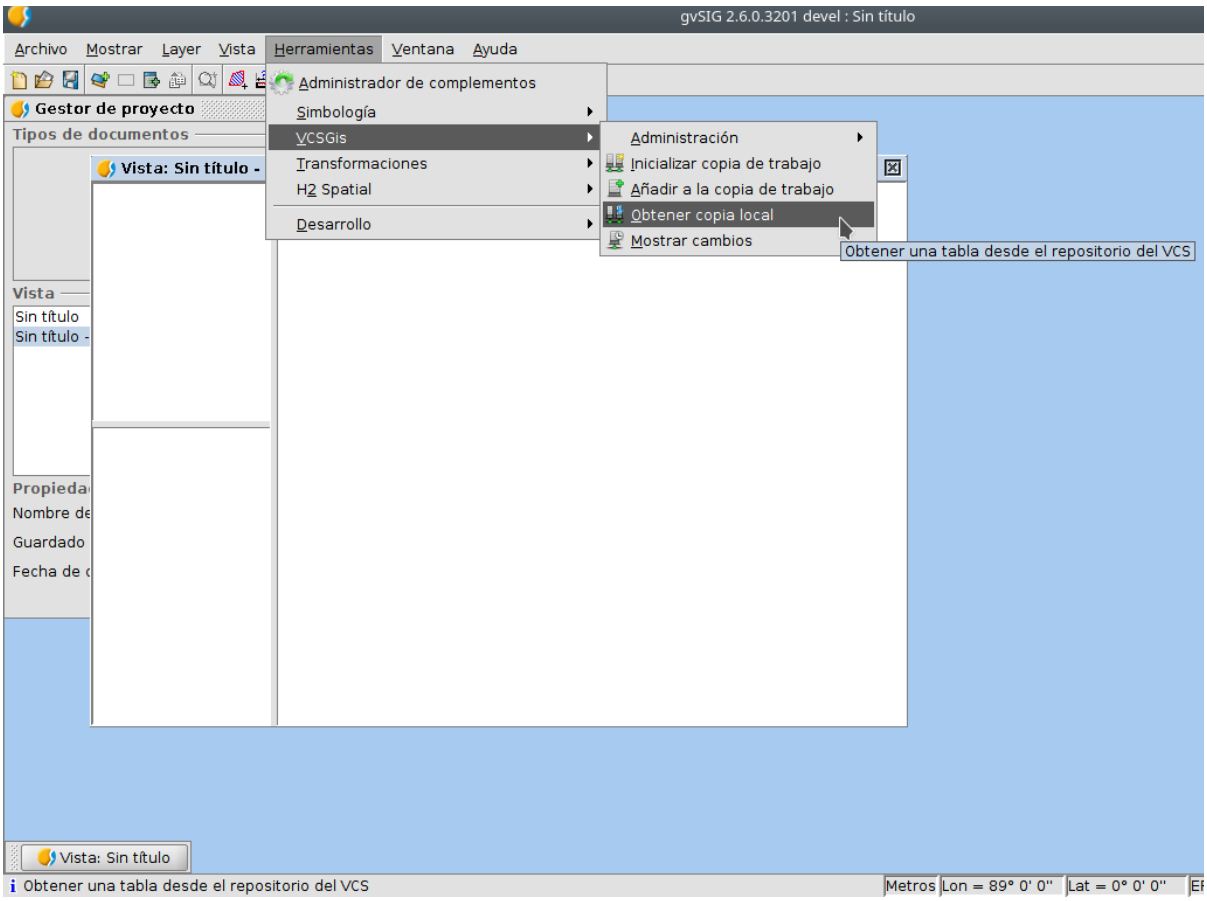


A modo de comprobación tras lo anterior si se selecciona la pestaña *Local* y se pulsa el botón de actualizar o refrescar área de visualización (componente 4 de la ventana), esta aparecerá vacía.

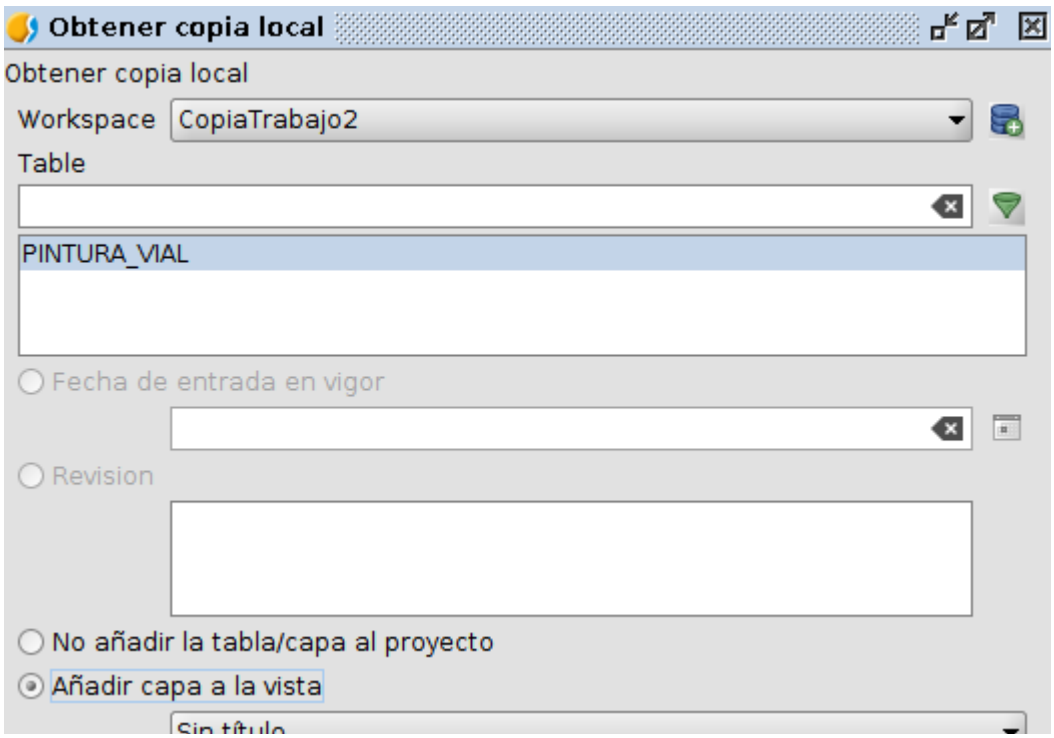


Añadir una capa del repositorio

El procedimiento de añadir una capa del repositorio a la copia de trabajo es muy similar al anterior y se realiza ejecutando el comando *Obtener copia local* (checkout) situado en la opción Herramientas del menú de *gvSIG Desktop*, pestaña *VCSGgis*.



Como resultado de la ejecución anterior se obtiene la siguiente ventana:



En la ventana hay que indicar la copia de trabajo a la que se quiere añadir la capa procedente del repositorio y la capa en cuestión, seleccionando esta de la lista presente. Además de lo anterior, la ventana presenta varias opciones, *No añadir la tabla/capa al proyecto*, *Añadir capa a la vista* y *Añadir la tabla al proyecto*.

Hacer incapie en una idea importante, si en el proceso de obtener una copia local, es decir una capa del repositorio no se marca la opción *Añadir capa a la vista*, seleccionando la vista deseada, la capa se almacena en la copia de trabajo pero no se representará en la vista. Llegado a este punto se tendría que cargar la capa mediante mediante el dialogó estandard de *Añadir capa* de gvSIG Desktop, seleccionando el tipo de conexion a base de datos y la conexion a la base de datos asociada a la *Copia de trabajo*.

> *NOTA: En proximas versiones de VCSGis se añadirá la posibilidad de cargar o añadir capas con control de versiones VCSGis desde el diálogo estandard Añadir capa de gvSIG Desktop.*

Ciclo de trabajo básico

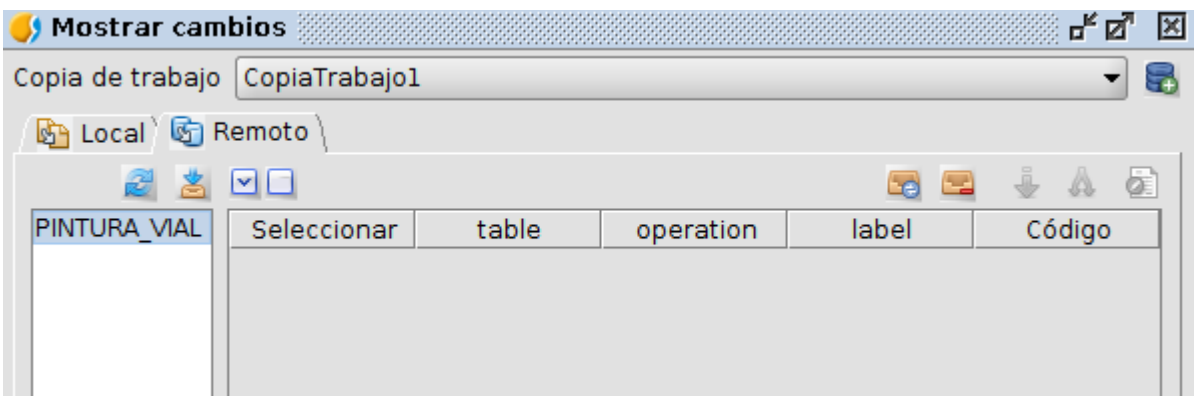
Una vez que tenemos una copia de trabajo con las tablas del repositorio que necesitemos ya podemos comenzar a trabajar. Los pasos típicos que se siguen cuando se trabaja el bajo control de versiones son:

- Actualizar la copia de trabajo (update).
- Realizar cambios. Es decir, trabajar con las capas normalmente.
- Revisar los cambios.
- Arreglar tus errores.
- Mezclar las capas con los cambios que se hayan realizado mientras trabajabas. Posiblemente haya que arreglar conflictos.
- Enviar los cambios al repositorio.

Y cada vez que se vuelve al trabajo, se repite este ciclo.

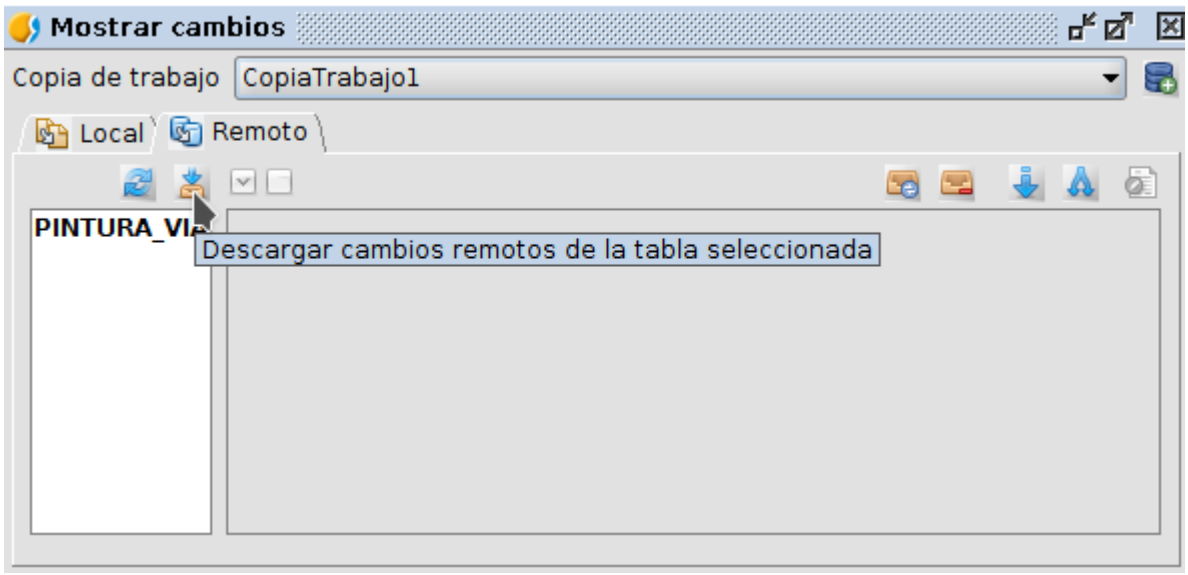
Actualizar la copia de trabajo

Para actualizar la copia de trabajo hay que ejecutar el comando *Mostrar cambios* y una vez en la ventana con el mismo nombre seleccionar la pestaña *Remoto*; quedando la ventana en cuestión como en la siguiente figura.

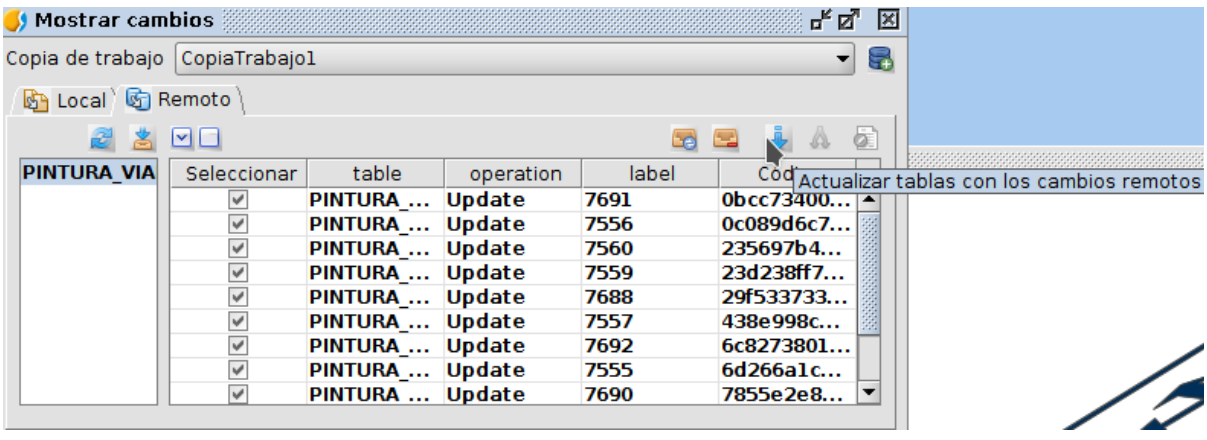


Si no existen cambios en el repositorio con respecto a la copia de trabajo, las capas de la lista de capas de la *Copia de trabajo* apareceran con una tipografía sin negrita en el componente 3 de la ventana (área con lista de capas de la *Copia de trabajo*) indicando que la *Copia de trabajo* está actualizada; pero si existen cambios, estas aparecerán en negrita estando la copia desactualizada.

Para actualizarla hay que seleccionar la capa en negrita y pulsar el botón para descargar los cambios o diferencias del repositorio frente a la *Copia de trabajo*, componente 5 de la ventana.



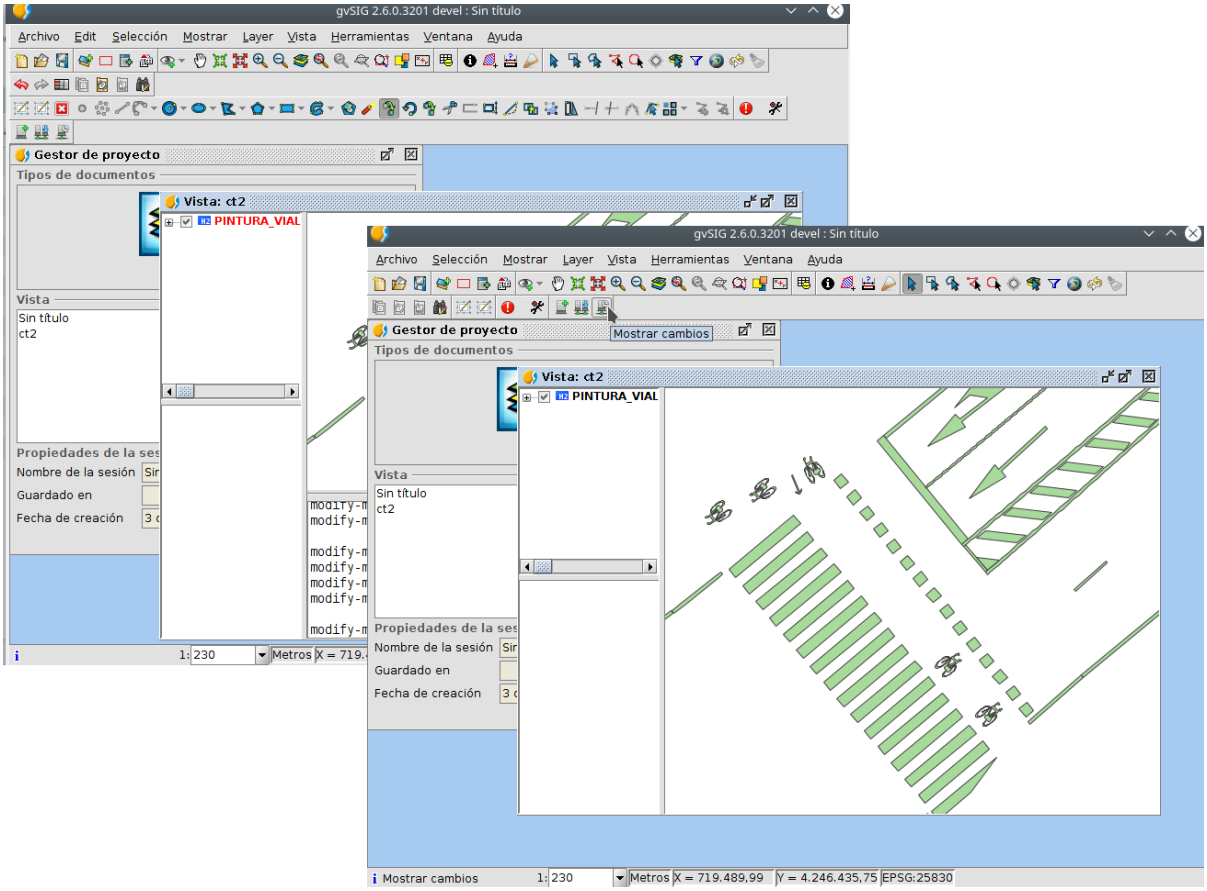
La acción anterior provocará que los cambios se descarguen del repositorio y se muestren en el área de visualización de cambios, componente 6. Estos cambios aun no se habran aplicaddo sobre las tablas del usuario. Para realizar esto deberan usarse los botones de **actualizar** (update) o **mezclar** (merge) cambios en la *Copia de trabajo*, componentes 9 y 10 de la ventana.



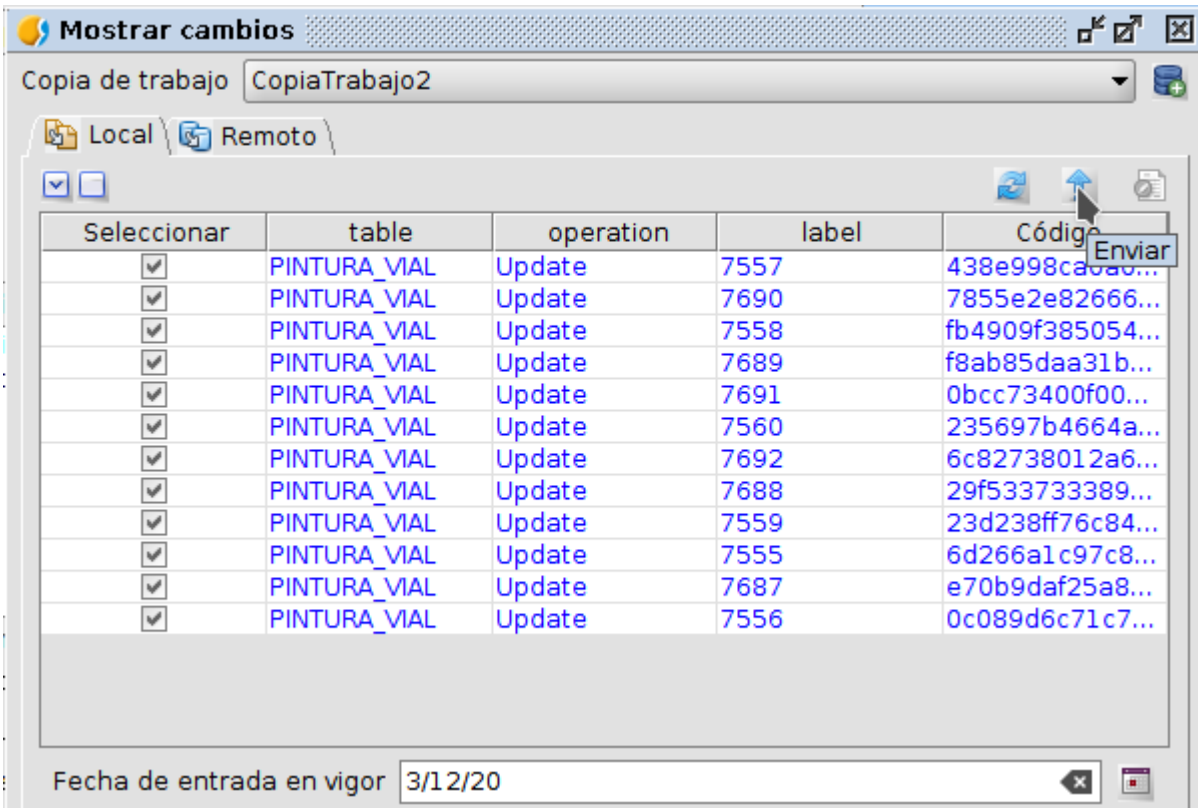
Revisando y enviando los cambios realizados

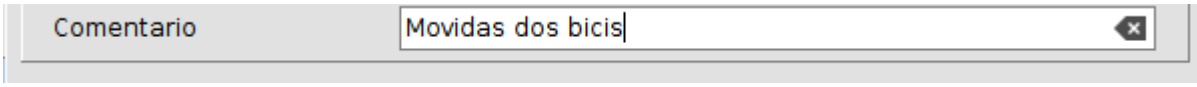
Para explicar mejor este comportamiento se va a usar el siguiente ejemplo:

Un usuario modifica una serie de elementos de una capa de su copia de trabajo llamada PINTURA_VIAL, tras terminar edición ejecuta el comando Mostrar cambios.



Como resultado al seleccionar la opción Local de la ventana Mostrar cambios, se muestran una serie de registros correspondientes a los citados cambios y solo tendría que pulsar el botón condirmación (commit) para actualizar esos cambios de su copia de trabajo en el repositorio.





Examinando la historia

TODO

Aun no disponible

Resolución de conflictos

TODO

Pendiente de realizar documentacion