

UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL RESISTENCIA
Ingeniería en Sistemas de Información

DACS

Trabajo Práctico 1

Docentes:

- Jorge Eduardo Villaverde
- Fabricio Quevedo

Integrantes

- ***Campuzano, Cinthia- Cinthialujancampuzano@gmail.com***
- ***Kugler, Benjamin - benja_996@hotmail.com***
- ***Marquez, Valeria - Marvalej5@gmail.com***
- ***Ygarzabal, Gustavo – Ygarzabal.gustavo@gmail.com***

CICLO LECTIVO 2021

Índice

Consignas.....	1
Actividad 1: Informe de investigación de Sistema de Control de Versiones	1
Actividad 2: Análisis y utilización de un Sistema de Control de Versiones Centralizado	1
Actividad 3: Actividad práctica sobre Git y Github.....	2
Actividad 1- Informe de investigación de Sistema de Control de Versiones	3
Git	3
Mercurial	4
BitKeeper	5
Subversion	6
CVS.....	7
Extras.....	8
a. Sistemas de control de versiones disponibles en el mercado.....	8
b. Plataformas comerciales de sistemas de control de versiones.....	8
Actividad 2: Análisis y utilización de un Sistema de Control de Versiones Centralizado.....	10
Apache Subversion	10
Ventajas y desventajas. Comparación con SCV Descentralizados.....	12
Capturas de commits de branch y merge.	13
Actividad 3: Actividad práctica sobre Git y Github	23
Bibliografía.....	25

Consignas

Actividad 1: Informe de investigación de Sistema de Control de Versiones

- Investigar y elaborar un breve informe de sistemas de control de versiones disponibles en el mercado, tanto del tipo centralizado como descentralizado (entre 5 y 8, ejemplo git, mercurial, svn, cvs, bitkeeper, etc). Indicar los siguientes ítems:
 1. Tipos de versionado soportados.
 2. Licencia
 3. Costo (gratuito / propietario)
 4. Quien lo mantiene.
 5. Plataformas soportadas (Windows, Unix, etc)
 6. Extras
 - Elaborar un cuadro comparativo que resuma los puntos antes mencionados
 - Realizar el mismo cuadro con plataformas comerciales de sistemas de control de versiones (entre 5 y 8, por ejemplo, Atlassian, Github, etc) agregando la columna sistemas de control de versiones que soporta mencionadas en el punto anterior. Además, mencionar que herramientas adicionales incluyen (por ejemplo, wiki, herramientas de gestión de proyectos, etc).

Actividad 2: Análisis y utilización de un Sistema de Control de Versiones Centralizado

- Investigar un SCV Centralizado y explicar las principales características brevemente.
- Enumerar ventajas y desventajas, y comparación con SCV Descentralizados (cuadro comparativo).
- Seleccionar un servidor que se encuentre en la nube/web gratuito para realizar un ejemplo.
- Realizar un ejemplo iniciar el repositorio, clonarlo, modificarlo y generar conflictos, crear ramas y realizar merge de las mismas con el trunk principal, en un pequeño equipo por lo menos 3 miembros del grupo.
- Utilizar de ser necesario una herramienta cliente (gráfico o consola) o IDE.
- Documentar el ejemplo con capturas de commits de los miembros del equipo sobre un mismo archivo y otro ejemplo de branch y merge.

Actividad 3: Actividad práctica sobre Git y Github

Utilizando Git por línea de comandos o desde la Web de Github (según corresponda) realizar el siguiente ejercicio (ir evidenciando documentando los pasos ver nota al final):

1. Un miembro del equipo va a clonar el siguiente repositorio y va a crear una rama para el grupo (la misma va a tener la forma GX/principal donde X es el número de grupo).
2. En su repositorio local el usuario va a crear un va a crear una carpeta de grupo (grupoX) y dentro de la misma va a crear un proyecto en Node.js. Commitear los cambios en el repositorio y subir la rama al servidor remoto. Les dejo un link de ayuda:
 - Link 1
 - Link 2
3. Una vez creada la rama del grupo en el servidor uno de los miembros del grupo va a hacer un fork de la rama. Clona el fork, va a insertar una función que imprime en un label una entrada de pantalla, commit.> push y pull request al repositorio del grupo.
4. Los demás miembros del grupo: Clonar el repositorio y toman la rama del grupo. A partir de la rama del grupo, crean una rama personal (gXiniciales grupo X e 2 iniciales) donde realizar una modificación en código (insertar una función que transforme el formato de un texto, que calcule una suma y la muestre en pantalla, etc) y realizar un commit y push, (Generar un conflicto y resolverlo). Ponerse de acuerdo en el grupo.
5. Realizar un pull request de la rama personal a la principal de grupo.
6. Aceptar / confirmar los pull request en la web, obtener a la funcionalidad completa del programa. Generar un tag para la versión con el nombre gX-V-1.0.0 X número de grupo (por línea de comando) y subir al repositorio remoto.
7. Realizar un cambio en el programa sobre la rama principal del grupo y subir el cambio (que introduce un error al programa).
8. Crear una rama a partir del tag creado y subir la rama al repo remoto y crear un pull request a la rama principal.
9. Aceptar / Confirmar el pull request creado en el paso anterior (corregir el error).

Actividad 1- Informe de investigación de Sistema de Control de Versiones

Un control de versiones es un sistema que registra los cambios realizados en un archivo o en un conjunto de archivos a lo largo del tiempo de modo que puedas recuperar versiones específicas más adelante. Por ejemplo, se usan archivos de código fuente como aquellos cuya versión está siendo controlada, en realidad puedes hacer lo mismo con casi cualquier tipo de archivo que encuentres en una computadora.

Si eres diseñador gráfico o de web y quieres mantener cada versión de una imagen o layout, usar un sistema de control de versiones es una decisión muy acertada. Dicho sistema te permite regresar a versiones anteriores de tus archivos seleccionados, regresar el proyecto entero a su versión anterior, comparar cambios a lo largo del tiempo, ver quién modificó por última vez algo que pueda estar causando problemas, ver quién introdujo un problema y cuándo, y mucho más. Usar un VCS también significa generalmente que, si arruinas o pierdes archivos, será posible recuperarlos fácilmente. Adicionalmente, obtendrás todos estos beneficios a un costo muy bajo.

Git

Git es un sistema de control de versiones distribuido de código abierto y gratuito diseñado para manejar todo, desde proyectos pequeños a muy grandes, con velocidad y eficiencia.

Ramificación y fusión

La característica de Git que realmente lo distingue de casi todos los demás SCM es su modelo de ramificación.

Git te permite y te anima a tener múltiples sucursales locales que pueden ser completamente independientes entre sí. La creación, fusión y eliminación de esas líneas de desarrollo lleva unos segundos.

Esto significa que puede hacer cosas como:

- Cambio de contexto sin fricciones. Cree una rama para probar una idea, confíe varias veces, vuelva al lugar desde donde se ramificó, aplique un parche, vuelva al lugar donde está experimentando y combínelo.
- Líneas de código basadas en roles. Tenga una rama que siempre contenga solo lo que va a producción, otra en la que combine el trabajo para realizar pruebas y varias más pequeñas para el trabajo diario.
- Flujo de trabajo basado en funciones. Cree nuevas ramas para cada nueva característica en la que esté trabajando para que pueda alternar sin problemas entre ellas, luego elimine cada rama cuando esa característica se fusione en su línea principal.
- Experimentación desechable. Crea una rama para experimentar, date cuenta de que no va a funcionar y simplemente bórrala, abandonando el trabajo, sin que nadie más la vea (incluso si has empujado otras ramas mientras tanto).

En particular, cuando empuja a un repositorio remoto, no tiene que empujar todas sus ramas. Puede optar por compartir solo una de sus sucursales, algunas de ellas o todas. Esto tiende a liberar a las personas para que prueben nuevas ideas sin preocuparse por tener que planificar cómo y cuándo van a fusionarlas o compartirlas con otros.

Área de ensayo

Git tiene algo llamado "área de preparación" o "índice". Esta es un área intermedia donde las confirmaciones se pueden formatear y revisar antes de completar la confirmación.

Con Git es posible preparar rápidamente algunos de sus archivos y enviarlos sin confirmar todos los demás archivos modificados en su directorio de trabajo o tener que listarlos en la línea de comandos durante la confirmación.

Pequeño y rápido

Git es rápido. Con Git, casi todas las operaciones se realizan localmente, lo que le da una gran ventaja de velocidad en sistemas centralizados que constantemente tienen que comunicarse con un servidor en algún lugar.

Git fue creado para funcionar en el kernel de Linux, lo que significa que ha tenido que manejar de forma eficaz grandes repositorios desde el primer día. Git está escrito en C, lo que reduce la sobrecarga de los tiempos de ejecución asociados con lenguajes de nivel superior. La velocidad y el rendimiento han sido un objetivo de diseño principal de Git desde el principio.

Repartido

Una de las mejores características de cualquier SCM distribuido, incluido Git, es que está distribuido. Esto significa que en lugar de hacer un "checkout" del consejo actual del código fuente, haces un "clon" de todo el repositorio.

Aseguramiento de datos

El modelo de datos que utiliza Git garantiza la integridad criptográfica de cada parte de su proyecto. Cada archivo y confirmación se suma de verificación y se recupera mediante su suma de verificación cuando se vuelve a verificar. Es imposible obtener algo de Git que no sean los bits exactos que ingresó.

También es imposible cambiar cualquier archivo, fecha, mensaje de confirmación o cualquier otro dato en un repositorio de Git sin cambiar las ID de todo lo que sigue. Esto significa que si tiene un ID de confirmación, puede estar seguro no solo de que su proyecto es exactamente el mismo que cuando se confirmó, sino de que no se cambió nada en su historial.

Gratis y de código abierto

Git se publica bajo la GNU General Public License versión 2.0, que es una licencia de código abierto. El proyecto Git eligió utilizar GPLv2 para garantizar su libertad de compartir y cambiar el software libre, para asegurarse de que el software sea gratuito para todos sus usuarios.

Sin embargo, restringimos el uso del término "Git" y los logotipos para evitar confusiones. Consulte nuestra política de marcas comerciales para obtener más detalles.

Mantenimiento

El mantenimiento del software Git está actualmente (2009) supervisado por Junio Hamano, quien recibe contribuciones al código de alrededor de 280 programadores.

Plataformas soportadas

POSIX, Windows, macOS

Mercurial

Mercurial es una herramienta de gestión de control de fuente distribuida y gratuita. Le ofrece el poder de manejar proyectos de cualquier tamaño de manera eficiente mientras usa una interfaz intuitiva. Es fácil de usar y difícil de romper, lo que lo hace ideal para cualquiera que trabaje con archivos versionados.

Arquitectura distribuida

Mercurial se distribuye verdaderamente, lo que le da a cada desarrollador una copia local de todo el historial de desarrollo. De esta forma funciona independientemente del acceso a la red o de un servidor central. Comprometerse, ramificarse y fusionarse son rápidos y económicos.

Rápido

La implementación y las estructuras de datos de Mercurial están diseñadas para ser rápidas. Puede generar diferencias entre revisiones o retroceder en el tiempo en segundos. Por lo tanto, Mercurial es perfectamente adecuado para proyectos grandes como nginx (hg) o NetBeans (hg).

Plataforma independiente

Mercurial se escribió pensando en la independencia de la plataforma. Por lo tanto, la mayor parte de Mercurial está escrito en Python, con una pequeña parte en C portátil por razones de rendimiento. Como resultado, las versiones binarias están disponibles en todas las plataformas principales.

Extensible

La funcionalidad de Mercurial se puede aumentar con extensiones, ya sea activando las oficiales que se envían con Mercurial o descargando algunas de la wiki o escribiendo las suyas propias. Las extensiones están escritas en Python y pueden cambiar el funcionamiento de los comandos básicos, agregar nuevos comandos y acceder a todas las funciones básicas de Mercurial.

Fuente abierta

Mercurial es un software gratuito con licencia según los términos de la Licencia Pública General GNU Versión 2 o cualquier versión posterior.

El sitio web

El sitio web es un proyecto de la comunidad Mercurial. La fuente tiene licencia GPLv2 o posterior. No dude en enviarnos parches.

Mantenimiento

El creador y desarrollador principal de Mercurial es [Matt Mackall](#).

Plataformas soportadas

Mercurial fue escrito originalmente para funcionar sobre GNU/Linux. Ha sido adaptado para Windows, Mac OS X y la mayoría de otros sistemas tipo Unix.

BitKeeper

BitKeeper es el sistema de gestión de fuentes distribuidas original. Ahora disponible como código abierto bajo la licencia Apache 2.0. BitKeeper es un SCM distribuido, rápido y listo para la empresa, que escala hasta proyectos muy grandes y hasta pequeños.

Simple

Una interfaz de línea de comandos fácil de usar.

Escalable

Los repositorios anidados son submódulos bien hechos. Colecciones de repositorios de control de versiones.

Flexible

Modo híbrido para archivos binarios que utiliza una nube de servidor para binarios en lugar de inflar los repositorios de origen.

Preciso

Seguimiento de operaciones de archivos como creación, eliminación y cambio de nombre.

Seguro

Todos los accesos a archivos validan las sumas de comprobación de integridad. Todas las escrituras de archivos incluyen redundancia para la corrección de errores.

Confiable

Fusión automática de alta precisión que utiliza todo el historial para resolver conflictos. La mayoría de los otros sistemas utilizan variaciones de diff3.

Rápido

Alto rendimiento y escala a repositorios muy grandes.

Gratis

Con licencia de Apache Versión 2

Mantenimiento

BitKeeper es producido por Bitmover Inc., una compañía privada con sede en Campbell (California), propiedad del CEO Larry McVoy.

Plataformas soportadas

Unix-like, Windows, macOS

Subversion

Subversion es un sistema de control de versiones libre bajo una licencia de tipo Apache/BSD y de código fuente abierto. Es decir, Subversion maneja ficheros y directorios a través del tiempo. Hay un árbol de ficheros en un repositorio central. El repositorio es como un servidor de ficheros ordinario, excepto porque recuerda todos los cambios hechos a sus ficheros y directorios. Esto le permite recuperar versiones antiguas de sus datos, o examinar el historial de cambios de los mismos. En este aspecto, mucha gente piensa en los sistemas de versiones como en una especie de “máquina del tiempo”. Subversion puede acceder al repositorio a través de redes, lo que le permite ser usado por personas que se encuentran en distintos ordenadores. A cierto nivel, la capacidad para que varias personas puedan modificar y administrar el mismo conjunto de datos desde sus respectivas ubicaciones fomenta la colaboración. Se puede progresar más rápidamente sin un único conducto por el cual deban pasar todas las modificaciones. Y puesto que el trabajo se encuentra bajo el control de versiones, no hay razón para temer por que la calidad del mismo vaya a verse afectada por la pérdida de ese conducto único—si se ha hecho un cambio incorrecto a los datos, simplemente deshaga ese cambio. Algunos sistemas de control de versiones son también sistemas de administración de configuración de software. Estos sistemas son diseñados

específicamente para la administración de árboles de código fuente, y tienen muchas características que son específicas del desarrollo de software— tales como el entendimiento nativo de lenguajes de programación, o el suministro de herramientas para la construcción de software. Sin embargo, Subversion no es uno de estos sistemas. Subversion es un sistema general que puede ser usado para administrar cualquier conjunto de ficheros. Para usted, esos ficheros pueden ser código fuente— para otros, cualquier cosa desde la lista de la compra de comestibles hasta combinaciones de vídeo digital y más allá.

Las plataformas soportadas son: Unix-like, Windows, macOS

CVS

CVS es un sistema de control de versiones, un componente importante de Source Configuration Management (SCM). Utilizándolo, puede registrar el historial de archivos de origen y documentos. Cumple una función similar a los paquetes de software libre RCS, PRCS y Aegis.

CVS es un sistema de calidad de producción de amplio uso en todo el mundo, incluidos muchos proyectos de software libre.

CVS almacena el historial de archivos individuales en el mismo formato que RCS.

Puede ejecutar scripts que puede proporcionar para registrar las operaciones de CVS o hacer cumplir las políticas específicas del sitio.

El CVS cliente / servidor permite a los desarrolladores dispersos por geografía o módems lentos funcionar como un solo equipo. El historial de versiones se almacena en un único servidor central y las máquinas cliente tienen una copia de todos los archivos en los que están trabajando los desarrolladores. Por lo tanto, la red entre el cliente y el servidor debe estar habilitada para realizar operaciones CVS (como registros o actualizaciones) pero no es necesario que esté habilitada para editar o manipular las versiones actuales de los archivos. Los clientes pueden realizar las mismas operaciones que están disponibles localmente.

En los casos en que varios desarrolladores o equipos quieran que cada uno mantenga su propia versión de los archivos, debido a la geografía y / o la política, las sucursales de proveedores de CVS pueden importar una versión de otro equipo (incluso si no usan CVS) y entonces CVS puede fusionar los cambios de la rama del proveedor con los archivos más recientes si eso es lo que se desea.

Pagos sin reservas, lo que permite que más de un desarrollador trabaje en los mismos archivos al mismo tiempo.

CVS proporciona una base de datos de módulos flexible que proporciona un mapeo simbólico de nombres a componentes de una distribución de software más grande. Aplica nombres a colecciones de directorios y archivos. Un solo comando puede manipular toda la colección.

Los servidores CVS se ejecutan en la mayoría de las variantes de Unix y también están disponibles clientes para Windows NT / 95, OS / 2 y VMS. CVS también funcionará en lo que a veces se denomina modo servidor contra repositorios locales en Windows 95 / NT.

Su desarrollador es The CVS Team. Sus desarrolladores difunden el sistema bajo la licencia GPL.

Extras

a. Sistemas de control de versiones disponibles en el mercado

CVS	Tipo de Versionado	Licencia	Costo	Quien lo Mantiene	Plataformas Soportadas
Git	Distribuido	GNU GPL	Free	Junio Hamano	POSIX, Windows, macOS
Mercurial	Distribuido	GNU GPL	Free	Matt Mackall	Unix-like, Windows, macOS
SVN	Centralizado	Apache	Free	Apache Software Foundation	Unix-like, Windows, macOS
CVS	Centralizado	GNU GPL	Free	The CVS Team	Unix-like, Windows, macOS
BitKeeper	Distribuido	Apache	Free	BitMover Inc	Unix-like, Windows, macOS

POSIX, estándar definido por la IEEE que asegura la compatibilidad con variantes de UNIX.

b. Plataformas comerciales de sistemas de control de versiones

CVS	Tipo de Versionado	Licencia	Costo	Quien lo Mantiene	Plataformas Soportadas	CVS Soportado	Herramientas adicionales
GitHub	Distribuido	GNU GPL	Free	Microsoft/ GitHub. Inc	POSIX, Windows, macOS	Git, SVN (parcial)	Automation and CI/CD, GitHub Pages, Code review Private repos , Client Apps, Project Management, Team Administration , Comunidad
Bitbucket	Distribuido	GNU GPL	Free	Atlassian	Unix-like, Windows, macOS	Git	Automation and CI/CD, Code review, Jira and Trello integration, Wiki
GitLab	Centralizado	Apache	Free	GitLab Inc.	Unix-like, Windows, macOS	Git	Automation and CI/CD, GitLab Pages, Wiki
Launchpad	Centralizado	GNU GPL	Free	Canonical	Unix-like, Windows, macOS	Bazaar	Code Review, Bug Tracking, Private Repository, Launchpad, Community Forum

SourceForge	Distribuido	Apache	Free	BizX LLC	Unix-like, Windows, macOS	Git, SVN, Mercurial	Code review, Bug tracking, Web hosting, Wiki
--------------------	-------------	--------	-------------	-----------------	------------------------------	--------------------------------	--

Actividad 2: Análisis y utilización de un Sistema de Control de Versiones Centralizado

Apache Subversion

Principales características:

Los directorios están versionados.

Subversion versiona directorios como objetos de primera clase, como archivos.

La copia, eliminación y cambio de nombre tienen versiones.

Copiar y eliminar son operaciones versionadas. El cambio de nombre también es una operación versionada.

Metadatos versionados de formato libre ("propiedades").

Subversion permite adjuntar metadatos arbitrarios ("propiedades") a cualquier archivo o directorio. Estas propiedades son pares clave / valor y están versionadas al igual que los objetos a los que están adjuntas.

Atomic confirma.

Ninguna parte de una confirmación entra en vigor hasta que la confirmación completa se haya realizado correctamente. Los números de revisión son por confirmación, no por archivo, y el mensaje de registro de la confirmación se adjunta a su revisión, no se almacena de forma redundante en todos los archivos afectados por esa confirmación.

Bloqueo de archivos.

Subversion admite (pero no requiere) archivos de bloqueo para que los usuarios puedan ser advertidos cuando varias personas intentan editar el mismo archivo. Un archivo puede marcarse como que requiere un bloqueo antes de ser editado, en cuyo caso Subversion presentará el archivo en modo de solo lectura hasta que se obtenga un bloqueo.

Se conserva la bandera ejecutable.

Subversion advierte cuando un archivo es ejecutable, y si ese archivo se coloca en el control de versiones, su ejecución se conservará cuando se desproteja en otras ubicaciones.

Opción de servidor de red Apache, con protocolo WebDAV / DeltaV.

Subversion puede utilizar el protocolo WebDAV / DeltaV basado en HTTP para comunicaciones de red y el servidor web Apache para proporcionar servicio de red del lado del repositorio. Esto le da a Subversion una ventaja sobre CVS en interoperabilidad y permite que ciertas características (como autenticación, compresión por cable) se proporcionen de una manera que ya es familiar para los administradores.

Opción de servidor independiente (svnserve).

Subversion ofrece una opción de servidor independiente que utiliza un protocolo personalizado, ya que no todo el mundo quiere ejecutar un servidor Apache HTTPD. El servidor independiente puede ejecutarse como un servicio inetd o en modo demonio, y ofrece el mismo nivel de funcionalidad de autenticación y autorización que el servidor basado en HTTPD. El servidor autónomo también se puede tunelizar a través de ssh.

Salida analizable.

Toda la salida del cliente de línea de comandos de Subversion está cuidadosamente diseñada para ser legible por humanos y automáticamente analizable; la capacidad de escritura es una alta prioridad.

Mensajes localizados.

Subversion usa gettext () para mostrar mensajes de error, informativos y de ayuda traducidos, basados en la configuración regional actual.

Resolución interactiva de conflictos.

El cliente de línea de comandos de Subversion (svn) ofrece varias formas de resolver cambios conflictivos, incluida la solicitud de resolución interactiva. Este mecanismo también está disponible a través de API, para que otros clientes (como los clientes gráficos) puedan ofrecer una resolución interactiva de conflictos adecuada a sus interfaces.

Duplicación de solo lectura del repositorio.

Subversion proporciona una utilidad, svnsync, para sincronizar (mediante push o pull) un repositorio esclavo de solo lectura con un repositorio maestro.

De forma nativa cliente / servidor, diseño de biblioteca en capas con API limpias.

Subversion está diseñado para ser cliente / servidor desde el principio; evitando así algunos de los problemas de mantenimiento que han afectado a CVS. El código está estructurado como un conjunto de módulos con interfaces bien definidas, diseñado para ser llamado por otras aplicaciones.

Los archivos binarios se manejan de manera eficiente.

Subversion es igualmente eficiente en archivos binarios que en archivos de texto, porque usa un algoritmo de diferenciación binaria para transmitir y almacenar revisiones sucesivas.

Los costos son proporcionales al tamaño del cambio, no al tamaño de los datos.

En general, el tiempo requerido para una operación de Subversion es proporcional al tamaño de los cambios resultantes de esa operación, no al tamaño absoluto del proyecto en el que se están produciendo los cambios.

Enlaces a lenguajes de programación.

Las API de Subversion vienen con enlaces para muchos lenguajes de programación, como Python, Perl, Java y Ruby. (Subversion en sí está escrito en C.)

Ventajas y desventajas. Comparación con SCV Descentralizados.

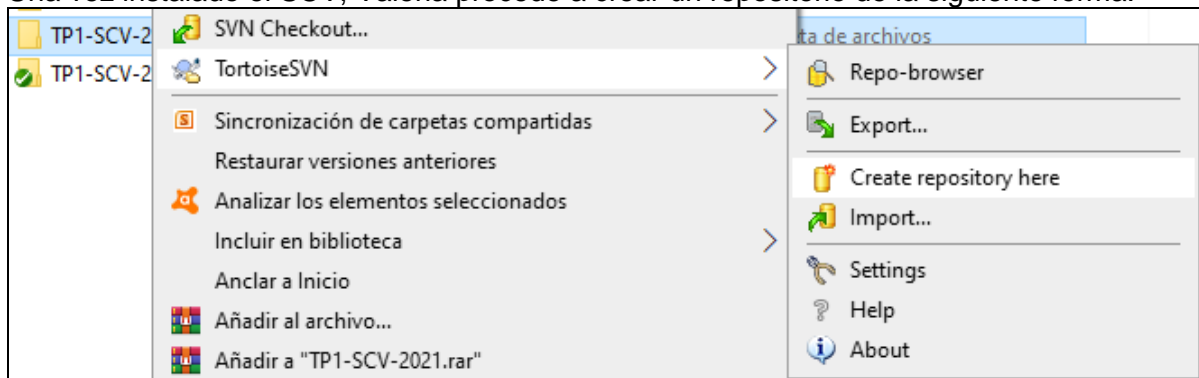
	SCV Centralizados	SCV Descentralizados
VENTAJAS	<ol style="list-style-type: none"> 1. Es posible colaborar con otros desarrolladores en otros sistemas. 2. Creación y mantenimiento centralizado de repositorios. 3. Fácil creación y organización de las carpetas de ficheros. 4. Creación de copias locales de todo o parte del repositorio, que nos permite trabajar de forma independiente en cada PC con la información del repositorio. 5. Sistemas de sincronización y coordinación entre los diferentes miembros del grupo u organización, para que el trabajo de un usuario no perjudique (al menos sin avisar) al trabajo de otro usuario. 6. Fácil creación de copias de seguridad y recuperación de la información, incluso a una fecha dada. 7. Visualización del histórico de cambios en el repositorio, con identificación del usuario que realiza cada cambio. 	<ol style="list-style-type: none"> 1. No es necesario estar conectado para guardar cambios. 2. Posibilidad de continuar trabajando si el repositorio remoto no está accesible. 3. El repositorio central está más libre de ramas de pruebas. 4. Se necesitan menos recursos para el repositorio remoto. 5. Más flexibles al permitir gestionar cada repositorio personal como se quiera. 6. Dado que cada programador tiene una copia completa del repositorio del proyecto, pueden compartir los cambios entre sí si fuera necesario obtener un feedback antes de persistir los cambios en el repositorio principal. 7. Si el servidor central sufre algún percance en algún momento, los datos perdidos pueden ser recuperados fácilmente desde cualquiera de los repositorios locales de los colaboradores.
DESVENTAJAS	<ol style="list-style-type: none"> 1. No se puede seguir trabajando si el servidor se cae. 2. No está disponible localmente, por lo que siempre es necesario estar conectado a una red para realizar cualquier acción. 3. Como todo está centralizado, si el servidor se 	<ol style="list-style-type: none"> 1. Necesita múltiples ramas para admitir desarrollos paralelos utilizados por los desarrolladores. 2. No hay control de acceso integrado y no admite archivos binarios. 3. No proporcionan mecanismos de control de acceso en caso de seguridad.

	rompe provocará la pérdida completa del proyecto.	
--	--	--

Capturas de commits de branch y merge.

Creación de repositorio:

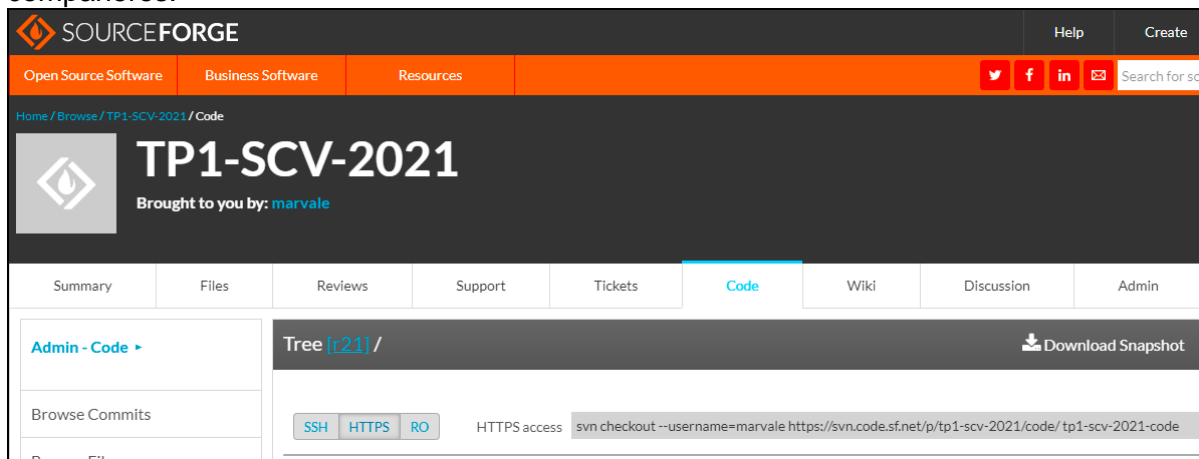
Una vez instalado el SCV, Valeria procede a crear un repositorio de la siguiente forma:



Se crean automáticamente las siguientes carpetas y archivos:

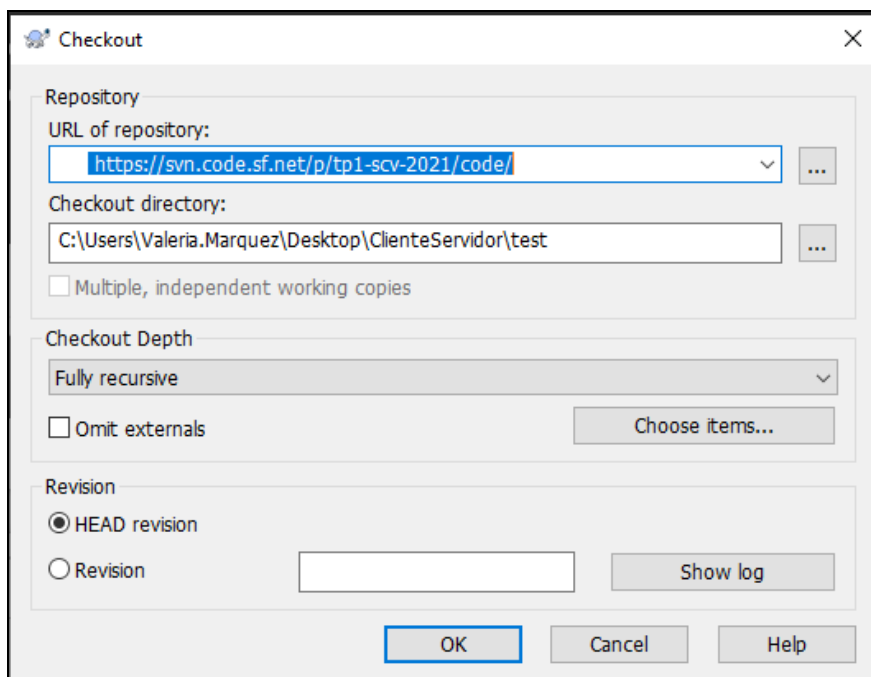
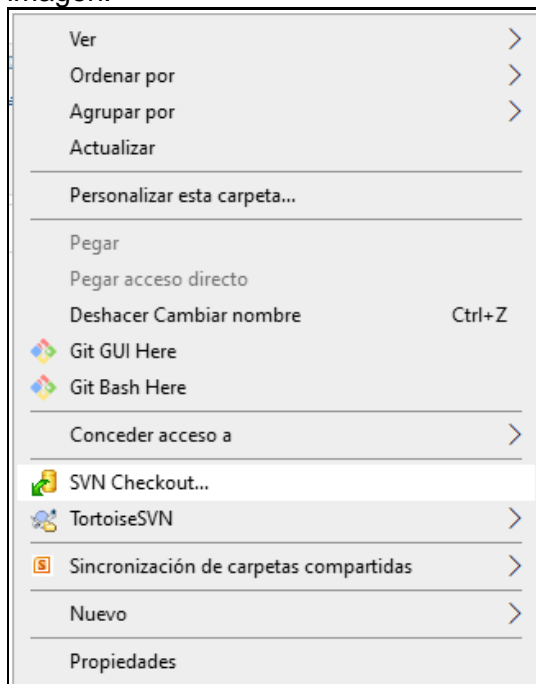
code	14/4/2021 19:03	Carpeta de archivos	
conf	14/4/2021 18:41	Carpeta de archivos	
db	14/4/2021 18:41	Carpeta de archivos	
hooks	14/4/2021 18:41	Carpeta de archivos	
locks	14/4/2021 18:41	Carpeta de archivos	
Desktop.ini	14/4/2021 18:41	Opciones de confi...	1 KB
format	14/4/2021 18:41	Archivo	1 KB
README.txt	14/4/2021 19:03	Documento de te...	1 KB
svn.ico	14/4/2021 18:41	Icono	177 KB
test2.txt	14/4/2021 19:03	Documento de te...	0 KB

Con la ayuda del sitio web SourceForge se habilita la colaboración del proyecto para con los compañeros:

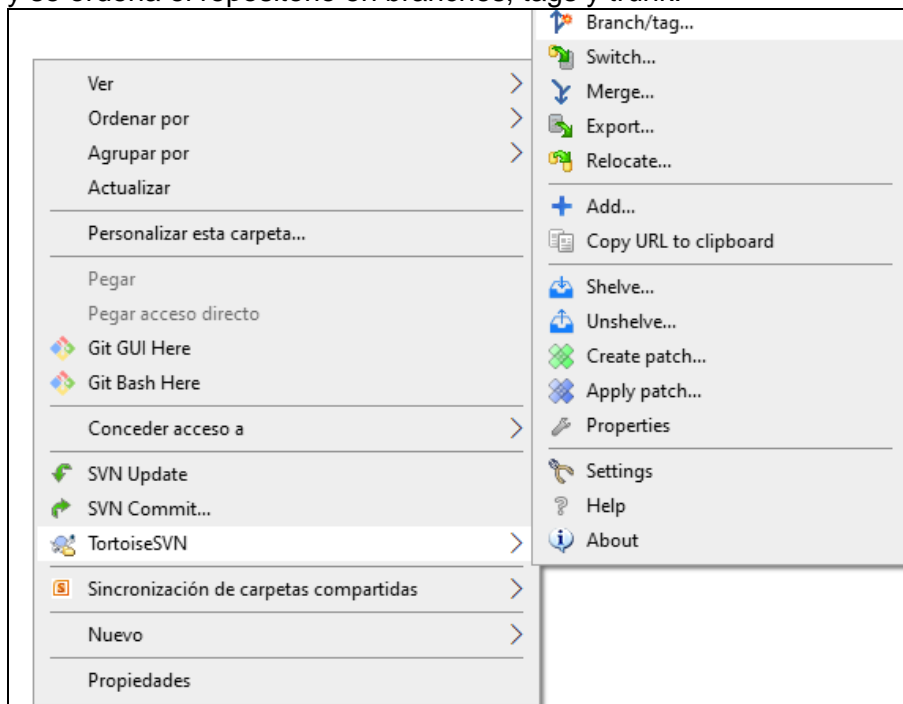





Group	Users	Permissions
Admin	<ul style="list-style-type: none"> Valeria Marquez (marvale) Add 	<ul style="list-style-type: none"> update admin create
Developer	All users in Admin group <ul style="list-style-type: none"> gustavo (ygarzabalgustav) Cinthia Lujan Campuzano (cinthia05) Add 	<ul style="list-style-type: none"> update admin create

Como única vez al comenzar el proyecto se va a hacer un checkout, y en el “URL of repository” vamos a pegar el URL que proporciona el sitio web SourceForge que mostramos en la primera imagen.



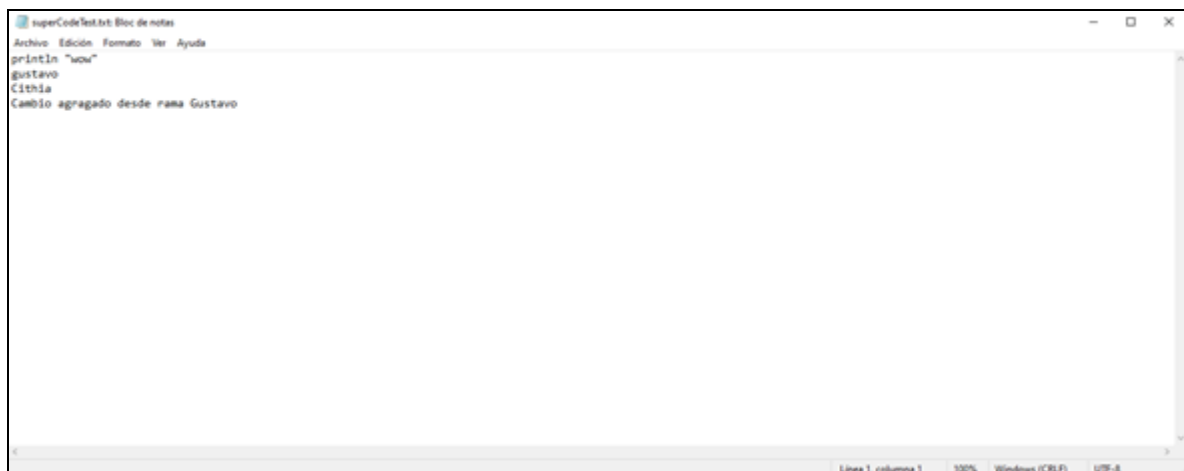
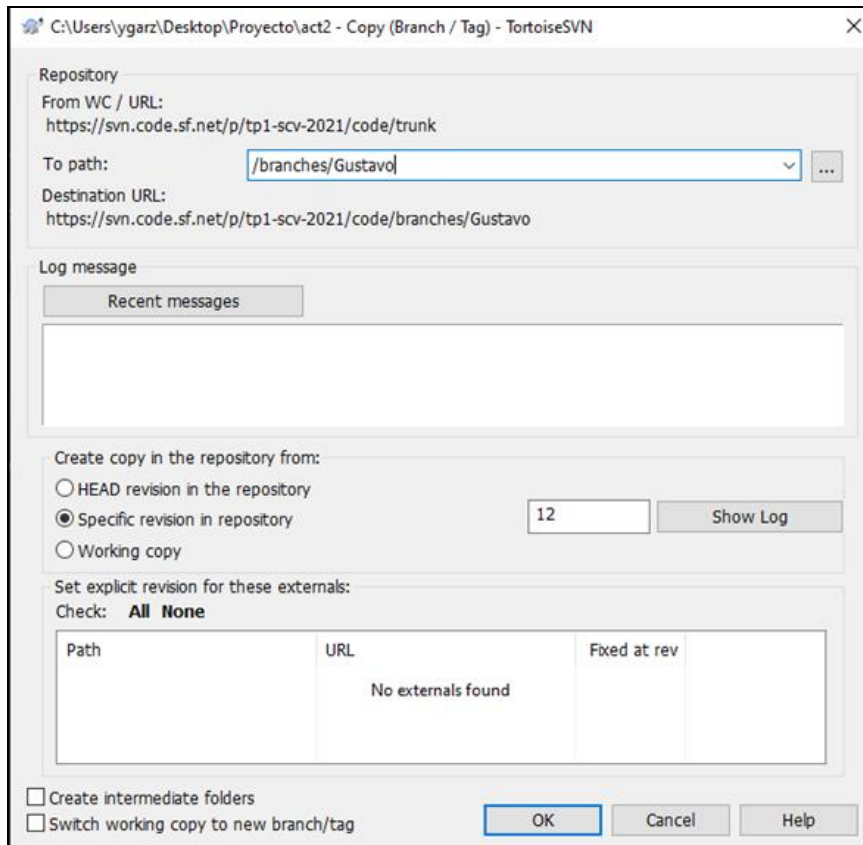
Por último, para tener un orden en el repositorio se selecciona la primera opción Branch/tag y se ordena el repositorio en branches, tags y trunk.

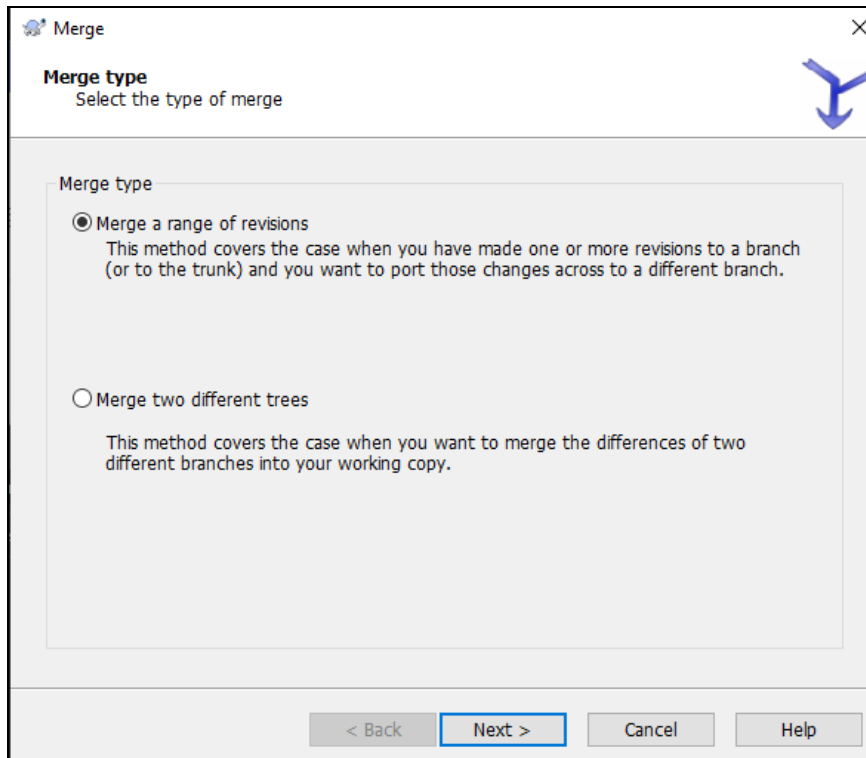
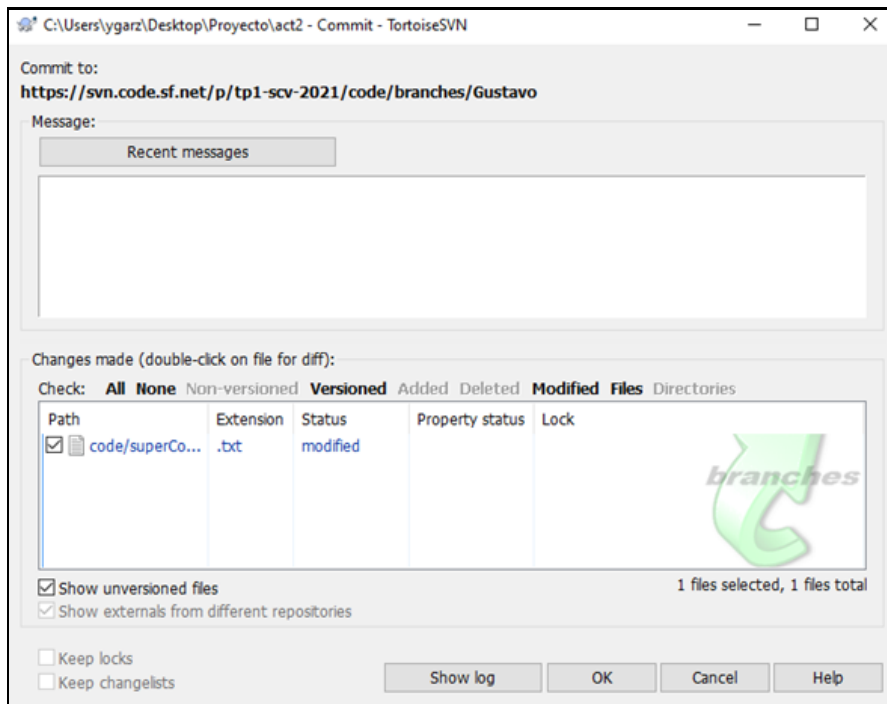



Nombre	Fecha de modificación	Tipo
 branches	14/4/2021 19:51	Carpeta de archivos
 tags	14/4/2021 19:27	Carpeta de archivos
 trunk	14/4/2021 19:41	Carpeta de archivos

Creación de rama y commit


Gustavo crea una rama:






 Merge

×



Merge revision range
Select the revisions to merge

URL to merge from

 https://svn.code.sf.net/p/tp1-scv-2021/code/branches/Gustavo

...

Revision range to merge

☐ all revisions

☒ specific range

☐ Reverse merge

Show log

Use the log dialog to select the revisions you want to merge, or enter the revisions to merge, separated by commas. A revision range can be specified by a dash.

Example: 4-7,9,11,15-HEAD@pegrevision

To merge all revisions (reintegrate/automatic merge), leave the box empty.

Working Copy

C:/Users/ygarz/Desktop/Proyecto/act2


Show log

< Back


Next >

Cancel

Help

 Merge

×



Merge options
Select the merge options

Merge revision range : Merge options

Merge depth:

Working copy

☐ Ignore line endings

☒ Compare whitespaces

☐ Ignore whitespace changes

☐ Ignore all whitespaces

☐ Ignore ancestry

☐ Force the merge

☐ Only record the merge (block revisions from getting merged)

☐ Do reintegrate instead of automatic merge (old style)

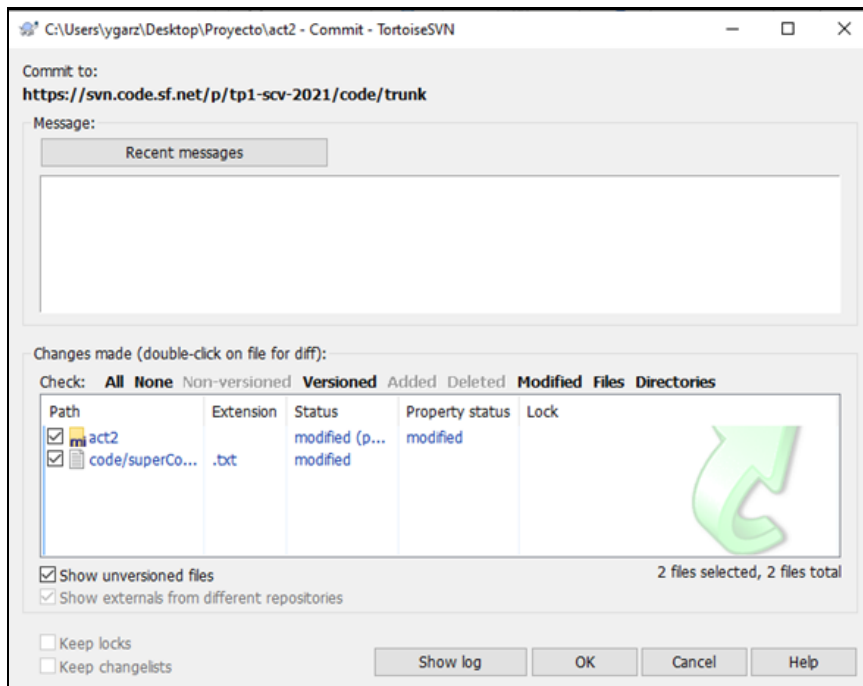
Test merge

< Back

Merge

Cancel

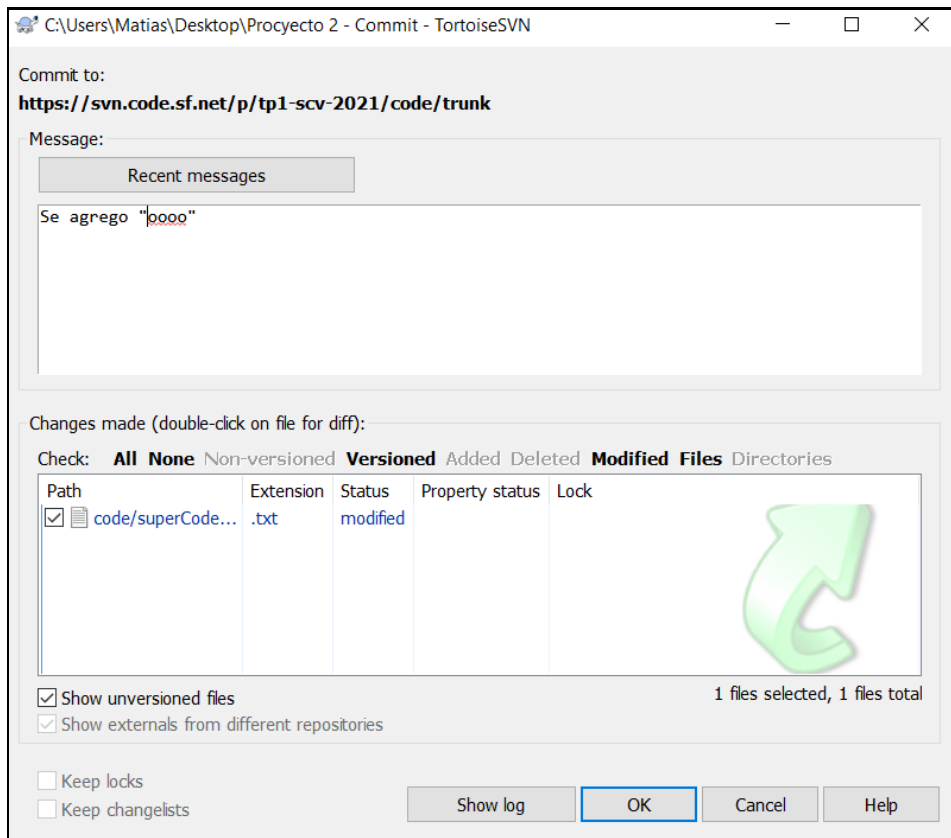
Help



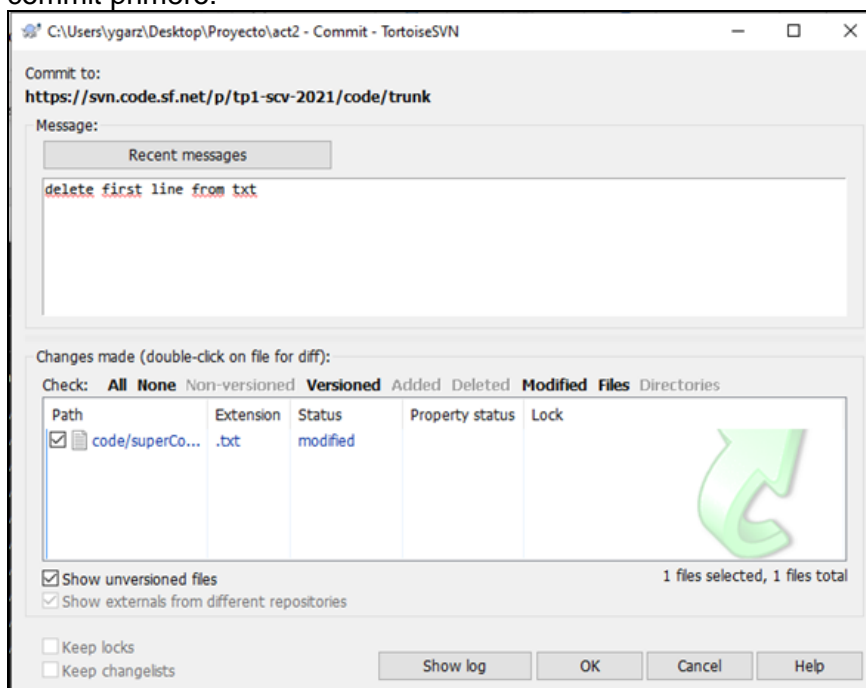
Commit para conflicto.

Cinthia y Gustavo modificaron el mismo archivo.
Cinthia commiteo primero.



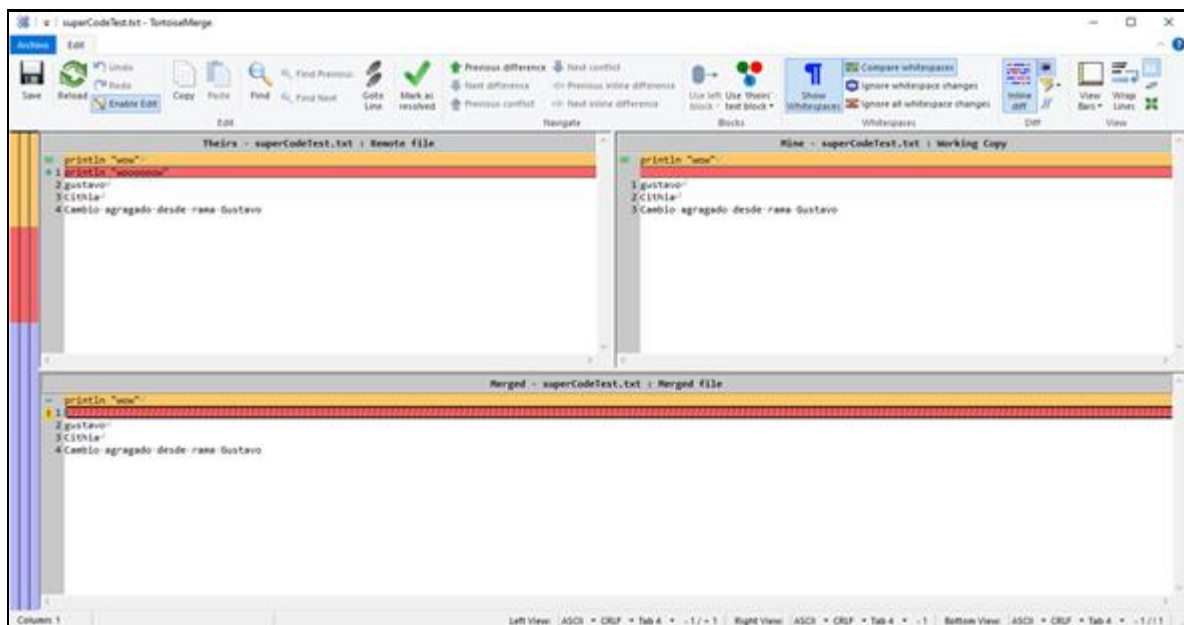
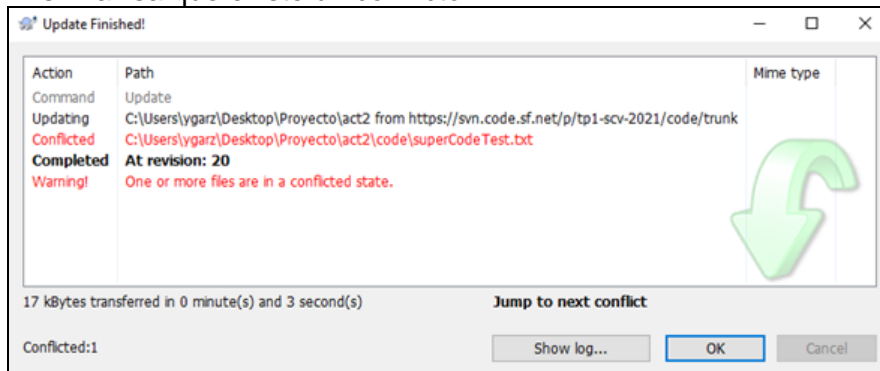


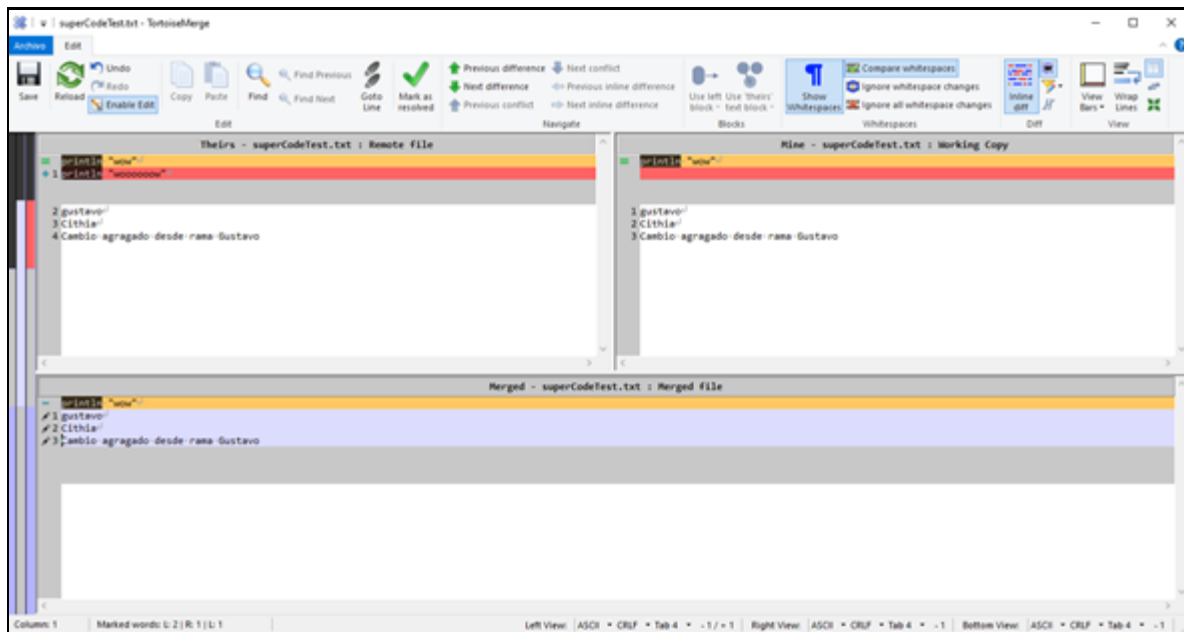
Ahora Gustavo quiere borrar una línea que ya no existe porque Cinthia lo borró e hizo el commit primero:





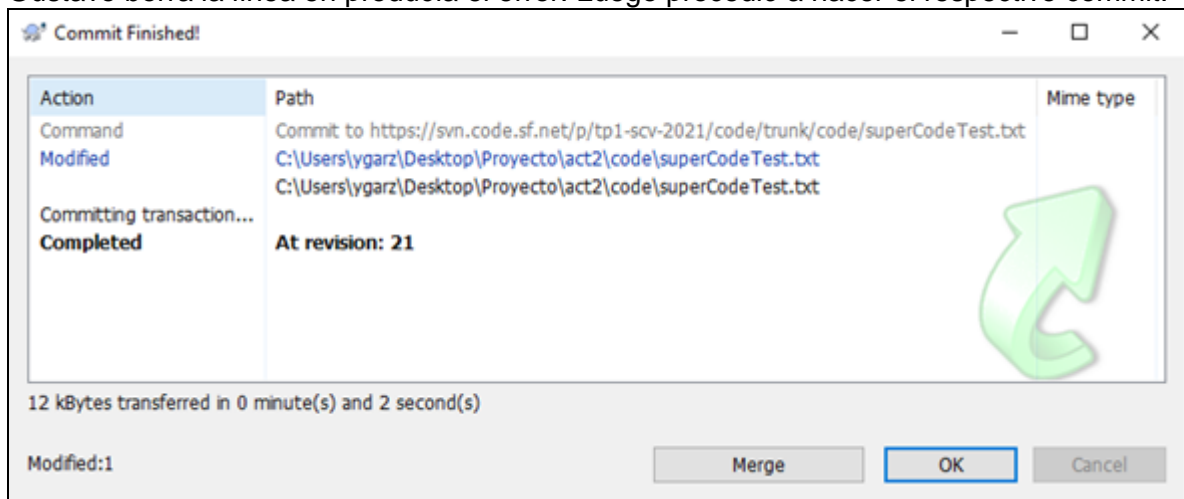
El svn avisa que existe un conflicto:





Resolución del conflicto:

Gustavo borra la línea en producía el error. Luego procedió a hacer el respectivo commit.



Actividad 3: Actividad práctica sobre Git y Github

Primero clonamos el repositorio principal en nuestro local y creamos la rama principal del grupo G1/principal

```
└─ cd cli-serv

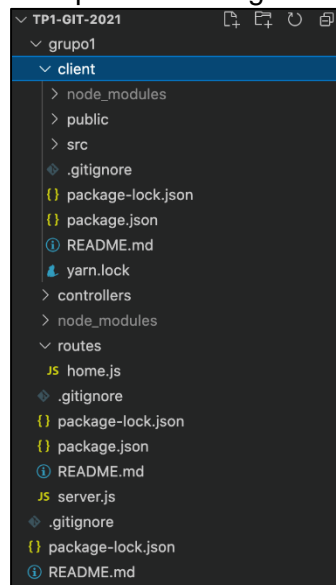
└─ [🍏] ~/Desktop/cli-serv ..... [✓]
└─ [ ] git clone https://github.com/FRRe-DACS/TP1-GIT-2021.git
└─ [ ] Cloning into 'TP1-GIT-2021'...
└─ [ ] remote: Enumerating objects: 379, done.
└─ [ ] remote: Counting objects: 100% (139/139), done.
└─ [ ] remote: Compressing objects: 100% (75/75), done.
└─ [ ] remote: Total 379 (delta 70), reused 83 (delta 55), pack-reused 240
└─ [ ] Receiving objects: 100% (379/379), 2.36 MiB | 26.00 KiB/s, done.
└─ [ ] Resolving deltas: 100% (145/145), done.

└─ [🍏] ~/Desktop/cli-serv ..... [✓] 1m 36s ⌚
└─ [ ] cd TP1-GIT-2021

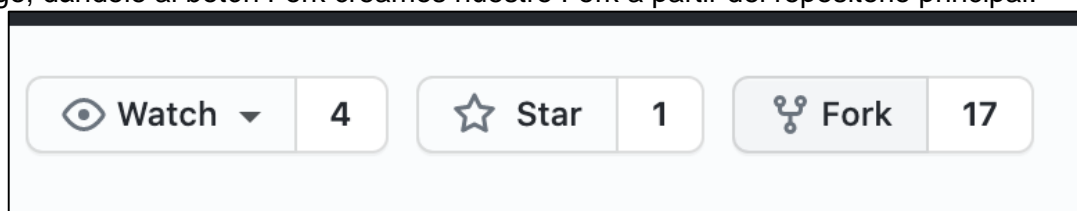
└─ [🍏] ~/Desktop/cli-serv/TP1-GIT-2021 ..... [✓]
└─ [ ] git checkout -b G1/principal
└─ [ ] Switched to a new branch 'G1/principal'

└─ [🍏] ~/Desktop/cli-serv/TP1-GIT-2021 ..... [✓]
└─ [ ] git G1/principal
```

Usando esta nueva rama, creamos un repositorio en node dentro de la carpeta grupo1. También decidimos agregar una carpeta cliente donde tenemos una aplicación que usa React. La estructura de carpetas nos quedó de la siguiente manera:



Luego, dándole al botón Fork creamos nuestro Fork a partir del repositorio principal.



Cada integrante del grupo clonó el fork, realizó un cambio, hizo un commit y subió su respectiva rama g1Nombre.

```
benjamin.kugler@benjamin:~/Desktop/cli-serv/TP1-GIT-2021
git remote add fork https://github.com/benjakugler96/TP1-GIT-2021

benjamin.kugler@benjamin:~/Desktop/cli-serv/TP1-GIT-2021
git checkout -b g1BK
Switched to a new branch 'g1BK'

benjamin.kugler@benjamin:~/Desktop/cli-serv/TP1-GIT-2021
git add .

benjamin.kugler@benjamin:~/Desktop/cli-serv/TP1-GIT-2021
git commit -m "Mostrar texto por pantalla"
On branch g1BK
nothing to commit, working tree clean

benjamin.kugler@benjamin:~/Desktop/cli-serv/TP1-GIT-2021
git push fork g1BK
```

Mediante pull requests fuimos revisando y aceptando los cambios que cada uno subió y al final nos quedó todo integrado en la rama principal G1/principal del fork.

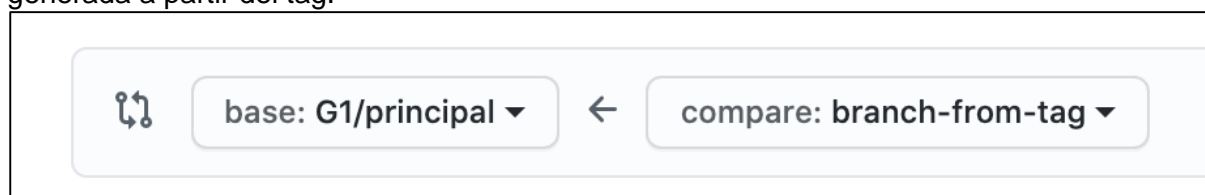
Luego de obtener la funcionalidad completa, generamos el tag g1-V-1.0.0, y también vimos los comandos que existen para listar tags o para ver la información sobre cierto tag.

```
benjamin.kugler@benjamin:~/Desktop/cli-serv/TP1-GIT-2021
git tag g1-V-1.0.0 -m "primer tag"

benjamin.kugler@benjamin:~/Desktop/cli-serv/TP1-GIT-2021
git tag

benjamin.kugler@benjamin:~/Desktop/cli-serv/TP1-GIT-2021
git show g1-V-1.0.0
```

Por último, generamos un error en la rama principal, y creamos un PR a partir de una rama generada a partir del tag.



Bibliografía

Control de versiones con Subversion Revision 4849 Ben Collins-Sussman Brian W. Fitzpatrick C. Michael Pilato

<https://git-scm.com/>

<https://www.mercurial-scm.org/>

<http://www.bitkeeper.org/>

<https://www.nongnu.org/cvs/>

<https://subversion.apache.org/>

<https://tortoisesvn.net/>

<https://sourceforge.net/>

<https://git-scm.com/>

<https://www.educba.com/introduction-to-git/>