**Instituto Tecnológico de Los Mochis.**



Ingeniería en Informática.

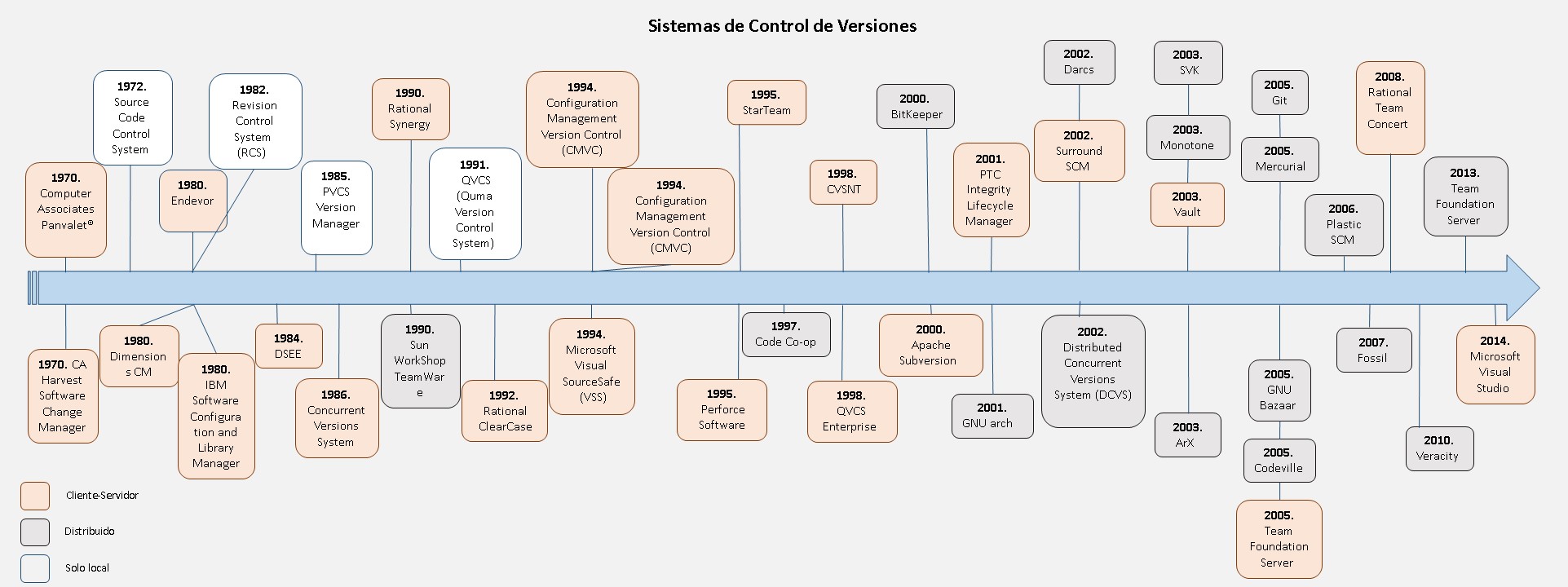
Herramientas de Apoyo para el Desarrollo de Software.

**Alumno:** Sánchez Zavala José Alejandro.

**Profesora:** Algara Norzagaray Juan Francisco.

Los Mochis, Sinaloa, a Febrero del 2018.

**1.- Linea de tiempo de los Sistemas de Control de Versiones.**



Antes de seguir definiremos lo que es un sistema de control de versiones. Es un software que administra el acceso a un conjunto de ficheros, y mantiene un historial de cambios realizados. El control de versiones es útil para guardar cualquier documento que cambie con frecuencia, como código fuente, documentación o ficheros de configuración.

Clasificación

Los sistemas de control de versiones se pueden clasificar en 2 grandes grupos:

Centralizados

En un sistema de control de versiones centralizado todas nuestras fuentes y sus versiones están almacenados en un único directorio (llamado repositorio de fuentes) de un ordenador (un servidor). Todos los desarrolladores que quieran trabajar con esas fuentes, deben pedirle al sistema de control de versiones una copia local para trabajar. En ella realizan todos sus cambios y cuando están listos y funcionando, le dicen al sistema de control de versiones que guarde las fuentes modificados como una nueva versión. Algunos ejemplos son CVS y subversión.

Distribuidos

En un sistema de control de versiones distribuido no hay un repositorio central. Todos los desarrolladores tienen su propia copia del repositorio, con todas las versiones y toda la historia. Por supuesto, según van desarrollando y haciendo cambios, sus fuentes y versiones van siendo distintas unas de otras. Sin embargo, los sistemas de control de versiones distribuidos permiten que en cualquier momento dos desarrolladores cualesquiera puedan "sincronizar" sus repositorios. Si uno de los desarrolladores ha tocado determinados fuentes y el otro no, los modificados se convierten en la versión más moderna. Ejemplos: Git y Mercurial.

Ahora continuamos con la línea del tiempo.

1970. Computer Associates Panvalet® (también conocido como CA-Panvalet).

1970. CA Harvest Software Change Manager (originalmente conocido como CCC / Harvest).

1972. Source Code Control System (SCCS).

1980. Dimensions CM.

1980. Endevor.

1980. IBM Software Configuration and Library Manager (SCLM).

1982. Revision Control System (RCS).

1984. DSEE.

1985. PVCS Version Manager (originalmente llamado Polytron Version Control System).

1986. Concurrent Versions System (CVS) también conocido como el Sistema Concurrent Versioning, es un sistema de control de revisiones cliente-servidor de software libre en el campo del desarrollo de software. Un sistema de control de versiones realiza un seguimiento de todo el trabajo y todos los cambios en un conjunto de archivos, y permite que varios desarrolladores (potencialmente muy separados en el espacio y el tiempo) [colaboren](https://en.wikipedia.org/wiki/Collaboration). [Dick Grune](https://en.wikipedia.org/wiki/Dick_Grune) desarrolló CVS como una serie de [guiones](https://en.wikipedia.org/wiki/Shell_scripts) de [shell](https://en.wikipedia.org/wiki/Shell_scripts) en julio de 1986.

CVS utiliza una arquitectura [cliente-servidor](https://en.wikipedia.org/wiki/Client%E2%80%93server) : un servidor almacena la (s) versión (es) actual (es) de un [proyecto](https://en.wikipedia.org/wiki/Project) y su historial, y los clientes se conectan al servidor para "verificar" una copia completa del proyecto, trabajar en esta copia y luego más tarde "registrar" sus cambios. Normalmente, el cliente y el servidor se conectan a través de una [LAN](https://en.wikipedia.org/wiki/Local_area_network) o de [Internet](https://en.wikipedia.org/wiki/Internet) , pero tanto el cliente como el servidor pueden ejecutarse en la misma máquina si CVS tiene la tarea de realizar un seguimiento del historial de versiones de un proyecto solo con desarrolladores locales.

Varios desarrolladores pueden trabajar en el mismo proyecto al mismo tiempo, cada uno editando archivos dentro de su propia "copia de trabajo" del proyecto y enviando (o *registrando*) sus modificaciones al servidor. Para evitar conflictos, el servidor solo acepta los cambios realizados en la versión más reciente de un archivo. Por lo tanto, se espera que los desarrolladores mantengan actualizada su copia de trabajo incorporando los cambios de otras personas de forma regular. El cliente de CVS maneja esta tarea en su mayoría automáticamente, y requiere intervención manual solo cuando surge un [conflicto de edición](https://en.wikipedia.org/wiki/Edit_conflict) entre una modificación registrada y la versión local aún no verificada de un archivo.

Los clientes también pueden comparar versiones, solicitar un historial completo de cambios o consultar una instantánea histórica del proyecto a partir de una fecha determinada o de un número de revisión.

Los servidores CVS pueden permitir el "acceso de lectura anónimo", en el que los clientes pueden verificar y comparar versiones con una contraseña en blanco o simple publicada (por ejemplo, "anoncvs"); solo el registro de cambios requiere una cuenta personal y contraseña en estos escenarios.

Los clientes también pueden usar el comando "actualizar" para actualizar sus copias locales con la versión más reciente en el servidor. Esto elimina la necesidad de descargas repetidas de todo el proyecto.

Por otro lado, cuando se le dice a CVS que almacene un archivo como binario, mantendrá cada versión individual en el servidor. Almacenar archivos como binarios es importante para evitar la corrupción de archivos binarios.

1990. Rational Synergy.

1990. Sun WorkShop TeamWare más tarde Forte TeamWare, luego Forte Code Management Software).

1991. QVCS (Quma Version Control System).

1992. Rational ClearCase.

1994. Configuration Management Version Control (CMVC).

1994. Microsoft Visual SourceSafe (VSS).

1995. Perforce Software**.**

1995. StarTeam.

1997. Code Co-op.

1998. CVSNT.

1998. QVCS Enterprise (Quma Version Control System).

2000. BitKeeper.

2000. Apache Subversion (a menudo abreviado SVN , después de su nombre de comando *svn* ) es un [sistema de control de](https://en.wikipedia.org/wiki/Software_versioning)[revisiones](https://en.wikipedia.org/wiki/Revision_control) y [versiones de software](https://en.wikipedia.org/wiki/Software_versioning) distribuido como [código abierto](https://en.wikipedia.org/wiki/Open_source) bajo la [licencia de Apache](https://en.wikipedia.org/wiki/Apache_License).

Algunas características:

- Se compromete como verdaderas operaciones atómicas (las operaciones de confirmación interrumpidas en CVS causarían incoherencia o corrupción del repositorio).  
- Los archivos renombrados / copiados / movidos / eliminados retienen el historial completo de revisiones.  
- El sistema mantiene el control de versiones para directorios, cambios de nombre y metadatos de archivos (pero no para marcas de tiempo). Los usuarios pueden mover y / o copiar árboles de directorios completos muy rápidamente, conservando el historial completo de revisiones.  
- Control de versiones de enlaces simbólicos.  
- Soporte nativo para archivos binarios, con almacenamiento binario-diff con ahorro de espacio.  
- Apache HTTP Server como servidor de red, WebDAV / Delta-V para protocolo. También hay un proceso de servidor independiente llamado svnserve que usa un protocolo personalizado sobre TCP / IP.  
- La ramificación es una operación económica, independiente del tamaño del archivo (aunque Subversion por sí misma no distingue entre una rama y un directorio).  
- Nativamente cliente-servidor, diseño de biblioteca en capas.  
- El protocolo cliente / servidor envía diffs en ambas direcciones.  
- Costos proporcionales al tamaño del cambio, no al tamaño de los datos.  
- Salida procesable, incluida la salida de registro XML.  
- licencia de fuente abierta - Apache License desde la versión 1.7; las versiones anteriores usan una derivada de la Licencia de Software Apache 1.1.  
- Mensajes de programas internacionalizados.  
- Bloqueo de archivos para archivos no fusionables ("registros reservados").  
- Autorización basada en ruta.  
- Enlaces de lenguaje para C #, PHP, Python, Perl, Ruby y Java.  
Soporte completo MIME: los usuarios pueden ver o cambiar el tipo MIME de cada archivo, y el software sabe qué tipos MIME pueden mostrar diferencias con respecto a las versiones anteriores.  
- Seguimiento de fusión: se realizará un seguimiento de las fusiones entre sucursales, esto permite la fusión automática entre las sucursales sin decirle a Subversion qué necesita y qué no necesita fusionarse.  
- Listas de cambios para organizar confirmaciones en grupos de compromiso.

2001. PTC Integrity Lifecycle Manager (anteriormente *MKS Integrity*).

2001. GNU arch.

2002. Darcs.

2002. Distributed Concurrent Versions System (DCVS).

2002. Surround.

2003. Vault.

2003. ArX.

2003. Monotone es una herramienta de [software de](https://en.wikipedia.org/wiki/Software)[código](https://en.wikipedia.org/wiki/Open_source)[abierto](https://en.wikipedia.org/wiki/Software) para el [control de revisiones distribuidas](https://en.wikipedia.org/wiki/Distributed_revision_control). Monotone realiza un seguimiento de las revisiones de los archivos, agrupa conjuntos de revisiones en conjuntos de cambios y rastrea el historial a través de los cambios de nombre. El objetivo del proyecto es la integridad sobre el rendimiento. Monotone está diseñado para operaciones distribuidas, y hace un uso intensivo de [primitivas criptográficas](https://en.wikipedia.org/wiki/Cryptographic_primitive) para rastrear revisiones de archivos (a través del [hash](https://en.wikipedia.org/wiki/SHA-1)[seguro](https://en.wikipedia.org/wiki/Cryptographic_hash_function)[SHA-1](https://en.wikipedia.org/wiki/SHA-1)) y para autenticar acciones del usuario (a través de [firmas criptográficas](https://en.wikipedia.org/wiki/Digital_signature)[RSA](https://en.wikipedia.org/wiki/RSA_(algorithm))).

Algunas características destacadas:

- Integridad. Antes de una gran optimización en la revisión 0.27, el énfasis de Monotone en la corrección sobre la optimización a menudo se atribuía a experiencias iniciales pobres. La primera acción de un nuevo usuario suele sincronizar (clonar) una gran base de datos Monotone existente, una acción que a menudo tomaba horas para grandes bases de datos, debido a la amplia validación y comprobación de integridad que realiza Monotone cuando las revisiones se mueven a través de la red. Una vez que se completa la base de datos inicial (clonar), las acciones posteriores generalmente se realizan más rápidamente.

- Flujo de trabajo. Monotone es especialmente fuerte en su soporte para un flujo de trabajo de divergencia / fusión, que logra en parte permitiendo siempre la confirmación antes de la fusión.

Otras características de Monotone incluyen:

* Buen soporte para [internacionalización y localización](https://en.wikipedia.org/wiki/Internationalization_and_localization)
* Diseño portátil, implementado en [C ++](https://en.wikipedia.org/wiki/C%2B%2B)
* La alta integridad es un objetivo clave de diseño
* Monotone puede importar proyectos de [CVS](https://en.wikipedia.org/wiki/Concurrent_Versions_System) .
* [Firma](https://en.wikipedia.org/wiki/Digital_signature) de revisiones utilizando certificados [RSA](https://en.wikipedia.org/wiki/RSA_(algorithm))
* Fácil de aprender, debido a un conjunto de comandos similar al de [CVS](https://en.wikipedia.org/wiki/Concurrent_Versions_System)
* Muy bueno en la ramificación (ambas divergencias dentro de una rama y ramas nombradas) y fusión
* Buena documentación
* Muy bajo mantenimiento
* Existen [interfaces de usuario gráficas](https://en.wikipedia.org/wiki/Graphical_user_interface) estables :
  + [guitone](http://guitone.thomaskeller.biz/) , un [Qt](https://en.wikipedia.org/wiki/Qt_(toolkit)" \o "Qt (kit de herramientas)) -front para gestionar espacios de trabajo y bases de datos (compatible con MS-Windows y Unix / Linux / MacOS)
  + [mtn-browse](http://www.coosoft.plus.com/software.html) , un navegador gráfico [Gtk2](https://en.wikipedia.org/wiki/GTK2) que le permite explorar la base de datos, incluso de forma remota, sin la necesidad de un espacio de trabajo (compatible con Unix / Linux / MacOS)
  + [Monotone-Viz](http://oandrieu.nerim.net/monotone-viz) , un historial de revisión grapher (compatible con MS-Windows y Unix / Linux)
  + [TracMtn](http://tracmtn.1erlei.de/) , un plugin [Trac](https://en.wikipedia.org/wiki/Trac) para el historial y la búsqueda de repositorios
* Completa y completa biblioteca [Perl](https://en.wikipedia.org/wiki/Perl) que le permite controlar completamente Monotone desde un script Perl (mtn-browse hace uso de esto.

2003. SVK (también escrito svk) es un [sistema de control de versiones](https://en.wikipedia.org/wiki/Version_control_system) descentralizado escrito en [Perl](https://en.wikipedia.org/wiki/Perl) , con un diseño distribuido jerárquico comparable a la implementación centralizada de [BitKeeper](https://en.wikipedia.org/wiki/BitKeeper" \o "BitKeeper) y [GNU arch](https://en.wikipedia.org/wiki/GNU_arch) . Se distribuye bajo la [Licencia artística](https://en.wikipedia.org/wiki/Artistic_License) y la [Licencia pública general de GNU](https://en.wikipedia.org/wiki/GNU_General_Public_License) . El autor principal de svk es Kao Chia-liang ( [chino](https://en.wikipedia.org/wiki/Chinese_language) : 高 嘉 良). El 5 de junio de 2006, Chia-liang Kao se unió a Best Practical Solutions, LLC, fabricantes de [Request Tracker](https://en.wikipedia.org/wiki/Request_Tracker" \o "Solicitar rastreador).

SVK usa el [sistema de](https://en.wikipedia.org/wiki/Apache_Subversion) archivos [Subversion](https://en.wikipedia.org/wiki/Apache_Subversion" \o "Subversion de Apache) pero proporciona características adicionales:

* Operaciones fuera de línea como *checkin*, *log*, *merge*.
* Ramas distribuidas.
* Ligera gestión de copia de pago (no *.svn* directorios).
* Cambio de firma y verificación.
* Puede duplicar y operar en repositorios [Subversion](https://en.wikipedia.org/wiki/Apache_Subversion" \o "Subversion de Apache) , [Perforce](https://en.wikipedia.org/wiki/Perforce" \o "Forzosamente) y [CVS](https://en.wikipedia.org/wiki/Concurrent_Versions_System).

2005. Mercurial es una herramienta [distribuida de control de revisiones](https://en.wikipedia.org/wiki/Distributed_revision_control) para [desarrolladores de software](https://en.wikipedia.org/wiki/Software_developer). Es compatible con [Microsoft Windows](https://en.wikipedia.org/wiki/Microsoft_Windows) y [sistemas tipo](https://en.wikipedia.org/wiki/Unix-like) Unix, como [FreeBSD](https://en.wikipedia.org/wiki/FreeBSD" \o "FreeBSD) , [macOS](https://en.wikipedia.org/wiki/MacOS" \o "Mac OS) y [Linux](https://en.wikipedia.org/wiki/Linux).

Los principales objetivos de diseño de Mercurial incluyen alto rendimiento y escalabilidad, desarrollo colaborativo descentralizado y completamente distribuido, manejo robusto tanto de [texto sin formato](https://en.wikipedia.org/wiki/Plain_text) como de [archivos binarios](https://en.wikipedia.org/wiki/Binary_file) , y funciones avanzadas de bifurcación y fusión, sin dejar de ser conceptualmente simples. [[3]](https://en.wikipedia.org/wiki/Mercurial#cite_note-3) Incluye una interfaz web integrada. Mercurial también ha tomado medidas para facilitar la transición para los usuarios de otros sistemas de control de versiones, particularmente [Subversion](https://en.wikipedia.org/wiki/Apache_Subversion" \o "Subversion de Apache).

2005. GNU Bazaar (anteriormente Bazaar-NG, herramienta de línea de comandos bzr) es un sistema de control de revisiones distribuido y cliente-servidor patrocinado por Canonical.

Bazaar puede ser utilizado por un único desarrollador que trabaje en múltiples ramas de contenido local o por equipos que colaboran en una red.

Bazaar está escrito en el lenguaje de programación Python, con paquetes para las principales distribuciones de GNU / Linux, Mac OS X y Microsoft Windows. Bazaar es software libre y parte del Proyecto GNU.

Los comandos de Bazar son similares a los encontrados en [CVS](https://en.wikipedia.org/wiki/Concurrent_Versions_System) o [Subversion](https://en.wikipedia.org/wiki/Subversion_(software)" \o "Subversion (software)).

A diferencia de los sistemas de control de versiones puramente distribuidos que no usan un servidor central, Bazaar admite el trabajo con o sin un servidor central. Es posible utilizar ambos métodos al mismo tiempo con el mismo proyecto. Los sitios web [Launchpad](https://en.wikipedia.org/wiki/Launchpad_(website)" \o "Launchpad (sitio web)) y [Sourceforge](https://en.wikipedia.org/wiki/Sourceforge" \o "Sourceforge) proporcionan un servicio de alojamiento gratuito para proyectos gestionados con Bazar.

2005. Git es un sistema de control de versiones para rastrear cambios en archivos de computadora y coordinar el trabajo en esos archivos entre varias personas. Se utiliza principalmente para la gestión de código fuente en el desarrollo de software, pero se puede utilizar para realizar un seguimiento de los cambios en cualquier conjunto de archivos. Como sistema de control de revisión distribuido, está orientado a la velocidad, integridad de datos, y soporte para flujos de trabajo distribuido, no lineal.

Git fue creado por Linus Torvalds.

Como con la mayoría de los otros sistemas de control de versiones distribuidas, ya diferencia de la mayoría de los sistemas cliente-servidor, cada directorio de Git en cada computadora es un repositorio completo con historial completo y capacidad de seguimiento de versión completa, independientemente del acceso a la red o un servidor central.

Git es un software gratuito distribuido bajo los términos de la Licencia Pública General GNU versión 2.

Características:

- Soporte fuerte para el desarrollo no lineal. Git admite ramificaciones y fusiones rápidas, e incluye herramientas específicas para visualizar y navegar un historial de desarrollo no lineal. En Git, una suposición básica es que un cambio se fusionará con más frecuencia de lo que está escrito, ya que se transmite a varios revisores. En Git, las ramas son muy livianas: una rama es solo una referencia a una confirmación. Con sus confirmaciones parentales, se puede construir la estructura de rama completa.

- Desarrollo distribuido.

Al igual que Darcs, BitKeeper, Mercurial, SVK, Bazaar y Monotone, Git proporciona a cada desarrollador una copia local del historial de desarrollo completo y los cambios se copian de uno de esos repositorios a otro. Estos cambios se importan como ramas de desarrollo agregadas y se pueden combinar de la misma manera que una rama desarrollada localmente.

- Compatibilidad con sistemas y protocolos existentes. Los repositorios se pueden publicar a través de Protocolo de transferencia de hipertexto (HTTP), Protocolo de transferencia de archivos (FTP), rsync (eliminado en Git 2.8.0) o un protocolo Git sobre un socket simple o Secure Shell (ssh).

- Manejo eficiente de grandes proyectos. Torvalds describió a Git como muy rápido y escalable, y las pruebas de rendimiento realizadas por Mozilla mostraron que era un orden de magnitud más rápido que algunos sistemas de control de versiones, y recuperar el historial de versiones de un repositorio almacenado localmente puede ser cien veces más rápido que ir a buscarlo desde el servidor remoto.

2005. Codeville.

2005. Team Foundation Server.

2006. Plastic SCM.

2007. Fossil.

2008. Rational Team Concert.

2010. Veracity.

2013. Team Foundation Server.

2014. Microsoft Visual Studio**.**

**2.- Indique para un sistema de revisión y control no distribuido como se realizan los siguientes procedimientos.**

**a) Crear un repositorio.**

$ svnadmin create /home/tu\_usuario/public\_html/ruta\_repositorio

**b) Eliminar un repositorio.**

$ svn rm -rf /path/to/directory

**c) Compartir un repositorio.**

**d) Agregar un archivo al repositorio.**

$svn add /ruta/index.php

$svn add \*

**e) Eliminar un archivo del repositorio.**

$ svn rm cambios.txt

Borrar un fichero de nuestro disco duro.

$ svn delete archivo

Quitar un fichero del proyecto (y lo borramos de nuestro disco duro).

$ svn delete –keep-local archivo2

Quitar otro fichero del repositorio, pero mantener la copia local.

**f) Comparar dos versiones de un archivo del repositorio.**

$svn diff nombre\_archivo

Este comando indica las diferencias entre la versión del svn actualizado y la copia local nuestra.

$ svn diff -r3:4 info.txt

El comando svn diff acepta el parámetro -r para especificar los números de versiones entre los que queremos conocer la diferencia. En este caso necesitamos conocer los cambios entre las revisiones 3 y 4, por lo que pasamos esos números. Se puede omitir el parámetro -r y entonces Subversion nos dará las diferencias entre el estado actual del archivo y el estado que tenía la última vez que actualizamos el repositorio.

**g) Recobrar versiones pasadas de un archivo del repositorio.**

$ svn revert info.txt

**3.- Igualmente que en el punto 2 pero para un sistema distribuido.**

**a) Crea un repositorio nuevo.**

Crea un directorio nuevo, abrir y ejecutar.

git init

**b) Eliminar un repositorio.**

git rm “nombre del repositorio”

**c) Compartir un repositorio.**

**d) Agregar un archivo al repositorio.**

Add & commit

Puedes registrar cambios (añadirlos al Index) usando

git add <filename>

git add \*

Este es el primer paso en el flujo de trabajo básico. Para hacer commit a estos cambios usa

git commit -m “Commit message”

Ahora el archivo está incluido en el HEAD, pero aún no en tu repositorio remoto.

**e) Eliminar un archivo del repositorio.**

Para borrar archivos o carpetas también “arriba” en nuestro respositorio (bitbucket, github…) se usa el comando “rm” de git siguiendo los siguientes 3 pasos.

1.1 Si quieres eliminar un archivo:

git rm miarchivo.php

1. b) Si quieres eliminar una carpeta:

git rm -r micarpeta

2. Creamos el commit

git commit -m “elimino archivos innecesarios”

3. Subimos los cambios al repositorio

git push

**f) Comparar dos versiones de un archivo del repositorio.**

**g) Recobrar versiones pasadas de un archivo del repositorio.**

**4.- ¿Cuáles son las mejores prácticas en cuanto al uso y almacenamiento de binarios en repositorios?**

* Reducir el tamaño de los documentos de Microsoft Office.
* Reducir el tamaño de sus imágenes.
  + Reducir sus imágenes, es decir, el tamaño físico, por ejemplo de 1.280 × 720 píxeles a 640 x 360 píxeles.
  + Comprimir imágenes es decir, mantener el tamaño físico, sino eliminar los datos innecesarios dentro de la imagen.
* No duplicar archivos.
* No crear estructuras de carpetas de más cuatro niveles.

**5.- ¿Cuáles son (y justifique) las mejores prácticas sobre el uso de repositorios y bases de datos?**

Si usas bases de datos distribuidas, existe una herramienta llamada klonio la cual es un repositorio para bases de datos.

Klonio funciona con git + github y puedes colaborar con otros desarrolladores sobre la misma aplicación.

Con Klonio puedes hacer diffs (comparaciones) en la base de datos y en los commits.

Los commits se respaldan y puedes restaurarlos, esto ayuda a que tu base de datos regrese a un estado de normal después de crear un commit que afecte tu programa.

Si planeas subir sql dump lo que puedes hacer es implementar el siguiente comando:

--skip-extended-insert

Lo que pasará es que le dirá a mysqldump que cada fila de la columna tendrá su propia inserción en el repositorio.

Esto creará un commit inicial más largo de lo común, pero hará los commits futuros más simples para leer.

Mantendrá el repositorio más lento, ya que solamente se modificarán commits individual de donde se añada, modifique, o elimine contenido.

También puedes usar git gc el cual te ayudará a limpiar archivos innecesarios.

Puedes ver la actividad git whatchanged –p para saber que ha cambiado en tu repositorio.

**6.- ¿Cuál es el papel que juegan los sistemas de revisión y control en la integración continua?**

La idea de la integración continua es hacer integraciones de código automáticas de un programa lo más a menudo posible para poder detectar fallos, los sistemas de revisión y control ayudan a los desarrolladores a subir su contenido a un sistema que permite ver cuáles han sido los cambios, si se ha modificado información, si se ha quitado o añadido, ésta herramienta ayuda no solamente a tener una actualización rápida y eficiente del código del programa sino que también ayuda a tener una visión más clara de lo que está pasando en nuestro sistema, poder pivotear en caso de fallos, poder saber que los ocasiona, cual es la prioridad de ese fallo, que tanto afecta y que tan rápido puede integrarse al proyecto inicial (en caso de usar ramas). Los desarrolladores pueden detectar y solucionar problemas de integración de forma continua, evitando el caos de última hora cuando se acercan las fechas de entrega.

Depende de la herramienta de revisión y control que utilices serán las herramientas que tendrás a tu disposición para poder tener un control más eficiente de tu proyecto.

Algunas de las ventajas que te brinda trabajar sobre disponibilidad constante de una versión para pruebas, demos o lanzamientos anticipados, ejecución inmediata de las pruebas unitarias, monitorización continúa de las métricas de calidad del proyecto.

**7. ¿Cuáles son las alternativas empresariales en cuanto a sistemas de revisión de control?**

Mercurial

- Fácil de aprender.  
- Usabilidad de GUI más sencilla.  
- No puedes re-escribir la historia.  
- Construido en Webserver.

SVN

- Visibilidad de cambio entre actores del equipo.  
- Ayuda a medir la calidad y la cantidad del trabajo en una unidad de tiempo.  
- No provoca inconsistencias en caso de ser interrumpido.  
- Guarda las diferencias de una versión a otra, pero solamente lo que es modificado, no duplica el archivo anterior.

GIT

- La forma de interactuar con ramas es muy dinámica y sencilla.  
- Puedes exportar tus repositorios a otros sistemas de revisión y control como mercurial.  
- Las extensiones pueden ser escritas en distintos lenguajes.  
- Control de etiquetas más elaborado.

IBM CLEAR CASE

- Modificaciones concurrentes de los mismos archivos.  
- Cuenta con autentificación de usuario.  
- Permite tener control del rastro del trabajo del equipo.  
- Te ayuda a tener control de código, librerías, documentación, binario, diseños de hardware y más.  
- Tiene seguridad por manejo y protección de IP.  
- Dependencia entre actividades no relacionadas.

PERFOCE

- Permite planificación ágil del proyecto.  
- Manejo de portafolio.  
- Manejo de issue’s para desarrollo de productos más complejos.  
- Disponible en dispositivos móviles.  
- Permite control de equipos por jerarquías.

**8. Investigue y exponga sobre casos de problemas de seguridad y privacidad de sistemas de revisión y control.**

“Add GitHub dorking to list of security concerns”

El otro componente clave fue la adición de la función de búsqueda interna de GitHub en 2013, lo que hace simple ejecutar consultas de repositorios públicos y privados GitHub a las que solo ciertos usuarios tenían acceso. Casi de inmediato, observadores astutos observaron que la característica podría utilizarse para revelar las claves de cifrado privadas y credenciales de inicio de sesión enterradas código registrado en GitHub.

“Hack brief: uber paid off hackers to hide a 57-million user data breach”

Según Bloomberg, la ruptura del 2016 de Uber se produjo cuando un grupo de hackers descubrieron que los desarrolladores de la compañía habían publicado código que incluyó sus nombres de usuario y contraseñas en una cuenta privada del repositorio de software Github. Esas credenciales le dieron a los hackers acceso inmediato a cuentas privilegiado de los desarrolladores de red de Uber, y con ello, acceso a servidores de Uber y con eso acceso a información sensible alojada en los servidores de Amazon, incluyendo los datos del pasajero y el conductor.

**10. ¿Qué se debe de hacer con los archivos que ayudan a realizar el proyecto pero que no forman parte directa de él (casos de uso, diagramas entidad relación, etc.)?**

1. Separar los documentos que quieres compartir y los que no se deben compartir, Administrar diferentes privilegios a las diferentes esferas del equipo (No es muy recomendable en equipos nuevos o inexpertos ya que puede retrasar el trabajo).

2. Almacenarlos en carpetas bien estructuradas, separadas del código (tomar en cuenta que esas carpetas deben ser removidas en caso de querer distribuir el software).

3. Está bien tener la documentación del proyecto en el repositorio, siempre y cuando esté en fase de desarrollo, esto es para agilizar el trabajo.

4. No es recomendable subir documentos confidenciales aun teniendo un buen control de privilegios, ya que es complicado borrar los archivos del historial.

**9. Como se reconcilian dos versiones de un archivo en un sistema no distribuido.**

**11. Igualmente que en el punto 9 pero para un sistema distribuido.**