Control de Versiones

GESTIÓN DEL SOFTWARE, COLABORACIÓN Y EL CONTROL DE VERSIONES

Evolución del software

- Durante el desarrollo
  - El desarrollo del software siempre es progresivo, incluso en el ciclo de vida en cascada
  - El desarrollo evolutivo consiste, precisamente, en una evolución controlada (ciclo de vida espiral, prototipos evolutivos)
- Durante la explotación
  - Durante la fase de mantenimiento se realizan modificaciones sucesivas del producto
- Además de ello, el desarrollo de software se realiza normalmente en equipo, por lo que es necesario integrar varios desarrollos en un solo producto

Control de versiones

- Un control de versiones es un sistema que registra los cambios realizados en un archivo o conjunto de archivos a lo largo del tiempo, de modo que puedas recuperar versiones específicas más adelante.
- Los sistemas de control de versiones son herramientas de software que ayudan a los equipos de software a gestionar los cambios en el código fuente a lo largo del tiempo.
- A medida que los entornos de desarrollo se aceleran, los sistemas de control de versiones ayudan a los equipos de software a trabajar de forma más rápida e inteligente. Son especialmente útiles para los equipos de DevOps, ya que les ayudan a reducir el tiempo de desarrollo y a aumentar las implementaciones exitosas.

Características

- Mecanismo de almacenamiento de los elementos que deba gestionar (ej. archivos de texto, imágenes, documentación...).
- Posibilidad de realizar cambios sobre los elementos almacenados (ej. modificaciones parciales, añadir, borrar, renombrar o mover elementos).
- Registro histórico de las acciones realizadas con cada elemento o conjunto de elementos (normalmente pudiendo volver o extraer un estado anterior del producto).

Ventajas

- Un completo historial de cambios a largo plazo de todos los archivos.
  - Esto quiere decir todos los cambios realizados por muchas personas a lo largo de los años. Los cambios incluyen la creación y la eliminación de los archivos, así como los cambios de sus contenidos. Que, quien y cuando se realizaron los cambios. Facilita volver a una versión anterior.
- Creación de ramas y fusiones.
  - La creación de una "rama" permite mantener múltiples flujos de trabajo independientes los unos de los otros al tiempo que ofrece la facilidad de volver a fusionar ese trabajo, lo que permite que los desarrolladores verifiquen que los cambios de cada rama no entran en conflicto. Facilita el trabajo colaborativo.
- Trazabilidad.
  - Ser capaz de trazar cada cambio que se hace en el software y conectarlo con un software de gestión de proyectos y seguimiento de errores, además de ser capaz de anotar cada cambio con un mensaje que describa el propósito y el objetivo del cambio, no solo ayuda con el análisis de la causa raíz y la recopilación de información.

Conceptos

- Versión
  - "Versión" es la forma particular que adopta un objeto en un contexto dado.
- Revisión
  - Desde el punto de vista de evolución, es la forma particular de un objeto en un instante dado. Se suele denominar "revisión".
- Configuración
  - Una "configuración" es una combinación de versiones particulares de los componentes (que pueden evolucionar individualmente) que forman un sistema consistente.
  - Desde el punto de vista de evolución, es el conjunto de las versiones de los objetos componentes en un instante dado

Conceptos

- Línea base
  - Llamaremos "línea base" a una configuración operativa del sistema software a partir del cual se pueden realizar cambios subsiguientes.
  - La evolución del sistema puede verse como evolución de la línea base
- Cambio
  - Un cambio representa una modificación específica a un archivo bajo control de versiones. La granularidad de la modificación considerada un cambio varía entre diferentes sistemas de control de versiones.
  - Un "cambio" es el paso de una versión de la línea base a la siguiente
  - Puede incluir modificaciones del contenido de algún componente, y/o modificaciones de la estructura del sistema, añadiendo o eliminando componentes

Conceptos

- Branch:
  - Una “ramificación”, bifurcación o variante es una bifurcación de la línea base u otra ramificación que a partir de ese momento evoluciona y se versiona por separado.
  - Las ramificaciones representan una variación espacial, mientras que las revisiones representan una variación temporal.
- Repositorio
  - El repositorio es el lugar en el que se almacenan los datos actualizados e históricos de cambios, a menudo en un servidor.
  - El repositorio permite ahorrar espacio de almacenamiento, evitando guardar por duplicado elementos comunes a varias versiones o configuraciones

Sistemas de Control de Versiones Manuales

- Un método de control de versiones, usado por muchas personas, es copiar los archivos a otro directorio (quizás indicando la fecha y hora en que lo hicieron, si son ingeniosos).
- Este método es muy común porque es muy sencillo, pero también es tremendamente propenso a errores.
- Es fácil olvidar en qué directorio te encuentras y guardar accidentalmente en el archivo equivocado o sobrescribir archivos que no querías.
- Consume mucho espacio al duplicar elementos que no se han modificado entre versiones.

Sistemas de Control de Versiones Centralizados

- El siguiente gran problema con el que se encuentran las personas es que necesitan colaborar con desarrolladores en otros sistemas. Los sistemas de Control de Versiones Centralizados (CVCS por sus siglas en inglés) fueron desarrollados para solucionar varias personas que necesitan colaborar.
- Estos sistemas, como CVS, Subversion y Perforce, tienen un único servidor que contiene todos los archivos versionados y varios clientes que descargan los archivos desde ese lugar central. Este ha sido el estándar para el control de versiones por muchos años.
- Esta configuración ofrece muchas ventajas, especialmente frente a VCS locales, permite trabajar de forma exclusiva: para poder realizar un cambio es necesario comunicar al repositorio el elemento que se desea modificar y el sistema se encargará de impedir que otro usuario pueda modificar dicho elemento, hasta que sea liberado.
- Sin embargo, esta configuración también tiene serias desventajas:
  - Es el punto único de fallo que representa el servidor centralizado.
  - Sin conexión al servidor nadie podrá colaborar o guardar cambios en archivos en los que hayan estado trabajando.

Sistemas de Control de Versiones Distribuidos

- Los sistemas de Control de Versiones Distribuidos o DVCS (como Git, Mercurial, Bazaar o Darcs), los clientes no solo descargan la última copia instantánea de los archivos, sino que se replica completamente el repositorio, existe un repositorio local y otro central.
- De esta manera, si un servidor deja de funcionar y estos sistemas estaban colaborando a través de él, cualquiera de los repositorios disponibles en los clientes puede ser copiado al servidor con el fin de restaurarlo. Cada clon es realmente una copia completa de todos los datos.
- Se puede trabajar de forma local. Permite operaciones más rápidas.
- Cada usuario modifica la copia local y cuando el usuario decide compartir los cambios el sistema automáticamente intenta combinar las diversas modificaciones. El principal problema es la posible aparición de conflictos que deban ser solucionados manualmente o las posibles inconsistencias que surjan al modificar el mismo fichero por varias personas no coordinadas.

https://git-scm.com/
https://git-scm.com/book/es/v2

GIT

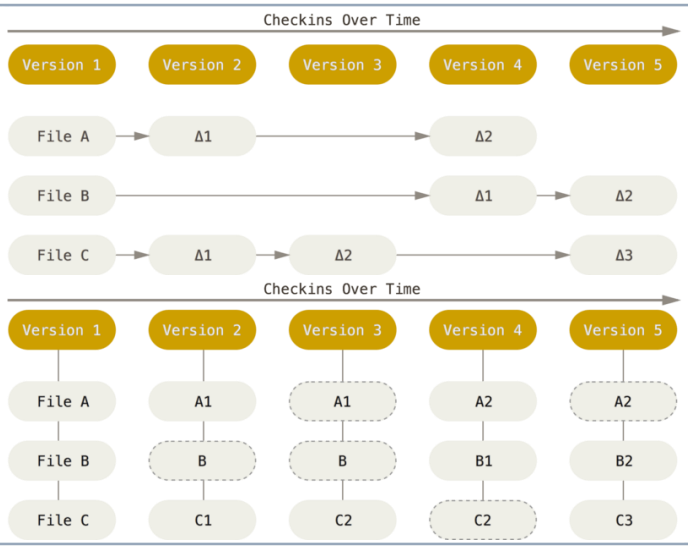
GIT

- Git (pronunciado "guit") 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.
- Git es, con diferencia, el sistema de control de versiones moderno más utilizado del mundo. Git es un proyecto de código abierto maduro y con un mantenimiento activo que desarrolló originalmente Linus Torvalds, el famoso creador del kernel del sistema operativo Linux, en 2005.
- Git es fácil de aprender y ocupa poco espacio con un rendimiento increíblemente rápido. Supera a las herramientas SCM como Subversion, CVS, Perforce y ClearCase con características como bifurcaciones locales económicas, áreas de preparación convenientes y múltiples flujos de trabajo. Funciona a la perfección en una amplia variedad de sistemas operativos e IDE (entornos de desarrollo integrados).
- No confundir con GitHub, que es una *forja* (plataforma de desarrollo colaborativo) para alojar proyectos utilizando el sistema de control de versiones Git, siendo la plataforma más importante de colaboración para proyectos Open Source.
- Instalación:
  - https://git-scm.com/

Fundamentos de Git

- La principal diferencia entre Git y cualquier otro VCS es la forma en la que manejan sus datos. Conceptualmente, la mayoría de los otros sistemas almacenan la información como una lista de cambios en los archivos, manejan la información que almacenan como un conjunto de archivos y las modificaciones hechas a cada uno de ellos a través del tiempo.
- Git maneja los datos como un conjunto de copias instantáneas de un sistema de archivos miniatura. Cada vez que se confirma un cambio o guarda el estado del proyecto en Git, él básicamente toma una foto del aspecto de todos los archivos en ese momento y guarda una referencia a esa copia instantánea. Para ser eficiente, si los archivos no se han modificado Git no almacena el archivo de nuevo, sino un enlace al archivo anterior idéntico que ya tiene almacenado. Git maneja los datos como una secuencia de copias instantáneas.
- La mayoría de las operaciones en Git sólo necesitan archivos y recursos locales para funcionar. Por lo general no se necesita información de ningún otro equipo de tu red.
- Todo en Git es verificado mediante una suma de comprobación (checksum) antes de ser almacenado, y es identificado a partir de ese momento mediante dicha suma. Esto significa que es imposible cambiar los contenidos de cualquier archivo o directorio sin que Git lo detecte.
- Cuando realizas acciones en Git, casi todas ellas sólo añaden información a la base de datos de Git. Es muy difícil hacer algo que no se pueda enmendar.

Instantáneas, no diferencias



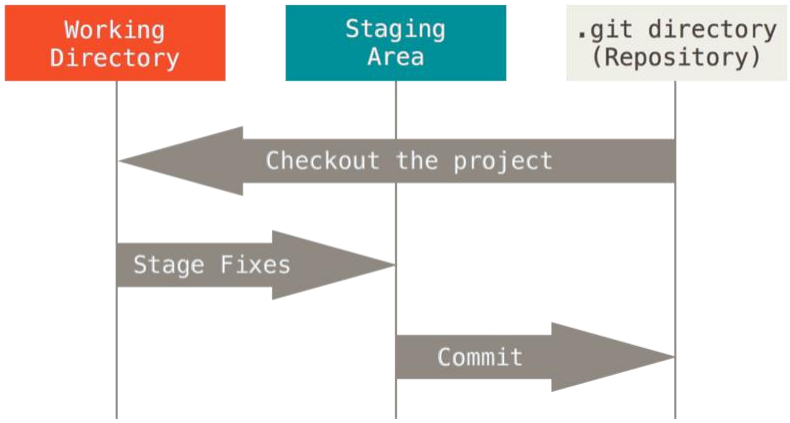
Estados

- Git tiene tres estados principales en los que se pueden encontrar tus archivos:
  - Modificado (modified): significa que has modificado el archivo pero todavía no lo has confirmado a tu base de datos.
  - Preparado (staged): significa que has marcado un archivo modificado en su versión actual para que vaya en tu próxima confirmación.
  - Confirmado (committed): significa que los datos están almacenados de manera segura en tu base de datos local.
- Esto nos lleva a las tres secciones principales de un proyecto de Git:
  - El directorio de Git (git directory) es donde se almacenan los metadatos y la base de datos de objetos para tu proyecto. Es la parte más importante de Git, y es lo que se copia cuando clonas un repositorio desde un repositorio central.
  - El directorio de trabajo (working directory) es una copia de una versión del proyecto. Estos archivos se sacan de la base de datos comprimida en el directorio de Git, y se colocan en disco para que los puedas usar o modificar.
  - El área de preparación (staging area) es un archivo, generalmente contenido en tu directorio de Git, que almacena información acerca de lo que va a ir en tu próxima confirmación. A veces se le denomina índice ("index"), pero se está convirtiendo en estándar el referirse a ella como el área de preparación.

Flujo de trabajo

- Git plantea una gran libertad en la forma de trabajar en torno a un proyecto. Sin embargo, para coordinar el trabajo de un grupo de personas en torno a un proyecto es necesario acordar como se va a trabajar con Git. A estos acuerdos se les llama flujo de trabajo. Un flujo de trabajo de Git es una fórmula o una recomendación acerca del uso de Git para realizar trabajo de forma uniforme y productiva. Los flujos de trabajo más populares son git-flow, GitHub-flow y One Flow.
- El flujo de trabajo básico en Git es algo así:
  1. Modificas una serie de archivos en tu directorio de trabajo.
  2. Preparas los archivos, añadiéndolos a tu área de preparación.
  3. Confirmas los cambios, lo que toma los archivos tal y como están en el área de preparación y almacena esa copia instantánea de manera permanente en tu directorio de Git.
  4. Periódicamente, se sincronizan el repositorio local con el repositorio central para que los cambios estén disponibles para el resto de los colaboradores, actuando también como copia de seguridad del repositorio local.
- Si una versión concreta de un archivo está en el directorio de Git, se considera confirmada (committed). Si ha sufrido cambios desde que se obtuvo del repositorio, pero ha sido añadida al área de preparación, está preparada (staged). Y si ha sufrido cambios desde que se obtuvo del repositorio, pero no se ha preparado, está modificada (modified).

Flujo de trabajo



Flujo de trabajo

- Git-Flow (Creado en 2010 por Vincent Driessen)
  - Es el flujo de trabajo más conocido. Está pensado para aquellos proyectos que tienen entregables y ciclos de desarrollo bien definidos. Está basado en dos grandes ramas con infinito tiempo de vida (ramas master y develop) y varias ramas de apoyo, unas orientadas al desarrollo de nuevas funcionalidades (ramas feature-*), otras al arreglo de errores (hotfix-*) y otras orientadas a la preparación de nuevas versiones de producción (ramas release-*).
- GitHub-Flow (Creado en 2011 por el equipo de GitHub)
  - Es la forma de trabajo sugerida por las funcionalidades propias de GitHub. Está centrado en un modelo de desarrollo iterativo y de despliegue constante. Está basado en cuatro principios:
    - Todo lo que está en la rama master está listo para ser puesto en producción
    - Para trabajar en algo nuevo, debes crear una nueva rama a partir de la rama master con un nombre descriptivo. El trabajo se irá integrando sobre esa rama en local y regularmente también a esa rama en el servidor
    - Cuando se necesite ayuda o información o cuando creemos que la rama está lista para integrarla en la rama master, se debe abrir una pull request (solicitud de integración de cambios).
    - Alguien debe revisar y visar los cambios para fusionarlos con la rama master
    - Los cambios integrados se pueden poner en producción.
  - GitHub intenta simplificar la gestión de ramas, trabajando directamente sobre la rama master y generando e integrando las distintas features directamente a esta rama.
- One Flow (Creado en 2015 por Adam Ruka)
  - En él cada nueva versión de producción está basada en la versión previa de producción. La mayor diferencia con Git Flow es que no tiene rama de desarrollo.

Terminología

- commit: un objeto Git, una instantánea de todo su repositorio comprimido en un SHA
- branch (rama): un puntero móvil ligero a una confirmación
- clone: una versión local de un repositorio, incluidas todas las confirmaciones y ramas
- remote: un repositorio común en un servidor central (como GitHub) que todos los miembros del equipo usan para intercambiar sus cambios
- fork: una copia de un repositorio del servidor central propiedad de un usuario diferente
- pull request (solicitud de extracción): un lugar para comparar y discutir las diferencias introducidas en una rama con revisiones, comentarios, pruebas integradas y más
- HEAD: que representa el directorio de trabajo actual, el puntero HEAD se puede mover a diferentes ramas, etiquetas o confirmaciones cuando se usa git checkout

Comandos

- Configurar: Configura la información del usuario para todos los respositorios locales
  - $ git config --global user.name "[name]"
    - Establece el nombre que estará asociado a tus commits
  - $ git config --global user.email "[email address]"
    - Establece el e-mail que estará asociado a sus commits
- Crear repositorios: Inicializa un nuevo repositorio u obtiene uno de una URL existente
  - $ git init [project-name]
    - Crea un nuevo repositorio local con el nombre especificado
  - $ git remote add origin [url]
    - Especifica el repositorio remoto para su repositorio local
  - $ git clone [url]
    - Descarga un proyecto y toda su historial de versiones
- Suprimir el seguimiento de cambios: Un archivo de texto llamado .gitignore suprime la creación accidental de versiones para archivos y rutas que concuerdan con los patrones especificados
  - $ git ls-files --others --ignored --exclude-standard
    - Enumera todos los archivos ignorados en este proyecto

Comandos

- Efectuar cambios
  - $ git status
    - Enumera todos los archivos nuevos o modificados de los cuales se van a guardar cambios
  - $ git diff
    - Muestra las diferencias entre archivos que no se han enviado aún al área de espera
  - $ git add [file]
    - Guarda el estado del archivo en preparación para realizar un commit
  - $ git diff --staged
    - Muestra las diferencias del archivo entre el área de espera y la última versión del archivo
  - $ git reset [file]
    - Mueve el archivo del área de espera, pero preserva su contenido
  - $ git commit -m"[descriptive message]"
    - Registra los cambios del archivo permanentemente en el historial de versiones
  - $ git rm [file]
    - Borra el archivo del directorio activo y lo pone en el área de espera en un estado de eliminación
  - $ git rm --cached [file]
    - Retira el archivo del historial de control de versiones, pero preserva el archivo a nivel local
  - $ git mv [file-original] [file-renamed]
    - Cambia el nombre del archivo y lo prepara para ser guardado

Comandos

- Sincronizar cambios: Registrar un marcador para un repositorio e intercambiar historial de versiones
  - $ git fetch [bookmark]
    - Descarga todo el historial del marcador del repositorio
  - $ git merge [bookmark]/[branch]
    - Combina la rama del marcador con la rama local actual
  - $ git push [alias] [branch]
    - Sube todos los commits de la rama local al repositorio central
  - $ git pull
    - Descarga el historial del marcador e incorpora cambios
- Branches (cambios grupales): Nombra una serie de commits y combina esfuerzos ya completados
  - $ git branch
    - Enumera todas las ramas en el repositorio actual
  - $ git branch [branch-name]
    - Crea una nueva rama
  - $ git checkout [branch-name]
    - Cambia a la rama especificada y actualiza el directorio activo
  - $ git merge [branch-name]
    - Combina el historial de la rama especificada con la rama actual
  - $ git branch -d [branch-name]
    - Borra la rama especificada

Comandos

- Repasar historial: Navega e inspecciona la evolución de los archivos de proyecto
  - $ git log
    - Enumera el historial de versiones para la rama actual
  - $ git log --follow [file]
    - Enumera el historial de versiones para el archivo, incluidos los cambios de nombre
  - $ git diff [first-branch]...[second-branch]
    - Muestra las diferencias de contenido entre dos ramas
  - $ git show [commit]
    - Produce metadatos y cambios de contenido del commit especificado
- Rehacer commits: Borra errores y elabora un historial de reemplazo
  - $ git reset [commit]
    - Deshace todos los commits después de [commit], preservando los cambios localmente
  - $ git reset --hard [commit]
    - Desecha todo el historial y regresa al commit especificado

Comandos

- Guardar fragmentos: Almacena y restaura cambios incompletos
  - $ git stash
    - Almacena temporalmente todos los archivos modificados de los cuales se tiene al menos una versión guardada
  - $ git stash pop
    - Restaura los archivos guardados más recientemente
  - $ git stash list
    - Enumera todos los grupos de cambios que están guardados temporalmente
  - $ git stash drop
    - Elimina el grupo de cambios más reciente que se encuentra guardado temporalmente

GitHub

- echo "# Mi repositorio" >> README.md
- git init
- git add .
- git commit -m "initial commit"
- git remote add origin https://github.com/myuser/myrepository.git
- git branch -M main
- git push -u origin main

- https://github.com/github/gitignore