DESARROLLO WEB FULLSTACK

con Python y JavaScript

polotic misiones

polotic misiones

Control de Versiones

Introducción

Cuando los proyectos crecen y se hacen mas complejos, intervienen mayor cantidad de desarrolladores y actores de la industria. Lo que hace que se necesite algún tipo de control de código.

Pasos en el Desarrollo de Software

Análisis de Requerimientos

Diseño de Alto nivel

Diseño de Bajo nivel

Implementacion

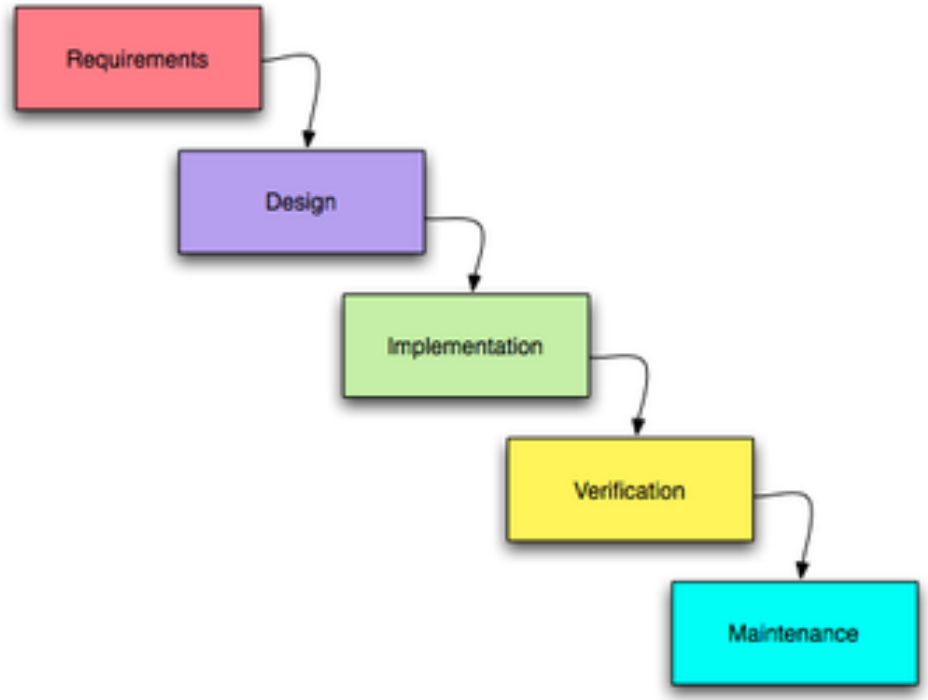
Tests y pruebas de unidad

Integracion

Testeo sistemático

Despliegue

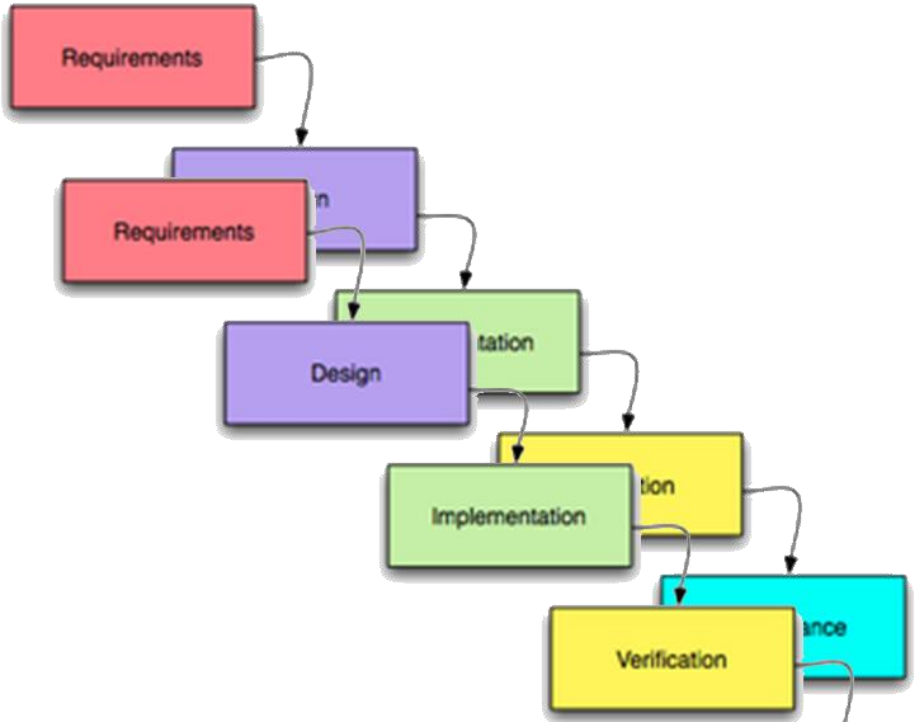
Mantenimiento



Proyectos Concurrentes

- Por lo general, se agregan nuevas funciones a la próxima versión
- Mientras que, al mismo tiempo, los defectos deben corregirse en las versiones existentes.

- Durante el despliegue y el mantenimiento esto se hace frecuentemente

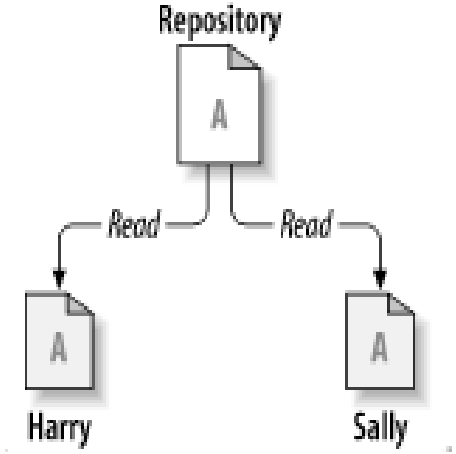


Trabajo en Conjunto

- Todos los miembros del equipo trabajan en el mismo código y desarrollan sus respectivas partes
- Un ingeniero puede ser responsable de toda una clase.
- O solo algunos métodos dentro de una clase en particular

Archivos Compartidos

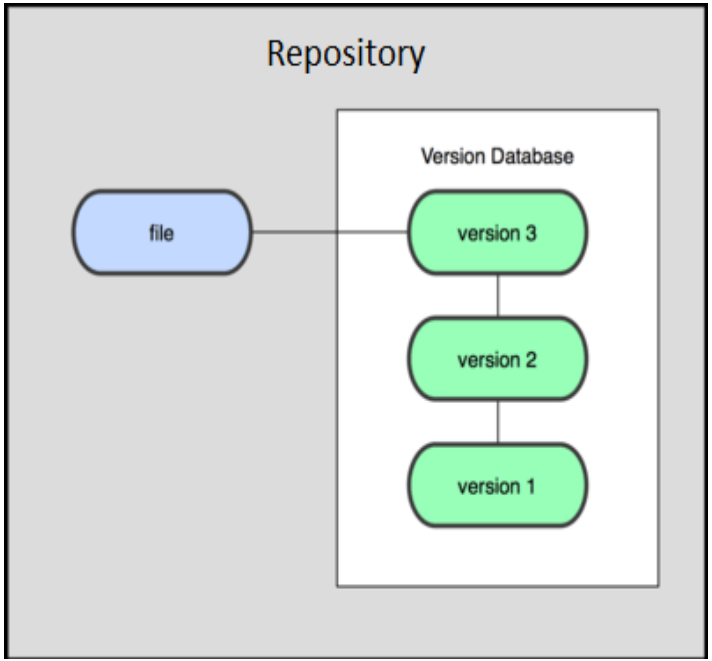
Los archivos compartidos se pueden guardar en algún tipo de repositorio donde varias personas pueden acceder a ellos y trabajar en ellos.



Harry y Sally obtienen sus propias **copias de trabajo** locales respectivas del archivo maestro en el repositorio

¿Por qué no sirven la compartición común y las nubes?

- ... pero las unidades de red privada USB/SD no mantienen un historial de revisión
- Las unidades en la nube (por ejemplo, Google Drive, Dropbox, iCould / Time Machine) mantienen un historial (*limitado*) de las diversas revisiones de un archivo
  - *Google Drive: mantiene las últimas 100 revisiones o los últimos 30 días*
- Podría ser viable para dos personas, ¿qué tal 5 o 10?



Propuestas existen desde hace muchos años

- SCCS, 1972 (IBM)
- RCS, 1982 (Unix)
  - SourceSafe 1995 (Microsoft)
- CVS, 1986 (multi-plataforma)
- SVN, 2000 (multi-plataforma)
- Git y Mercurial, 2005 (multi-plataforma)

La forma en que funcionan estos sistemas ha evolucionado a lo largo de los años.

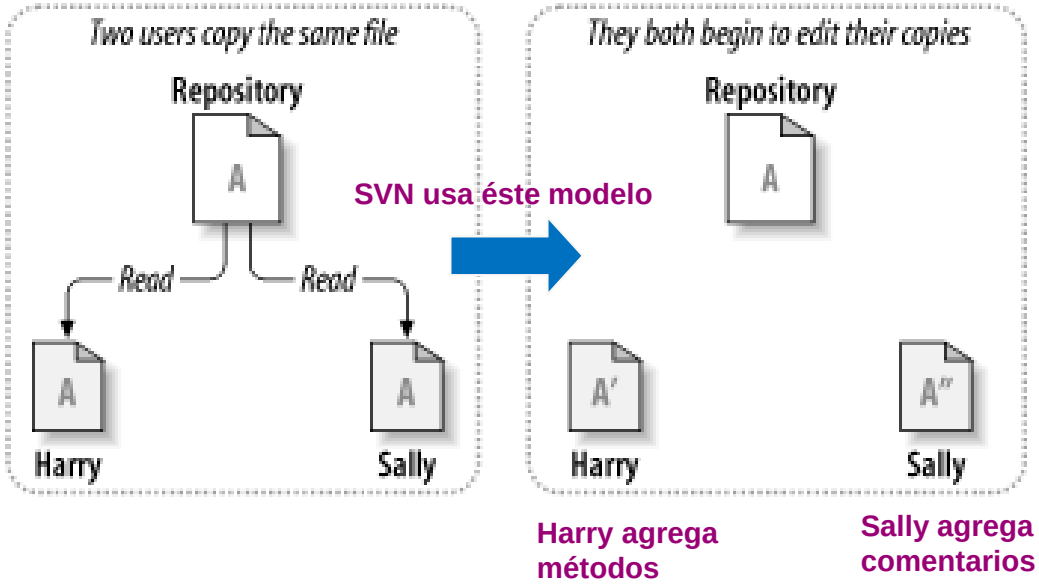
Características de los Sist. de Control de Versiones

- ✓ Se realiza un seguimiento de todos los cambios en un archivo
  - ✓ Los historiales de versiones muestran **quién**, **qué**, **cuándo**
- ✓ Los cambios se pueden revertir (rebobinar a cualquier versión anterior)
- ✓ Los cambios entre revisiones son fáciles de identificar

Solución de tipo Copy-Modify-Merge

Dos usuarios copian el mismo archivo

Ambos comienzan a editar sus copias

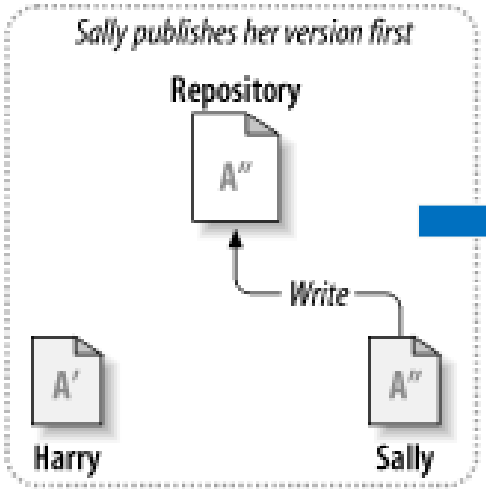


Harry agrega métodos

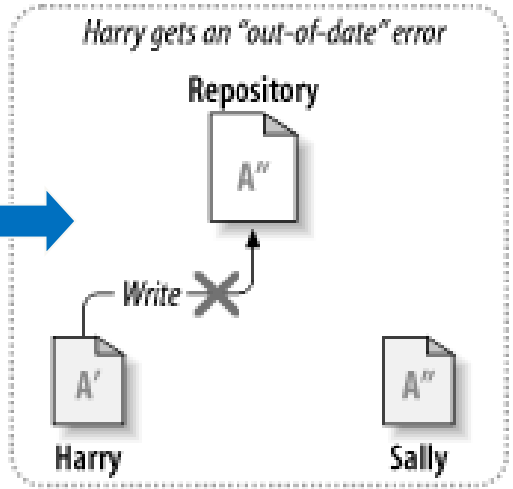
Sally agrega comentarios

El Repositorio reconoce conflictos

Sally publica su version primero



Harry obtiene un error de caducidad

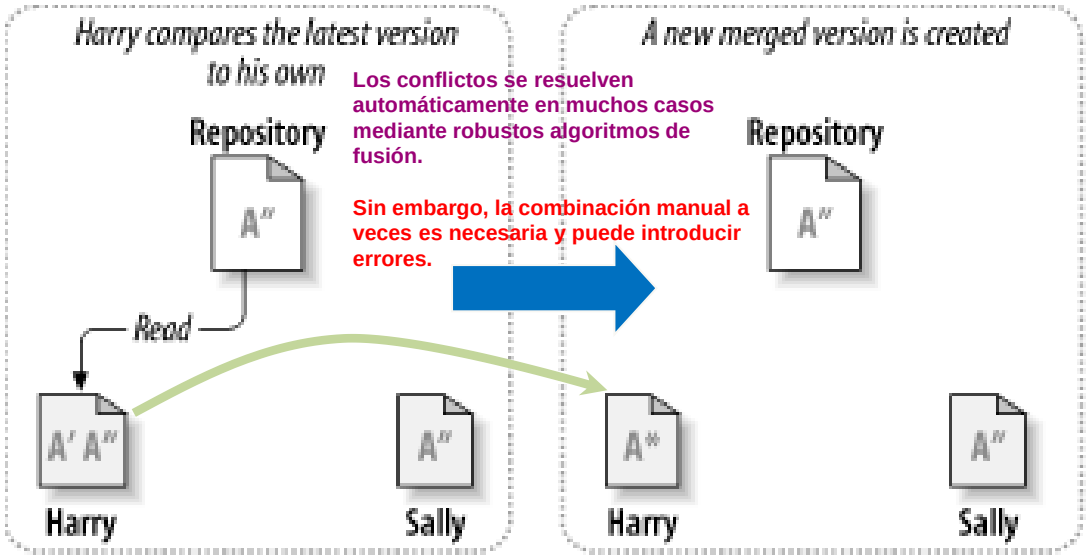


Harry no puede
Escribir sus cambios

Las versiones conflictivas se fusionan (merge)

Harry compara la ultima version con la suya

Se crea una nueva y fusionada version



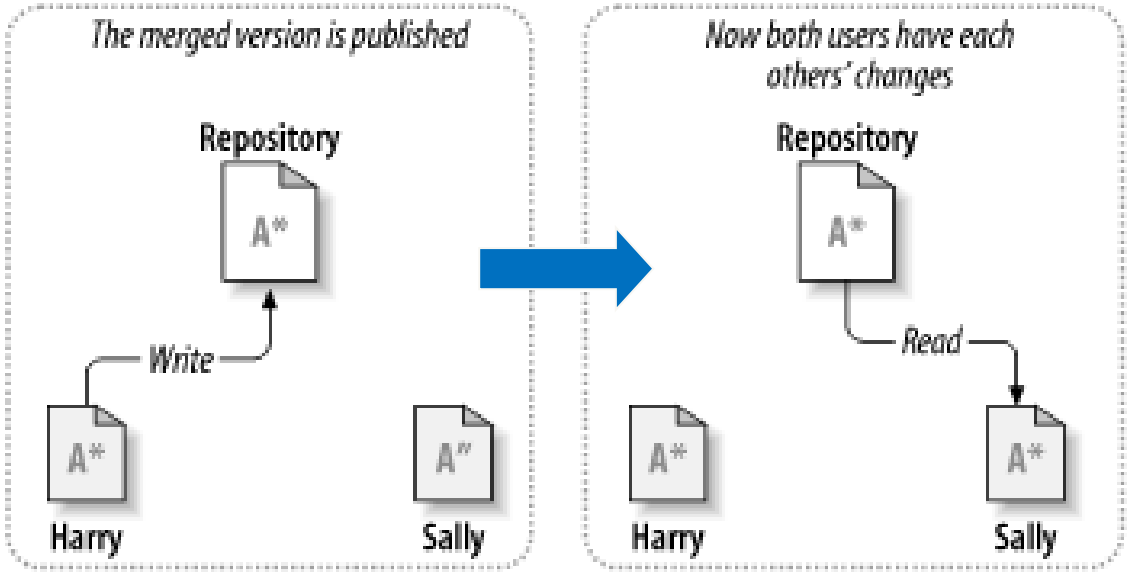
Los conflictos se resuelven automáticamente en muchos casos mediante robustos algoritmos de fusión.

Sin embargo, la combinación manual a veces es necesaria y puede introducir errores.

El repositorio mantiene a ambos usuarios sincronizados

Se publica la version fusionada

Ahora ambos usuarios tienen los cambios de los demás

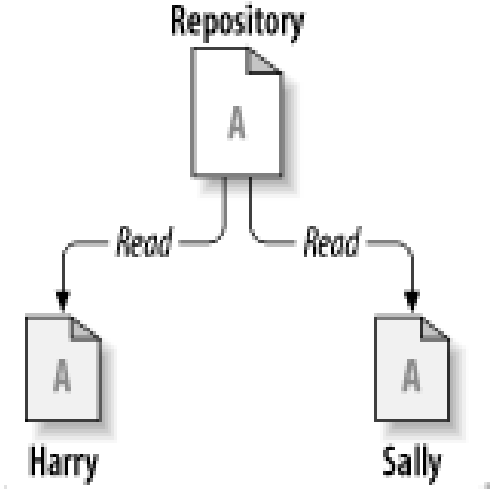


Copy-Modify-Merge

- ✓ Existe un sistema en control de versiones para mantener siempre a los usuarios sincronizados que se denomina Copiar-Modificar-Fusionar
- ✓ Es generalmente fácil de usar y administrar
- ✓ Los usuarios pueden trabajar en paralelo
- ✓ La mayoría de las veces, los cambios simultáneos no se solapan
    - Por lo general, las personas no editan exactamente el mismo código de forma simultánea, por lo que la combinación (merge) se realiza automáticamente en muchos casos.
    - La cantidad de tiempo que se dedica a resolver conflictos es casi siempre menor que el tiempo que se necesitaría dedicar si un archivo estuviera bloqueado.

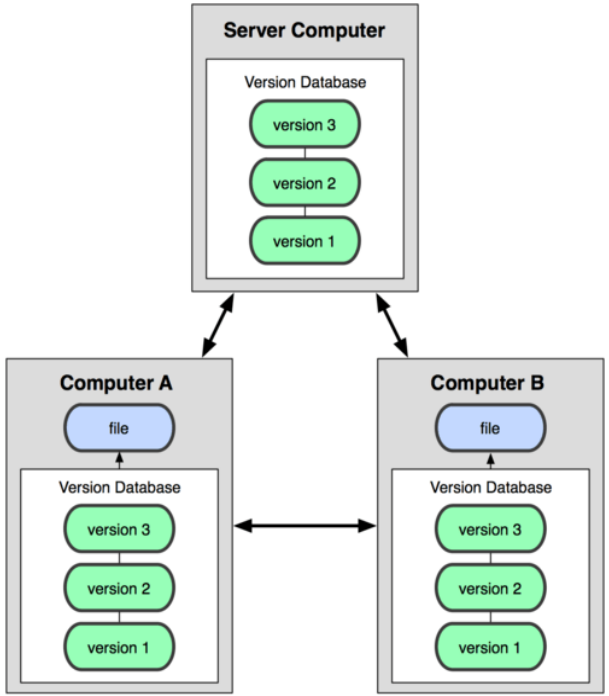
El principal inconveniente de Subversion es la naturaleza centralizada del repositorio

- ✓ Normalmente, un repositorio se aloja en un servidor.
- ✓ El historial de versiones está en el servidor, no en su PC local
- ✓ Si el servidor o la red no están disponibles, todos se quedan atascados con la copia de trabajo que tienen.

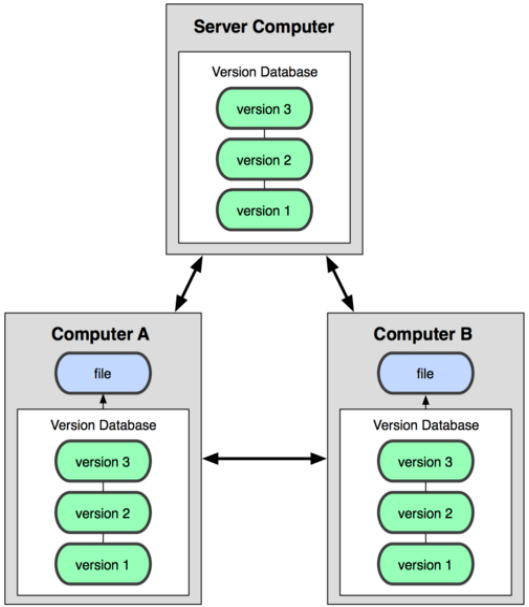


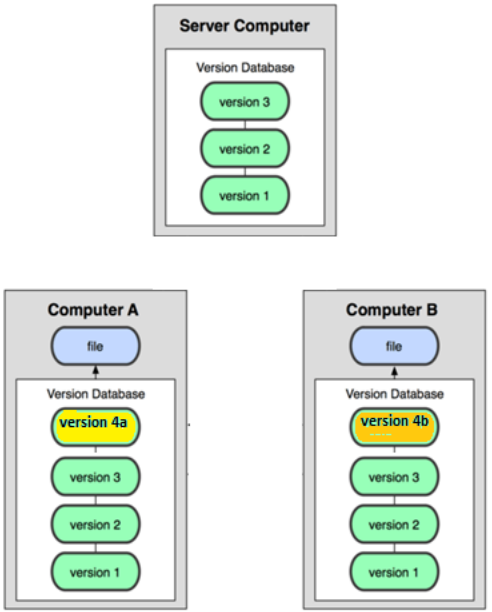
Control de Versiones Distribuida

- ✓ Un repositorio "maestro remoto" normalmente se aloja en un servidor.
- ✓ Los clones del repositorio remoto se mantienen en la computadora de cada usuario.
- ✓ Si el servidor deja de funcionar, los usuarios pueden seguir compartiendo código (siempre que puedan conectarse a través de una red) sincronizándose con los repositorios de los demás.
- ✓ Si la red no está disponible, los usuarios pueden continuar leyendo y escribiendo en su propio repositorio local, sincronizándose con otros más tarde



Control de Versiones Distribuida – Parte 1

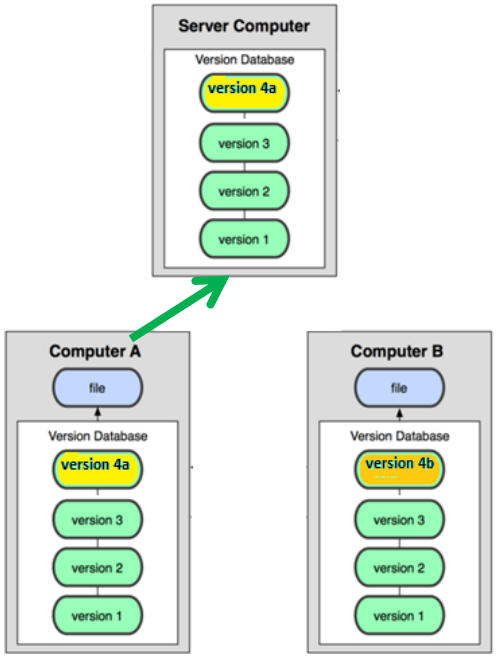




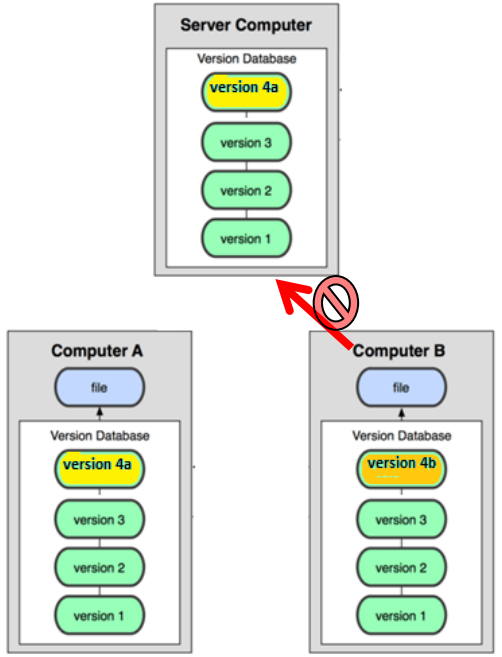
1. Inicialmente, los repositorios de ambos usuarios se sincronizan con el maestro remoto; Todos los archivos son idénticos.

2. Ambos usuarios editan el mismo archivo y sus repositorios locales reflejan sus cambios.

Control de Versiones Distribuida – Parte 2

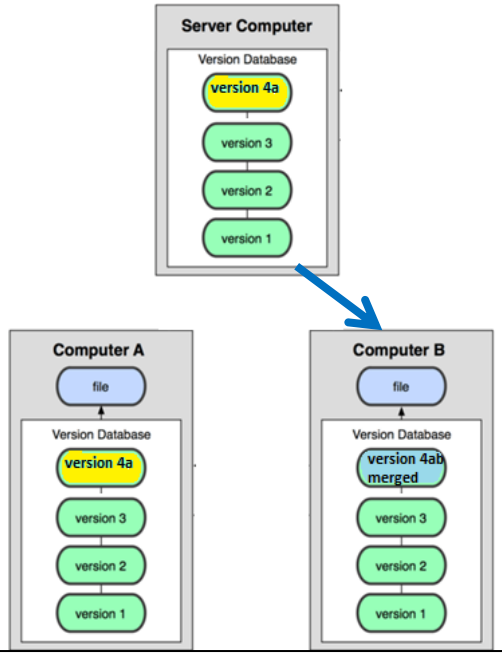


3. El usuario A hace un "push" de su repositorio al remoto maestro (master) para sincronizarlo; esto tiene éxito.

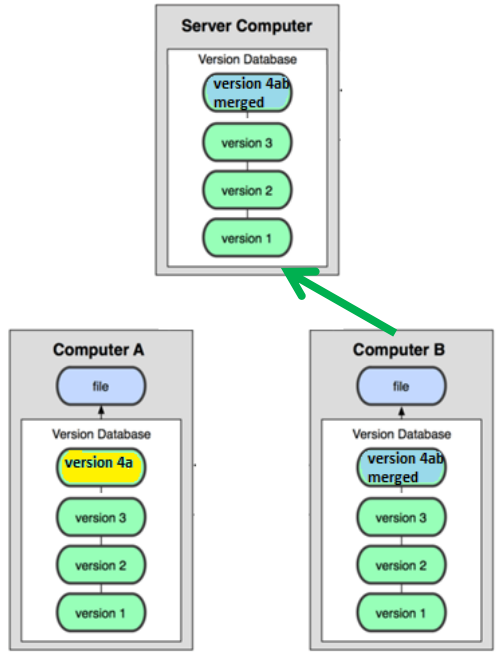


4. El usuario B intenta hacer un push (enviar) su repositorio al remoto para sincronizarlo; esto falla debido al push anterior del Usuario A. El remoto no cambia.

Control de Versiones Distribuida – Parte 3

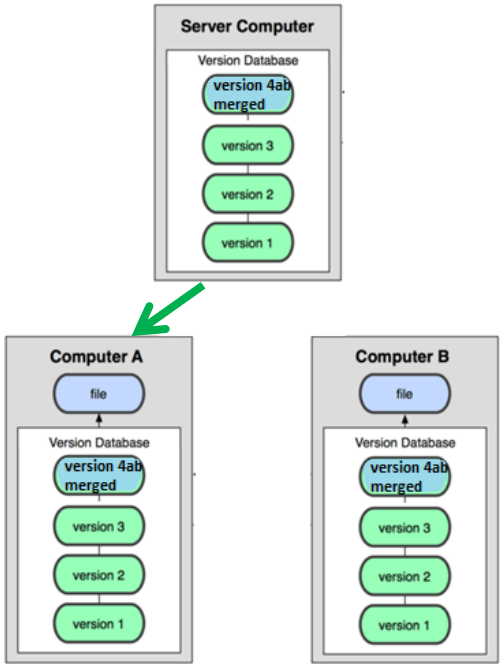


5. El usuario B hace un “pull“ de las actualizaciones del usuario A del remoto master, haciendo un merge de A y B juntos. La fusión es automática con algunas excepciones.



6. El usuario B hace un “push" de su repositorio al remoto para sincronizarlo; esto ahora tiene éxito

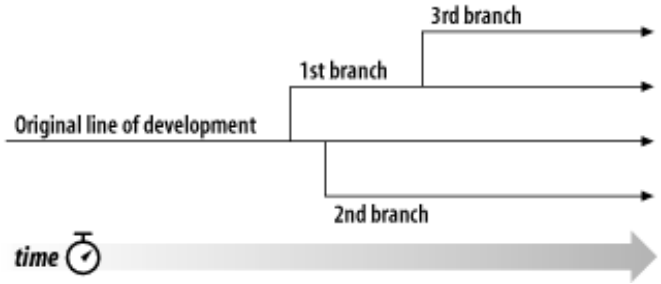
Control de Versiones Distribuida – Parte 4



7. El usuario B hace un “pull" del remoto y todos están sincronizados nuevamente.

Seguimiento de Versiones Paralelas

Los repositorios también pueden gestionar y realizar un seguimiento de las diferencias entre las revisiones paralelas de un documento.



Por lo general, un producto de software se someterá a un desarrollo paralelo en diferentes ramas o **branches** (por ejemplo, para funciones nuevas / futuras) mientras que el desarrollo primario se lleva a cabo en la rama principal (master).

Sistemas basados en Git que puedes usar

Hay muchos sistemas de hosting online de control de versiones, entre ellos los mas famosos son:

Sistemas basados en Git que puedes usar

GITHUB

- Puedes adjuntar hitos, proyectos y etiquetas para proporcionar contexto al desarrollo
- Puedes suscribirte para recibir una notificación cuando cambia el pull request
- Asignar pull requests a compañeros de equipo
- Diferencia de cambios entre la fuente (source) y los branches
- Hacer merge o eliminar branch de origen con un clic
- Integración con herramientas externas de integración continua
- Plantillas de pull requests para garantizar que se sigan las pautas de contribución
- Conversaciones sobre partes del código que requieren resolución.
- Revisiones requeridas para asegurar que cada pull request sea firmada por alguien antes de un merge.

Sistemas basados en Git que puedes usar

BITBUCKET

- Asignar pull request a compañeros de equipo
- Editor de texto avanzado para comentarios y descripciones.
- Suscríbase para recibir una notificación cuando cambie el pull request
- Diferencia de cambios entre el branch fuente y base
- Hacer merge o eliminar branch de origen con un clic
- Integración con herramientas externas de integración continua
- Opción para requerir la aprobación del revisor antes de hacer un merge.

Sistemas basados en Git que puedes usar

GITLAB

- Asignar merge requests a compañeros de equipo
- Indicador WIP (Work In Progress) para abrir merge requests antes de que esten listos para merge
- Integración con hitos / etiquetas para el contexto de merge requests
- Los miembros del equipo pueden suscribirse para recibir una notificación cuando su request de merge ha sido realizada
- Diferencia de cambios entre el branch fuente y base
- Integración con herramientas externas de integración continua
- Combinar y eliminar branch de origen con un clic

Sistemas basados en Git que puedes usar

polotic misiones

COMPARACIÓN

Free Plans	Public Repos	Private Repos	Collaborators	Storage Space	Hosting	Support
GitHub Public	Unlimited	0	Unlimited	N / A	Cloud	Email / Forum
Bitbucket Small teams	Unlimited	Unlimited (1Gb /project)	5	N / A	Cloud	Email / Forum
GitLab Cloud Hosted	Unlimited	Unlimited (10Gb / Project)	Unlimited	Unlimited	Cloud	Forum
GitLab Community Edition	Unlimited	Unlimited	Unlimited	N / A	Self-hosted	Forum
Coding Free Plan	Unlimited	Unlimited	10	1GB	Cloud	Email /Forum

Instalar Git

Lo que primero debes hacer para trabajar con el control de versiones de Git es instalarlo en tu sistema operativo. Para ello te diriges a la pagina web de descargas de Git (https://git-scm.com/downloads) y descargas la versión para tu sistema operativo y la instalas.

En este curso, como ejemplo, usaremos GitLab

Si bien tienes instalado Git y puedes usar sus comandos, necesitarás un lugar seguro donde alojar tu código y compartirlo con otros. En nuestro curso usaremos GitLab, si quieres puedes instalarlo tu en un servidor o usar sus servicios de hosting de proyectos.

En este curso, como ejemplo, usaremos GitLab

GitLab nos permite hostear nuestro código, para ello tenemos que:

- Crearnos una cuenta en GitLab
- Crear un proyecto haciendo click en el botón azul New Project (Nuevo Proyecto).
- Crear un proyecto en blanco (Blank Project)
- Completar el campo de nombre, url que tendrá nuestro proyecto, generalmente usuario o grupo seguido de un slug (apodo para el proyecto que sea breve y conciso) y la visibilidad como Privada.
- ¡Listo! Ahora debemos crear un respositorio para nuestro proyecto.

GitLab

Ahora clonamos un repositorio de GitLab en nuestra computadora local

- En el terminal escribimos git clone `https://gitlab.com/tusuario/sistemadeprueba.git`
- Creamos un nuevo documento dentro de nuestra nueva carpeta de prueba (`echo $null >> miscript.py (en Windows) o  touch miscript.py` (en Unix))
- Agregamos el archivo con `git add miscript.py`
- Preparamos el archivo para subirlo con `git commit -m "add miscript"`
- Lo subimos al repositorio con el comando `git push -u origin master`
- ¡Listo! Subiste tu primer archivo de código al repositorio online.

Git - Commits

- ✓ El comando COMMIT lo que hace es tomar una snapshot de tu código (como un pantallazo de su estado actual).
- ✓ Puedes usar git status para comparar tu código con el código remoto.
- ✓ El comando PUSH define que estás listo para subir los cambios.
- ✓ Si solo has cambiado archivos existentes y no has creado otros nuevos, en lugar de usar `git add.` y luego `git commit ...`, podemos condensar esto en un comando: `git commit -am "Algun mensaje"`. Este comando confirmará todos los cambios que realizaste.
- ✓ A veces, el repositorio remoto en GitLab estará más actualizado que la versión local. En este caso, primero desearás hacer `commit` de los cambios y luego ejecutar `git pull` para extraer los cambios remotos de tu repositorio.

Git – Conflictos con Merges

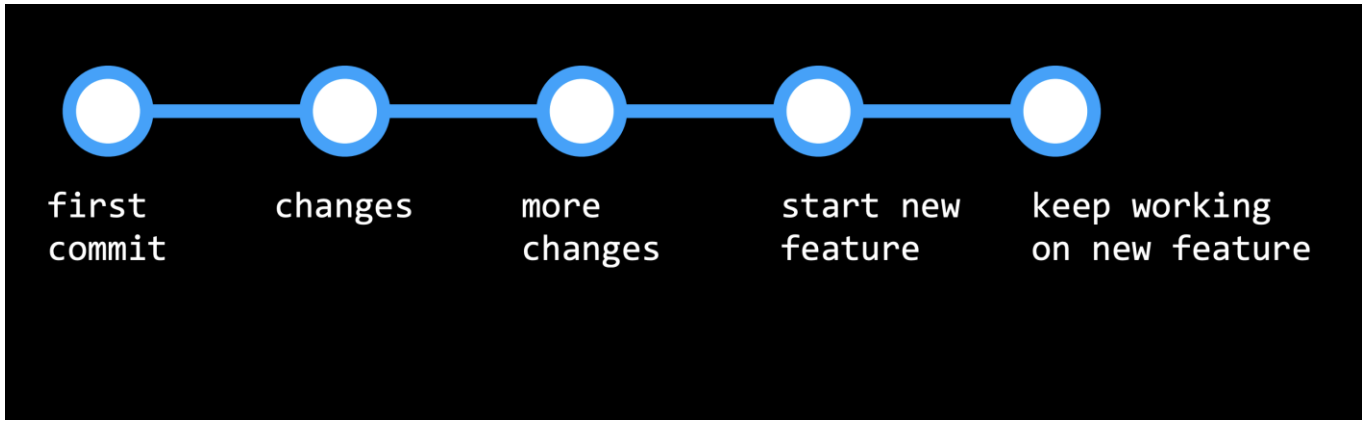
- ✓ Un problema que puede surgir cuando se trabaja con Git, especialmente cuando se colabora con otras personas, es algo llamado conflicto de merge.
- ✓ Un conflicto de merge se produce cuando dos personas intentan cambiar un archivo de formas que entran en conflicto entre sí.
- ✓ Esto suele ocurrir cuando intentas hacer `git push` o `git pull`. Cuando esto suceda, Git cambiará automáticamente el archivo a un formato que describa claramente cuál es el conflicto. Tendrás que aceptar o rechazar los cambios manualmente.
- ✓ Otro comando git potencialmente útil es `git log`, que le brinda un historial de todos tus `commits` en ese repositorio.

Git – Conflictos con Merges

- ✓ Potencialmente aún más útil, si te das cuenta de que has cometido un error, puedes volver a un `commit` anterior utilizando el comando `git reset` de una de estas dos formas:
- ✓ `git reset --hard <commit>` revierte tu código exactamente como estaba después de el commit especificado. Para especificar el commit, usa el hash asociado al commit que se puede encontrar usando `git log`.
- ✓ `git reset --hard origin / master` revierte tu código a la versión actualmente almacenada en línea en GitLab.

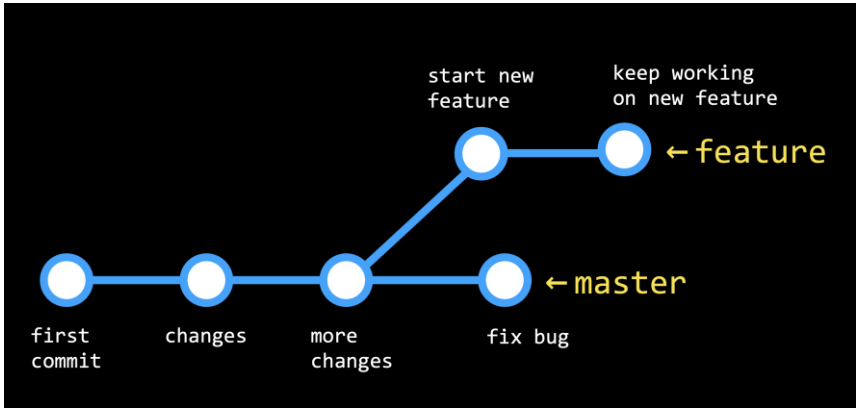
Git – Branching

- ✓ Después de haber estado trabajando en un proyecto durante algún tiempo, puedes decidir que deseas agregar una función adicional. Por el momento, es posible que solo realicemos cambios en esta nueva función, como se muestra en el gráfico a continuación.



Git – Branching

- ✓ Pero esto podría volverse problemático si luego descubrimos un error en nuestro código original y queremos revertir sin cambiar la nueva característica.
- ✓ Aquí es donde la ramificación puede resultar realmente útil.
- ✓ La ramificación (branching) es un método para moverse en una nueva rama al crear una nueva funcionalidad (feature), y solo combinar esta nueva funcionalidad con la parte principal de tu código (master por ej), o la rama principal, una vez que hayas terminado. Este flujo de trabajo se parecerá más al gráfico siguiente:



Git – Branching

- ✓ La rama que estás mirando actualmente está determinada por HEAD, que apunta a una de las dos ramas. De forma predeterminada, HEAD apunta a la rama master, pero también podemos ver otras ramas.
- ✓ Ahora, veamos cómo implementamos realmente la ramificación en nuestros repositorios de git:
- ✓ Ejecutas `git branch` para ver en qué rama estás trabajando actualmente, que tendrá un asterisco a la izquierda de su nombre.

1. Para crear una nueva rama, ejecutaremos `git checkout -b <nuevo nombre de rama>`
2. Cambia entre ramas usando el comando `git checkout <nombre de rama>` y has commit de los cambios en cada rama.
3. Cuando estemos listos para fusionar nuestras dos ramas, revisaremos la rama que deseamos mantener (casi siempre la rama master) y luego ejecutaremos el comando `git merge <otro nombre de rama>`. Esto se trata de manera similar a un `push` o `pull`, y pueden aparecer conflictos de `merge`.

Git – Otras Cuestiones Adicionales

FORKING

- ✓ Si tu quieres ser el propietario de algún código publico de otra persona entonces haces lo que se denomina como fork. El fork hace una copia exacta del código en tu propio repositorio pero ahora tu eres el dueño.

PULL REQUEST

- ✓ Puede suceder que te hayas bajado el código de alguien mediante un fork (como por ejemplo el de Bootstrap) y le hayas hecho cambios que lo han mejorado y quieres que se una al desarrollo principal. Entonces lo que haces es un pull request de tu código hacia el repositorio oficial (de, por ejemplo, Bootstrap). Si la gente de Bootstrap evalua tus cambios y cree que es un valor agregado al software entonces pueden llegar a aceptar tu pull request y se fusionara. Éste tipo de desarrollos es uno de los pilares fundamentales del desarrollo de software de código abierto.

Trabajemos Juntos

Llegó la hora del desafío:

- Create una cuenta en GitLab
- Sube algún script que hayas hecho en los ejemplos o desafíos anteriores al repositorio remoto.
- Intenta modificar ese código, agregando o sacando líneas en tu versión local. Luego vuélvelo a subir haciendo un push.

¡HASTA LA *próxima!*

www.polotic.misiones.gob.ar

/poloticmisiones

polotic misiones

Gobierno de Misiones