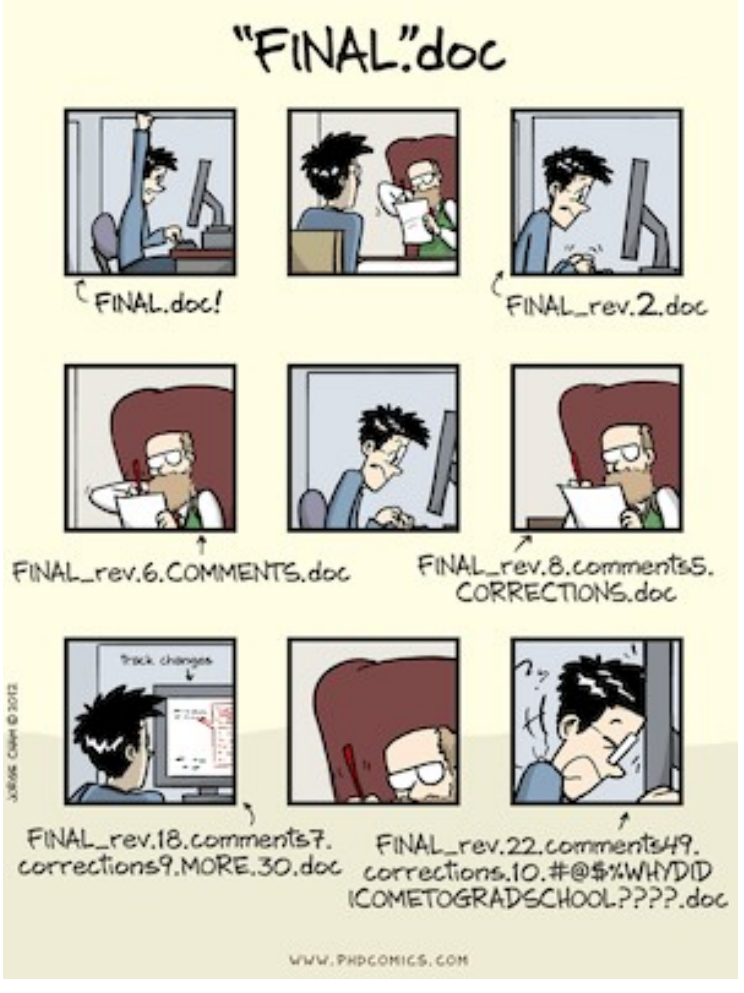
Git - Github

Introducción

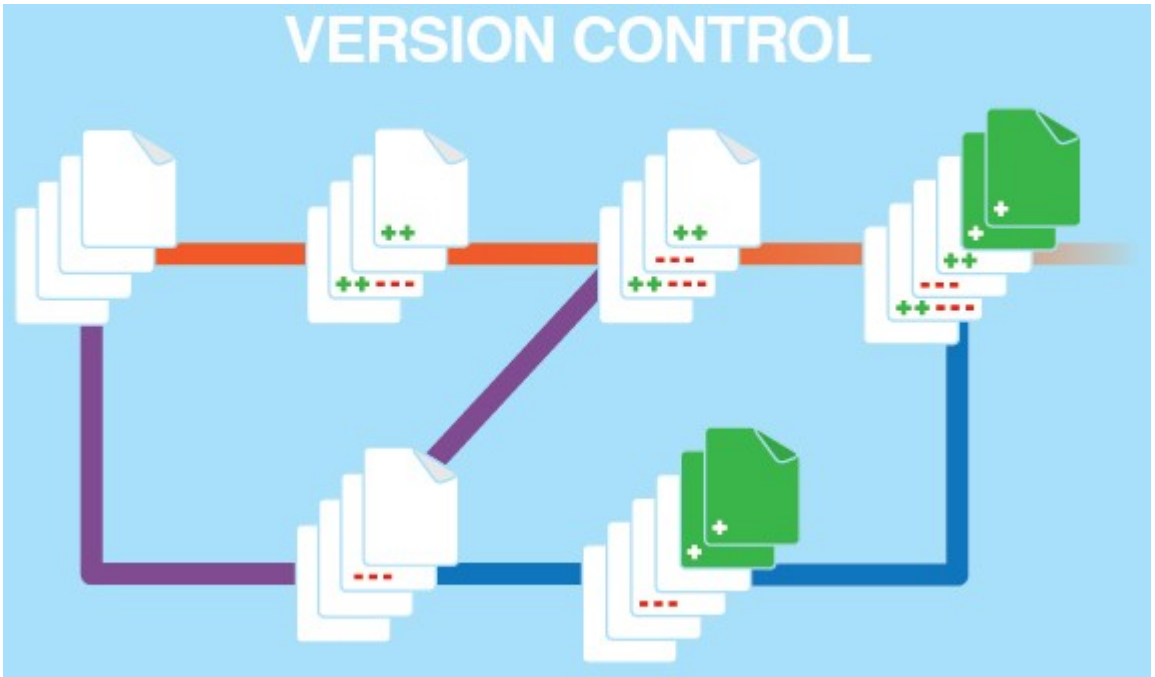
La **reproducibilidad**, entendida como la capacidad que tienen algunos trabajos o proyectos de ser recreados de forma independiente a partir de los mismos datos y el mismo código que utilizó el equipo original , es una condición *sine qua non* del método científico

Control de Versiones



¿Qué es el Control de Versiones? (I)

El control de versiones es un sistema que registra los cambios en un archivo o conjunto de archivos a lo largo del tiempo para que pueda recuperar versiones específicas más adelante.



¿Qué es el Control de Versiones? (II)

- Gestión de ficheros a lo largo del tiempo
  - Evolución del trabajo
- Gestión del versionado de los ficheros
  - Si un archivo se corrompe o hemos cometido un fallo volvemos atrás
- Mecanismo para compartir ficheros
- Habitualmente tenemos nuestro propio mecanismo y modelo de trabajo
  - Versionado: Documento.docx, Documento_v2.docx
  - Herramientas: Dropbox, Adjunto correo.

¿Por qué un Sistema de Control de Versiones?

- Las metodologías/mecanismos particulares no escalan para proyectos de desarrollo
- Un SCV permite
  - Crear copias de seguridad y restaurarlas
  - Sincronizar (mantener al día) a los desarrolladores respecto a la última versión de desarrollo
- Deshacer cambios
  - Tanto problemas puntuales, como problemas introducidos hace tiempo
- Gestionar la autoría del código
- Realizar pruebas (aisladas)
  - Simples o utilizando el mecanismo de branches/merges

Vocabulario de trabajo con los SCV (I)

Elementos básicos

- Repositorio
  - Almacén de que guarda toda la información del proyecto.
  - Habitualmente tiene estructura de árbol.
- Servidor
  - Máquina donde está alojado el Repositorio.
- Working Copy/Working Set (Copia de trabajo)
  - Copia local donde el desarrollador trabaja.
- Trunk/Main/master (Rama principal)
  - Localización dentro del repositorio que contiene la rama principal de desarrollo.

Vocabulario de trabajo con los SCV (II)

Operaciones básicas (I)

- Add
  - Añade un archivo para que sea rastreado por el SCV.
- Revisión
  - Versión de un archivo/directorio dentro del SCV
- Head
  - Última versión del repositorio (completo o de una rama)
- Check out
  - Creación de una copia de trabajo que rastrea un repositorio
- Check in / Commits
  - Envío de cambios locales al repositorio
  - Como resultado cambia la versión del archivo(s)/repositorio
- Mensaje de Check in/log
  - Todo Check in tiene asociado un mensaje que describe la finalidad del cambio
  - Puede estar asociado al un sistema de gestión de incidencias

Vocabulario de trabajo con los SCV (III)

Operaciones básicas (II)

- Log (Historia)
  - Permite visualizar/revisar la lista de cambios de un archivo/repositorio
- Update/Syncronize/fetch&pull (Actualizar)
  - Sincroniza la copia de trabajo con la última versión que existe en el repositorio.
- Revert/Reset (Deshacer)
  - Permite deshacer los cambios realizados en la copia de trabajo y dejar el archivo/recurso en el último estado conocido del repositorio.

Vocabulario de trabajo con los SCV (IV)

Operaciones Avanzadas

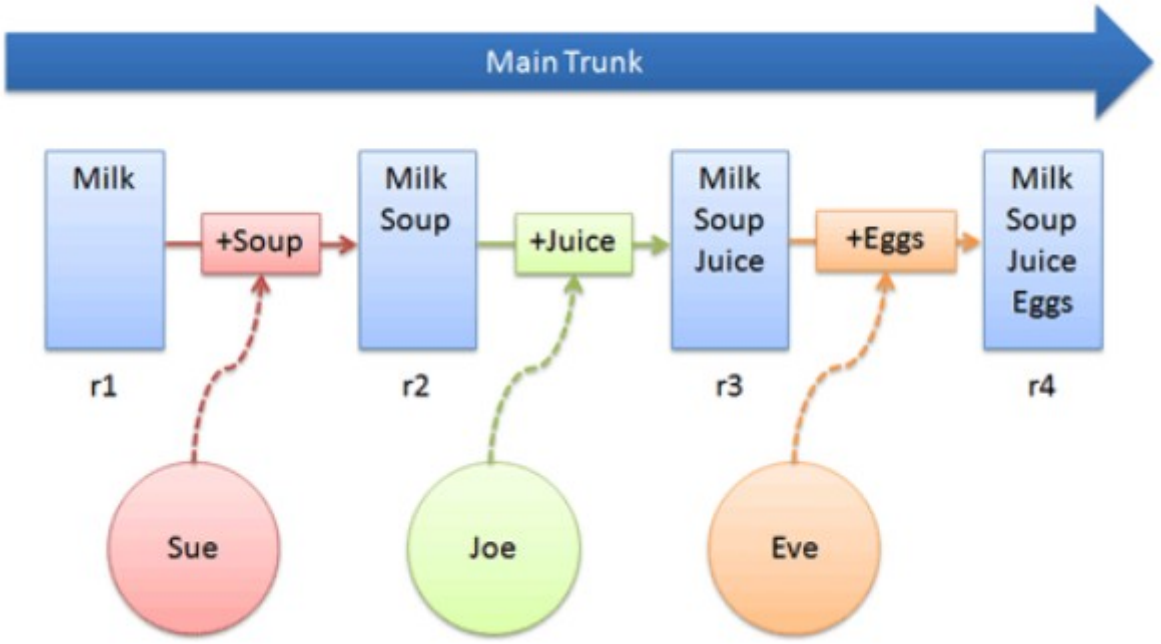
- Branching (ramas)
  - Permite crear una copia de un archivo/carpeta rastreada
  - Permite desarrollar en paralelo en otra “rama” pero dejando constancia de la relación que existe con la rama original.
- Diff/Change/Delta/ (Cambio)
  - Permite encontrar las diferencias entre dos versiones del repositorio.
  - Se puede generar un parche que permitiría pasar de una versión a otra.
- Merge/Patch
  - Aplica los cambios de un archivo a otro
  - Utilizado habitualmente para mezclar branches
- Conflict (Conflicto)
  - Problema que surge cuando varios desarrolladores modifican el mismo recurso y los cambios se solapan.

Tipos de Sistemas de Control de Versiones

Centralizados

- Existe un servidor centralizado que almacena el repositorio completo
- La comunicación/colaboración entre desarrolladores se lleva a cabo (forzosamente) utilizando el repositorio centralizado
- Son más simples de usar
- Los modelos de trabajo son más restringidos

SCV Centralizado

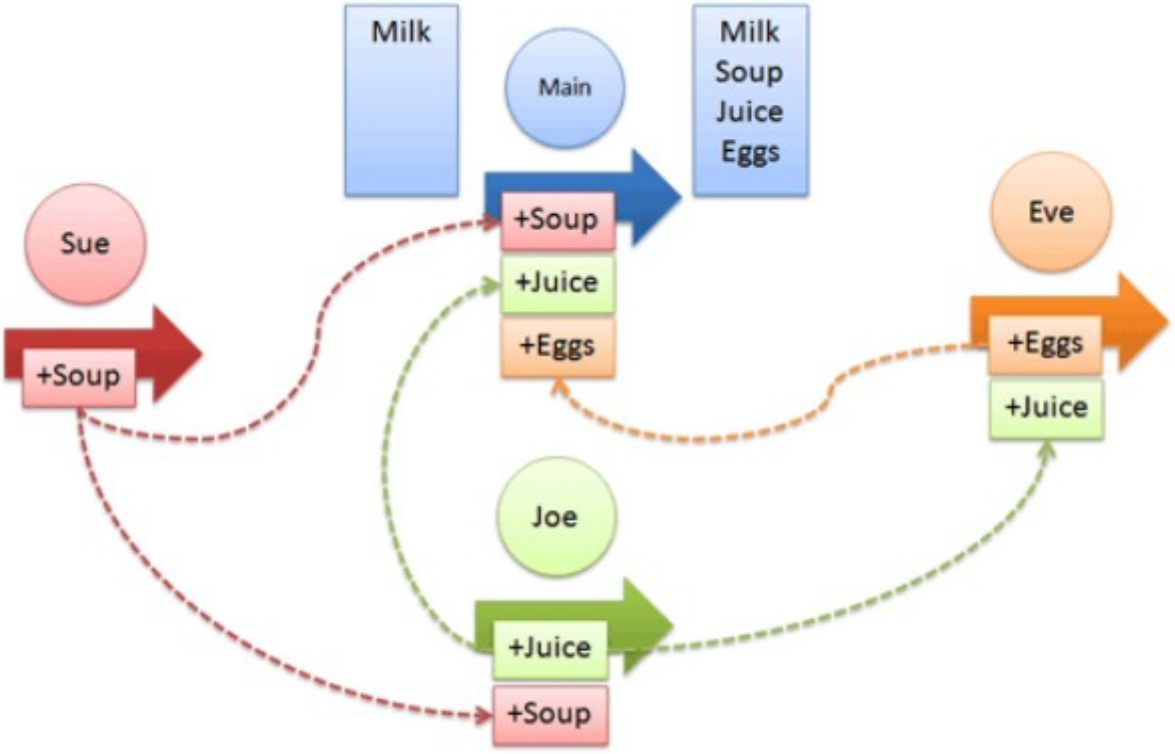


Tipos de Sistemas de Control de Versiones

Distribuidos

- Cada desarrollador contiene una copia completa de todo el repositorio
- Los mecanismos de comunicación/colaboración entre desarrolladores son más abiertos
- Son (un poco) más difíciles de utilizar que los sistemas centralizados
- Los modelos de trabajo son más flexibles
- “Los branches/merges son más simples”

SCV Distribuido



Herramientas SCV

Centralizados

- Subversion (SVN)
  - http://subversion.apache.org, http://subversion.tigris.org/
- Concurrent Version System (CVS)
  - http://www.nonfnu.org/cvs/
- Microsoft Visual Source Safe
- Perforce

Distribuidos

- Git
  - http://git-scm.com
- Mercurial
  - http://hg-scm.com
- Bazaar, DARCS

Políticas de Control de Versiones

- Para aprovechar los SCV es necesario
  - Establecer una política para el control de versiones para los proyectos
    - Estructura del repositorio
    - Política para la rama principal
    - **...**
- Documentar el desarrollo
  - Utilizando alguna herramienta de gestión de seguimiento: Trac
- Es interesante adoptar un modelo de trabajo que sea adecuado para el equipo de desarrollo
  - El modelo de trabajo del equipo de desarrollo puede influir en la elección del SCV a utilizar.

Metodología básica de trabajo

- **La primera vez**

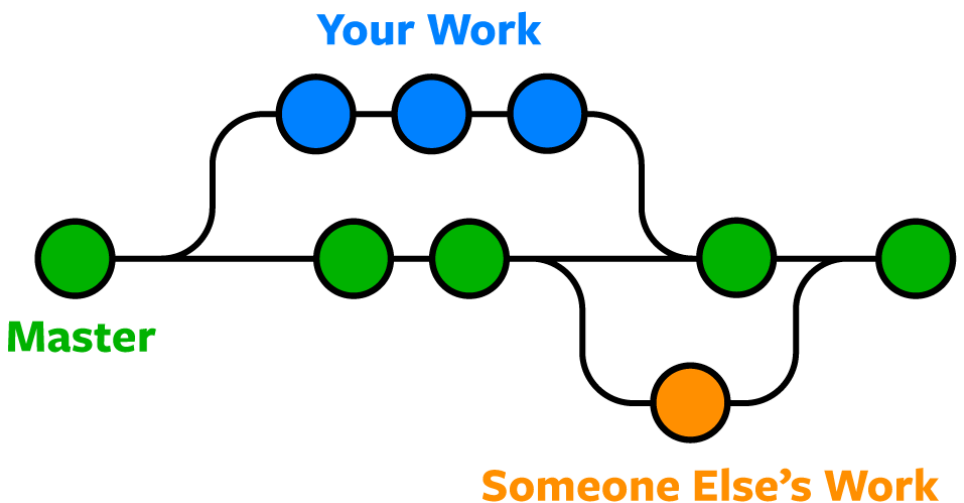
1. Creación del repositorio del proyecto (Opcional)
   - Importación inicial del código del proyecto (Opcional)
2. Crear una copia de trabajo del repositorio
3. Modificar la copia de trabajo
4. Envío de cambios al repositorio

- **Siguientes ocasiones**

1. Actualizar el repositorio.
2. Modificar la copia de trabajo.
3. Envío de cambios al repositorio.

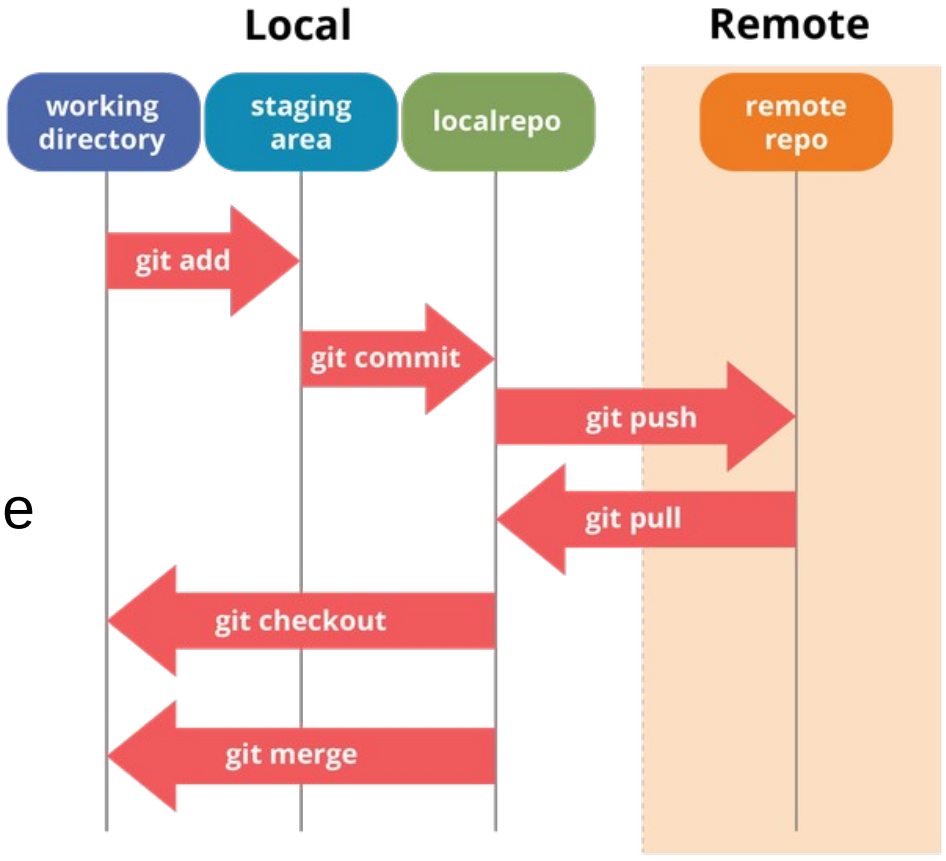
Por qué Git (I)

- **Branch locales "baratos"**
  - Fáciles de crear y borrar
  - No tienen por qué ser públicos
  - Útiles para organizar el trabajo y los experimentos



Por qué Git (II)

- **Todo es local**
  - Operaciones más rápidas
  - Puedes trabajar sin red
  - Todos los repositorios de los desarrolladores son iguales
    - En caso de emergencia puede servir de backup



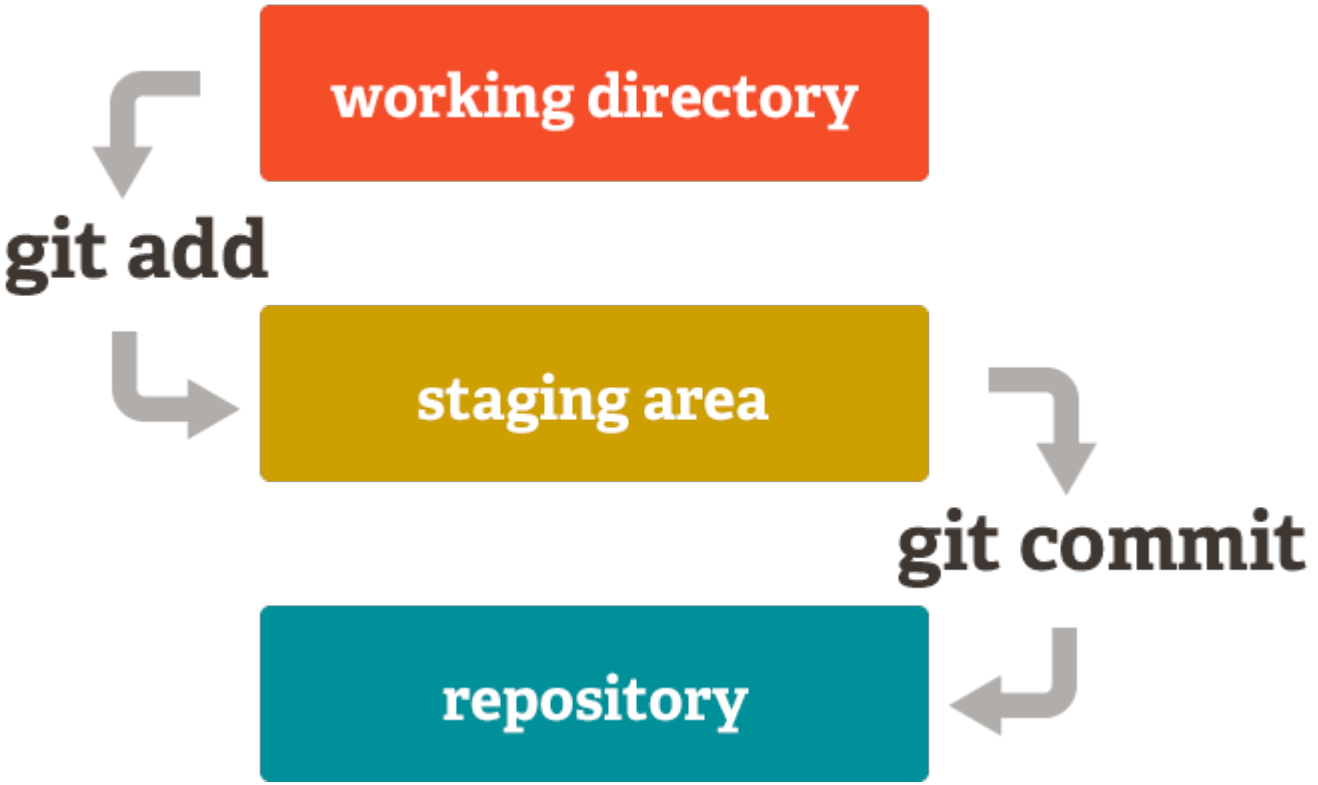
Por qué Git (III)

- **Git es rápido**
  - Comparado con otras herramientas
- **Git es pequeño**
  - Pese a que es una copia de todo el repositorio
  - En algunos casos incluso comparándolo con svn
- Notas
  - Si dos archivos son iguales sólo se guarda el contenido de 1.
  - El contenido de los archivos se guarda comprimido
  - Periódicamente se compactan los archivos
    - Se generan deltas entre las diferentes versiones de los archivos.

Por qué Git (IV)

- La "staging area" (área de ensayo)
  - También denominada índice.
  - Es la zona donde se añaden los cambios que se van a hacer commit.
    - NO es necesario añadir todos los archivos de la WC a la staging area
    - Es incluso posible añadir a la staging area modificaciones concretas dentro de 1 archivo (hunks).
      - Si hay 2 cambios en el archivo se puede hacer commit de 1 de los cambios en un primer paso y otro segundo commit en un segundo
  - Promociona una buena práctica de Git: **haz commit frecuentemente**
    - Que sean pequeños (si es posible)
    - Incluye sólo las modificaciones concretas que resuelvan el problema/tarea.

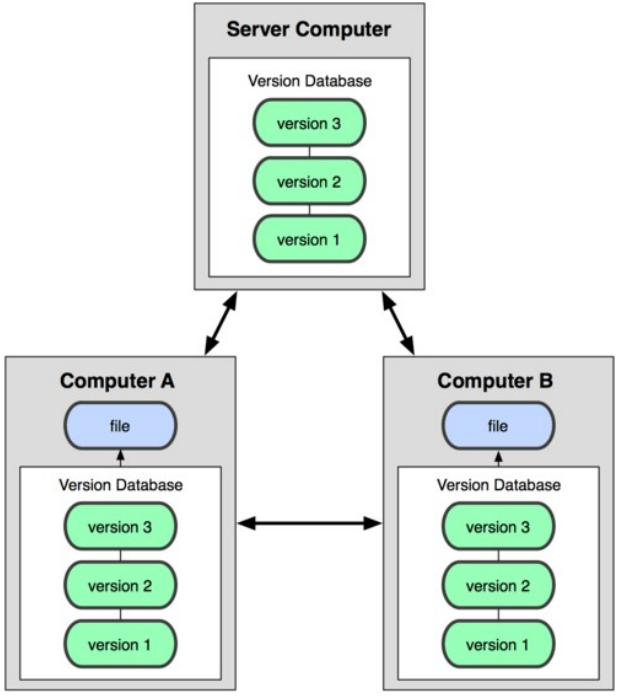
Por qué Git (IV)

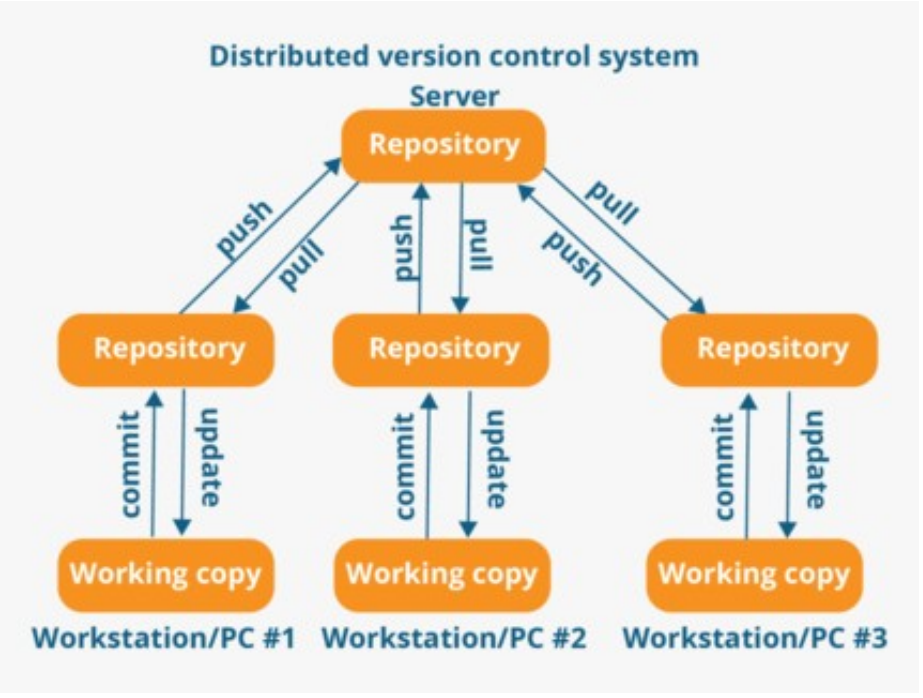


Por qué Git (V)

- Es distribuido
  - Todos los desarrolladores tienen una copia completa del repositorio
    - Pueden ser usadas como backups de emergencia
  - No es (demasiado) lento comparado con SVN
    - Teniendo en cuenta que con SVN sólo trabajamos con una rama a la vez.

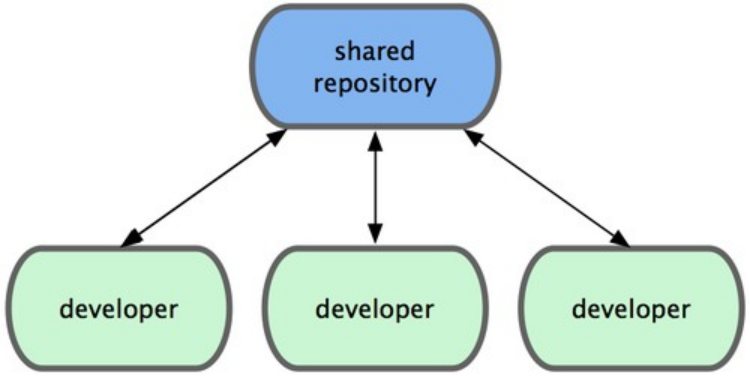
Por qué Git (V)

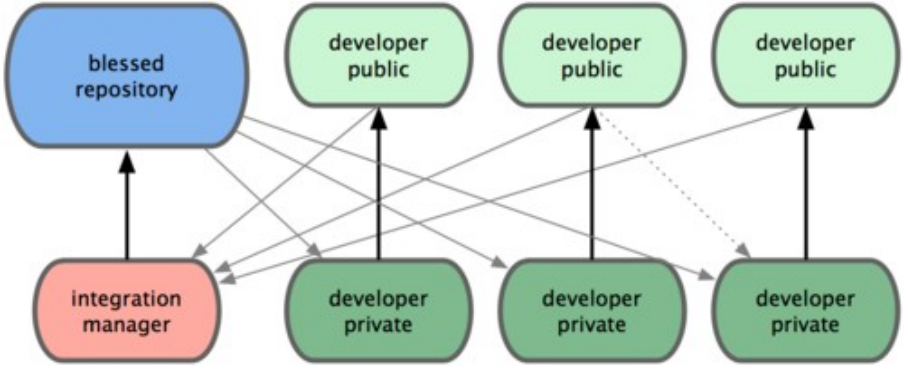

Server Computer
Version Database
version 3
version 2
version 1
Computer A
file
Version Database
version 3
version 2
version 1
Computer B
file
Version Database
version 3
version 2
version 1


Distributed version control system
Server
Repository
push
pull
push
pull
pull
push
Repository
Repository
Repository
commit
update
commit
update
commit
update
Working copy
Working copy
Working copy
Workstation/PC #1
Workstation/PC #2
Workstation/PC #3

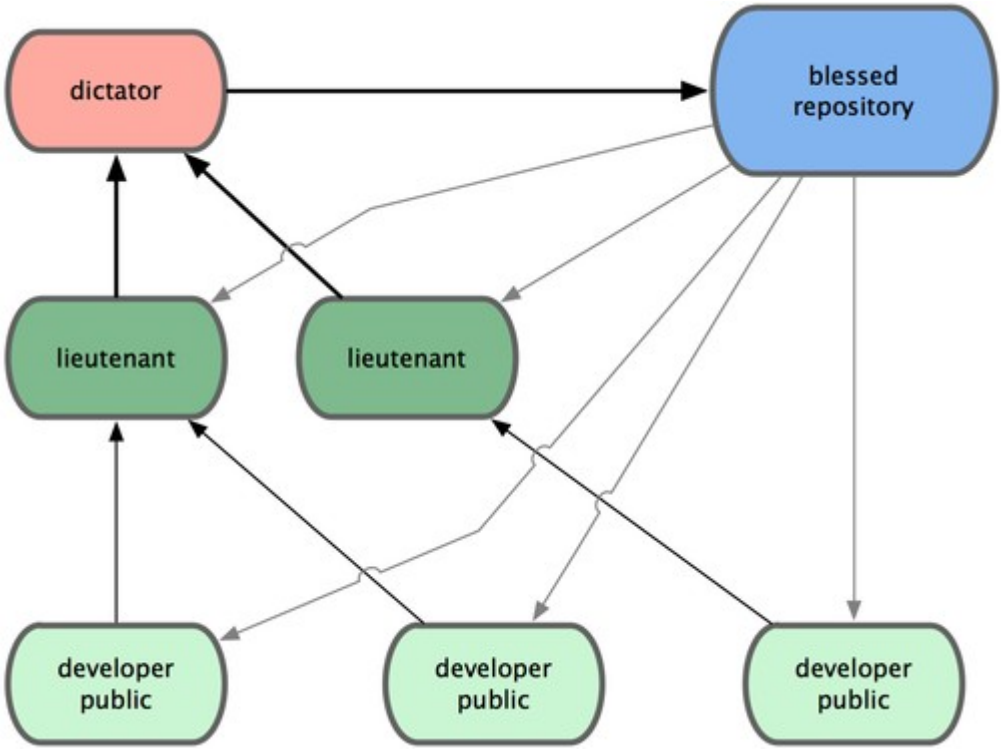
Por qué Git (VI)

- Permite múltiples flujos de trabajo





Por qué Git (VI)



Por qué Git (VII)

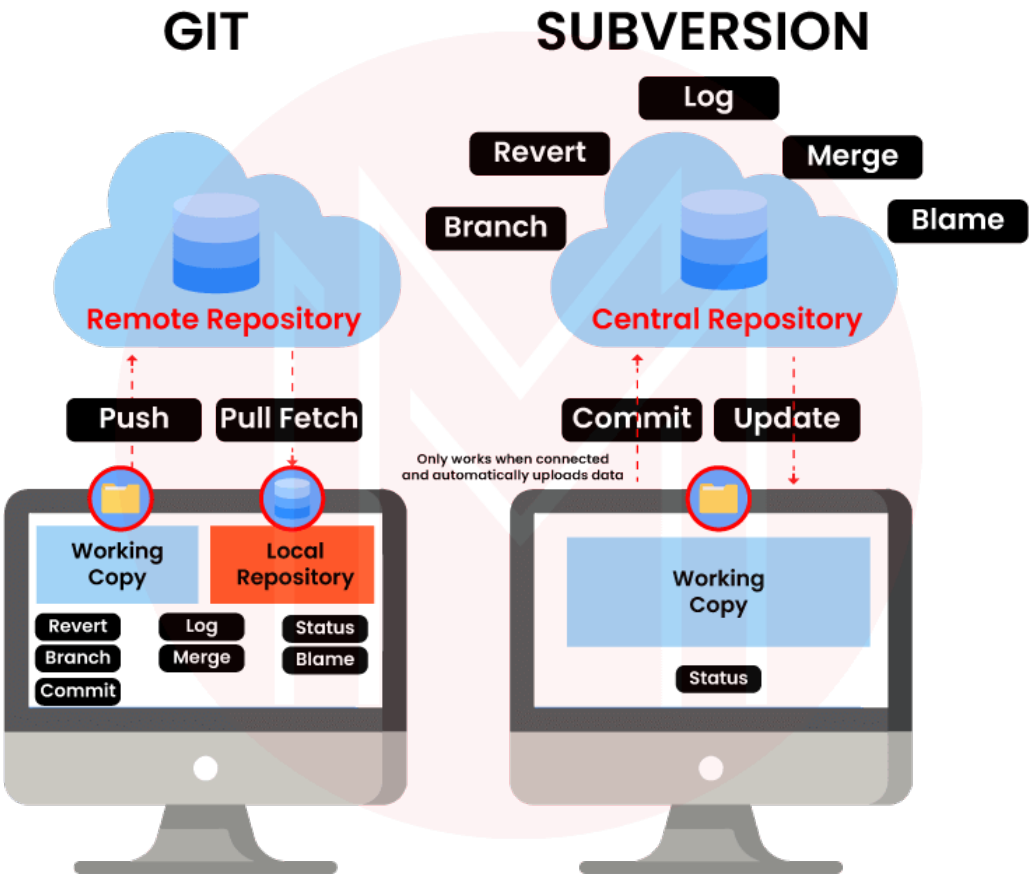
- GitHub y similares
  - Revolución en los proyectos de código libre
    - Mucho más simple colaborar y experimentar
    - Modelo Fork-PullRequest
- Git gestionado
- Además permite alojar la web del proyecto, crear una Wiki, discutir sobre el código o las contribuciones…

Por qué Git (VIII)

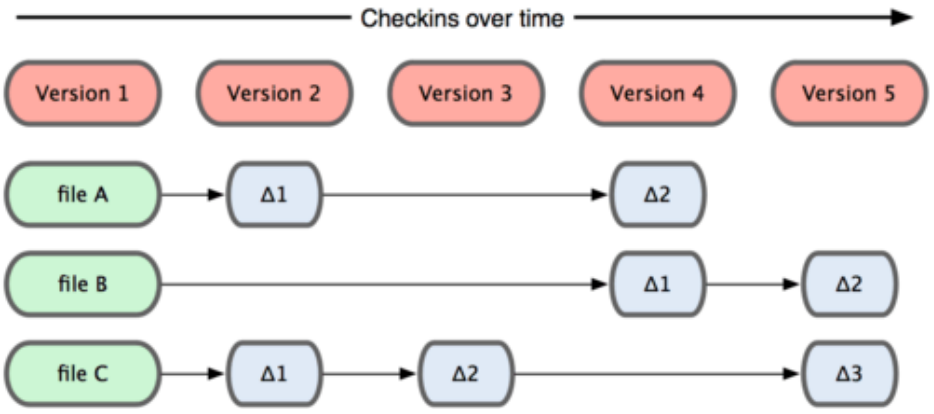
- Git es el nuevo estándar
  - En una gran cantidad de proyectos Open Source: Android, Apache (algunos), Debian, Drupal, ….
  - Cada vez hay más empresas que están migrando su código a Git
  - Hay productos "Enterprise" como JIRA y otros de Atlassian que soportan activamente Git.

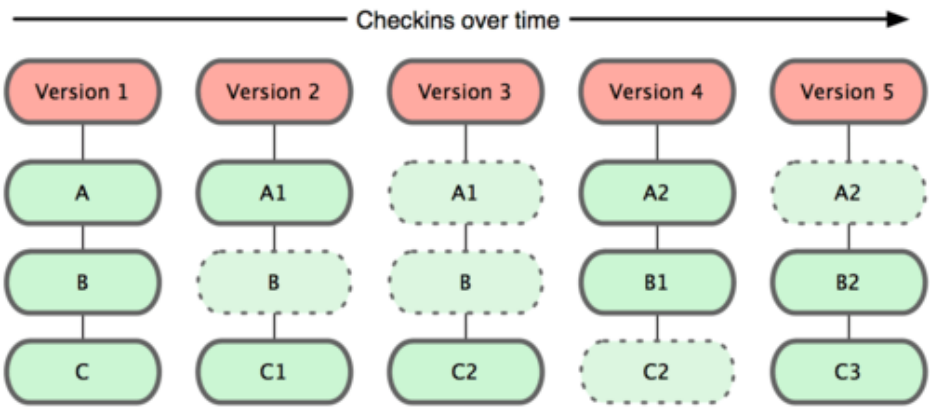
Conceptos básicos (I)

- Git gestiona el repositorio como instantáneas de su estado
  - SVN gestiona el repositorio llevando la cuenta de los cambios incrementales que ha habido.
  - Este hecho simplifica la gestión de branches



Conceptos básicos (II)





Conceptos básicos (II)

Conceptualmente, la mayoría de los otros sistemas almacenan información como una lista de cambios basados en archivos. Estos sistemas (CVS, Subversion, Perforce, Bazaar, etc.) consideran la información que guardan como un conjunto de archivos y los cambios realizados en cada archivo a lo largo del tiempo

Git piensa en sus datos más como un conjunto de instantáneas de un mini sistema de archivos. Cada vez que confirma o guarda el estado de su proyecto en Git, básicamente toma una imagen de cómo se ven todos sus archivos en ese momento y almacena una referencia a esa instantánea. Para ser eficiente, si los archivos no han cambiado, Git no vuelve a almacenar el archivo, solo un enlace al archivo idéntico anterior que ya ha almacenado.

Conceptos básicos

- La mayoría de operaciones son locales
  - En la máquina del desarrollador
  - Incluso para revisar la historia del repositorio
- Git tiene integridad
  - Todo en Git (archivos, carpetas, commits, etc.) tiene una firma asociada
    - SHA1: Bastante seguro respecto a colisiones
    - E.g.: 24b9da6552252987aa493b52f8696cd6d3b00373
- Git normalmente sólo añade datos al repositorio
  - Las operaciones de Git añaden datos dentro del repositorio del proyecto
  - Es posible deshacer fácilmente casi cualquier cambio realizado.

Laboratorio