



UNIVERSIDAD  
SERGIO ARBOLEDA

**TODOS** *A LA* **U**

Fórmate digital

Github en Equipo

Desarrollo de Software

# Contenido

- Forma de trabajo con GIT
- Configuración
- Creando un repositorio GIT
- Gestión del área de ensayo
- Branches en GIT
- Colaboración con Remotes
- Ejercicio

# Forma de trabajo con GIT

Las metodologías no escalan para proyectos de desarrollo. Un Sistema de Control de Versiones permite:

- Crear copias de seguridad y restaurarlas
- Sincronizar (mantener al día) a los desarrolladores respecto a la última versión de desarrollo
- Deshacer cambios
- Gestionar la autoría del código
- Realizar pruebas (aisladas)

# Forma de trabajo con GIT

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

Luego de tener un GIT

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

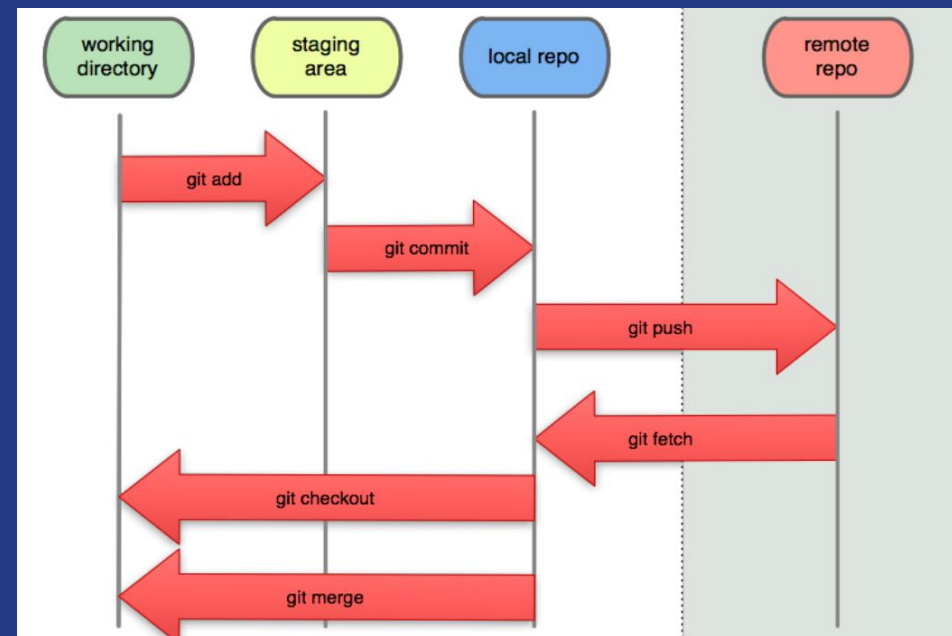
# Branch Locales

- Fáciles de crear y borrar
- No tienen por qué ser públicos
- Útiles para organizar el trabajo y los experimentos



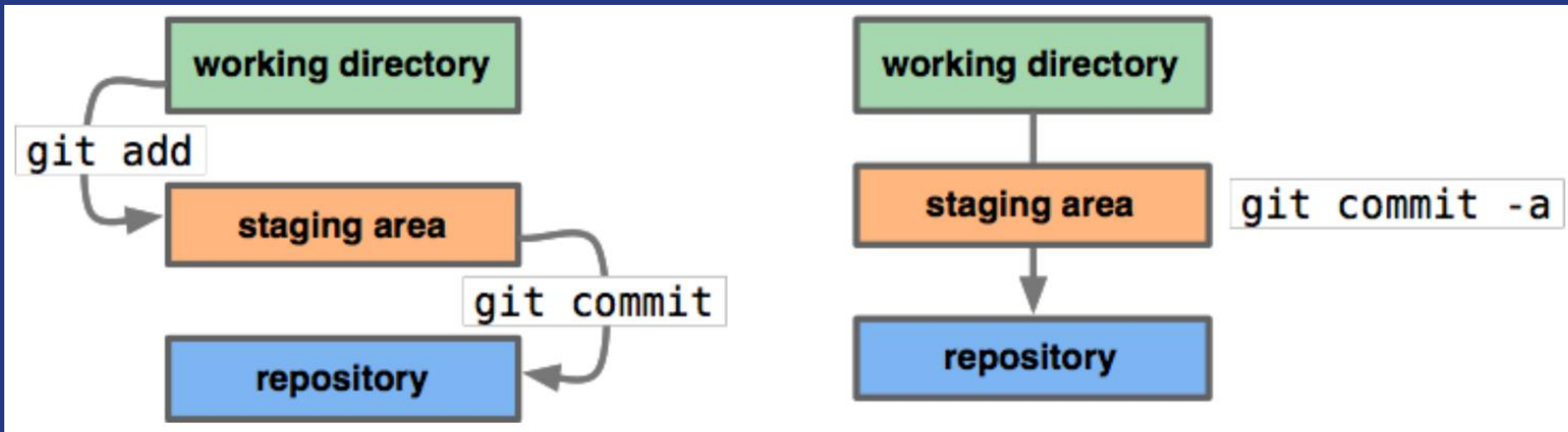
# Características

- 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



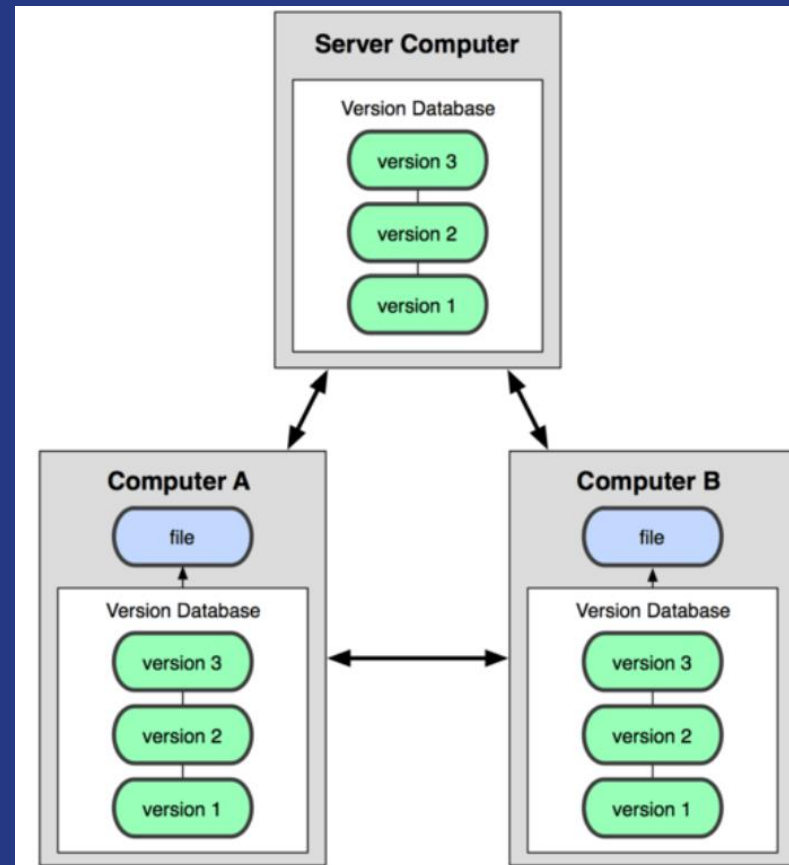
# Área de ensayo

- Es la zona donde se añaden los cambios que se van a hacer commit.
- Promociona una buena práctica de Git: haz commit frecuentemente.



# Distribución

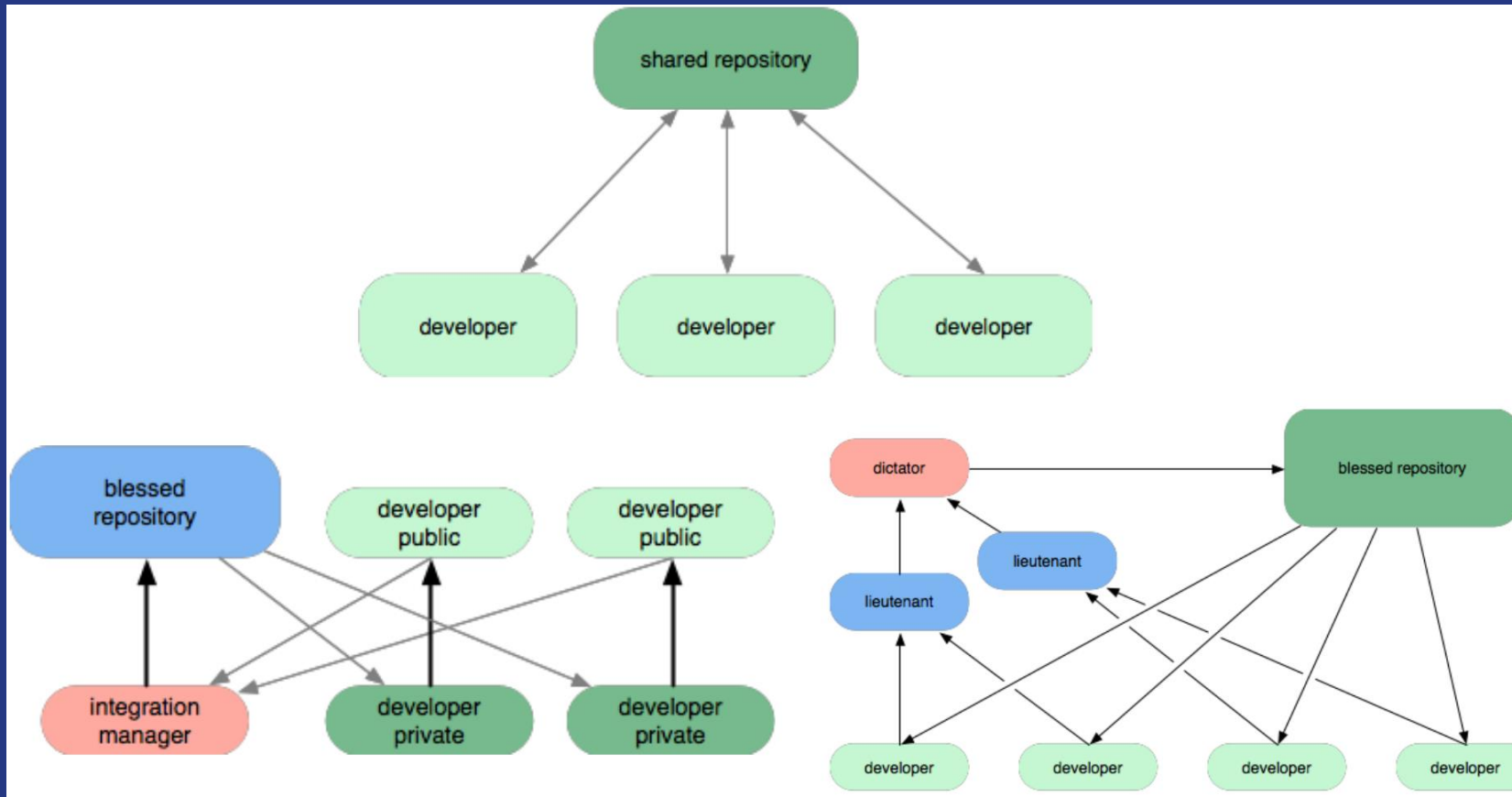
- Todos los desarrolladores tienen una copia completa del repositorio
- No es (demasiado) lento comparado con una aplicación de Subversion.





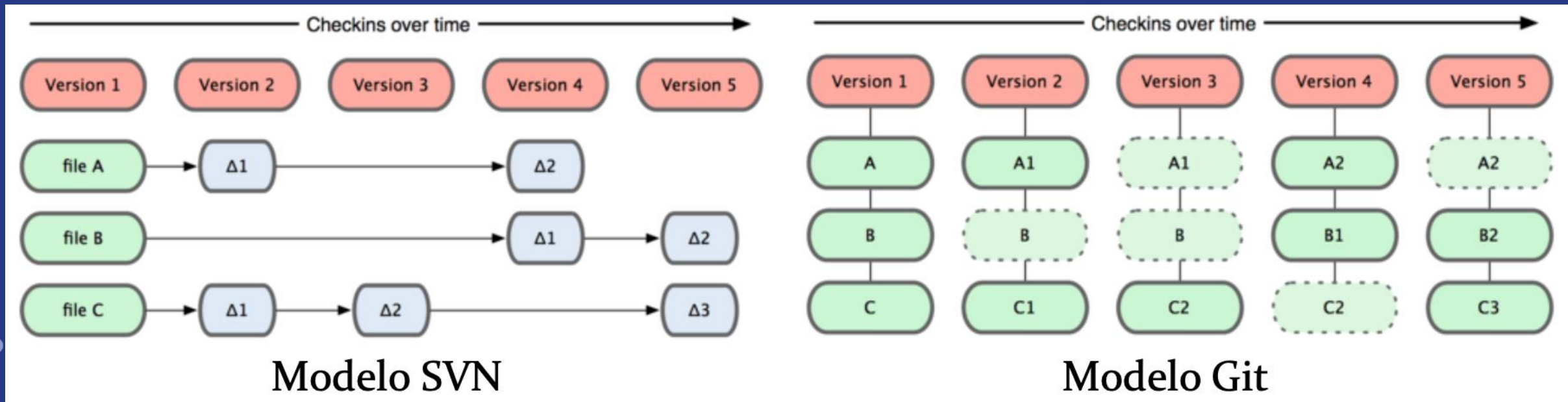
# Distribución

Prmite Multiples flujos de trabajo.



# Conceptos

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



# Configuración

## *Consulta*

- gitconfig--list

## *Modificación*

- Configuración a nivel de proyecto
  - git config <param> <valor>.
  - Edita el archivo <proyecto>/git/config
- Configuración a nivel global (usuario)
  - git config --global ....
  - Crea/Edita el archivo ~/.gitconfig
- Configuración a nivel del sistema
  - git config --system ....

# Configuración

## *Parámetros necesarios (globalmente)*

- user.name,user.email

## *Parámetros interesante*

- core.editor: Controla el editor utilizado en los mensajes de commit
- merge.tool, mergetool.XXXX.path: Controla la herramienta externa utilizada para resolver conflictos.

# Creando un repositorio GIT

## *Creación de un repositorio a partir de código ya existente*

- `cd <ruta proyecto>; git init`

## *Creación de proyecto en blanco*

- `git init <ruta proyecto>`

## *Creación de un repositorio a compartir*

- `git init --bare <ruta proyecto>`
- Se puede crear en la máquina de desarrollo y mover más adelante a un servidor compartido.

# Creando un repositorio GIT

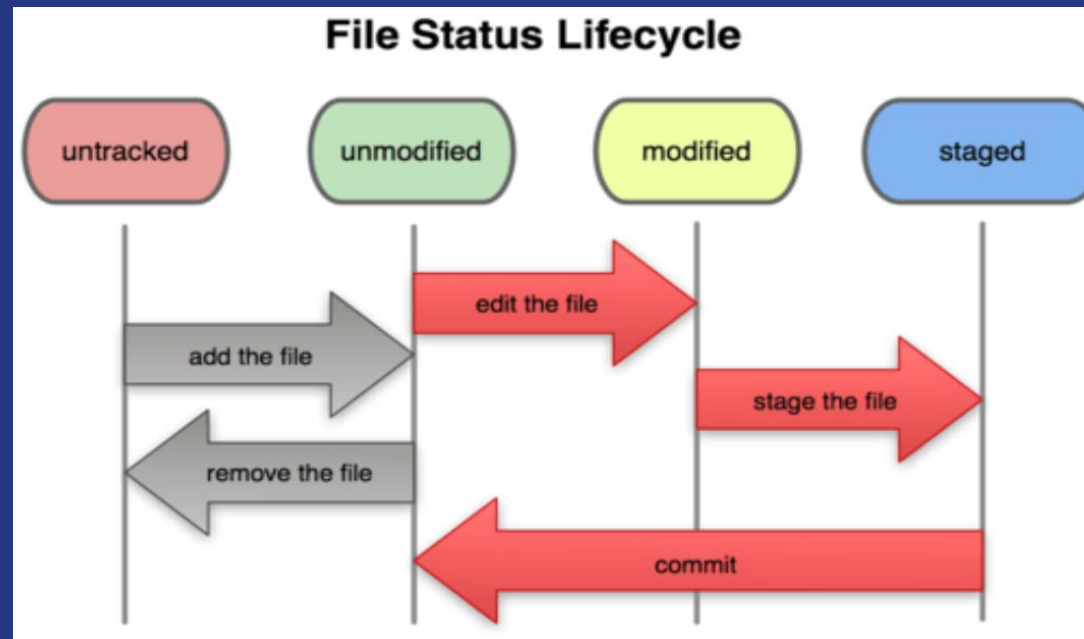
*Crea la carpeta .git en la raíz del proyecto*

- NO en el caso de --base

*Dentro se alojan todos archivos y carpetas internos que gestionan un repositorio de git*

# Estado de los archivos

- Committed: Gestionado por GIT
- Modificado: Gestionado por GIT pero modificado en la WC (Working Copy)
- Staged: Marcado como modificado para incluirlo en el siguiente commit.
- Untracked: fuera de la gestión de Git



# Gestión del área de ensayo

## *Verificar el estado de la staging area*

- gitstatus: Cambios pendientes de commit
- gitdiff: Muestra los cambios de archivos modificados pero NO añadidos al staging area
- gitdiff--cached: Muestra los cambios de archivos modificados que SI están añadidos al staging area



# Gestión del área de ensayo

## *Añadir archivos a la staging area*

- `git add <ruta archivo>`
- `git add .` # Añade todos los archivos nuevos o modificados
- NOTA: si modificas el archivo añadido tendrás que volver a añadirlo
- `git add -A` # Añade todos los archivos modificados, nuevos o borrados
- La opción `-n` muestra los cambios a realizar en la staging area pero no los realiza

# Gestión del área de ensayo

## *Eliminando archivos*

- Opción 1 (recomendada): `git rm <archivo>`
- Opción 2: `rm <archivo>; git rm [-f ] <archivo>`
- Elimina el archivo tanto de la WC y anota en la staging area la eliminación.

## *Crear una instantánea del repositorio*

- `git commit [-m "Mensaje"]`
- Crea una instantánea en el repositorio teniendo en cuenta:
  - El estado de la última instantánea realizada
  - El contenido de la staging area

# Gestión del área de ensayo

## *Renombrando*

- git mv <origen> <destino>
- Es un resumen de: mv <origen> <destino>; git rm <origen>; git add <destino>

*Git seda cuenta de que estamos renombrando el archivo debido a la firma del archivo.*

# Gestión del área de ensayo

## *Visualizar la historia de los commits*

- `git log [-p ] [-2]`
  - `p`: Visualiza los cambios realizados(diff) en los commit
  - `2,ó-N`: límite del número de commits a visualizar
- Cuando se complica la estructura del repositorio mejor utilizar `gitk` o la interfaz gráfica
- `git log --pretty=format:"%h %s" --graph`: proporciona representación textual si no se tiene a mano una interfaz gráfica

# Gestión del área de ensayo

## *Buscar el culpable*

- `git blame <file>`
  - Muestra el autor que ha modificado por última vez cada línea de un archivo.

# Errores de actualización

## *Error en el último commit*

- `gitcommit--amend`

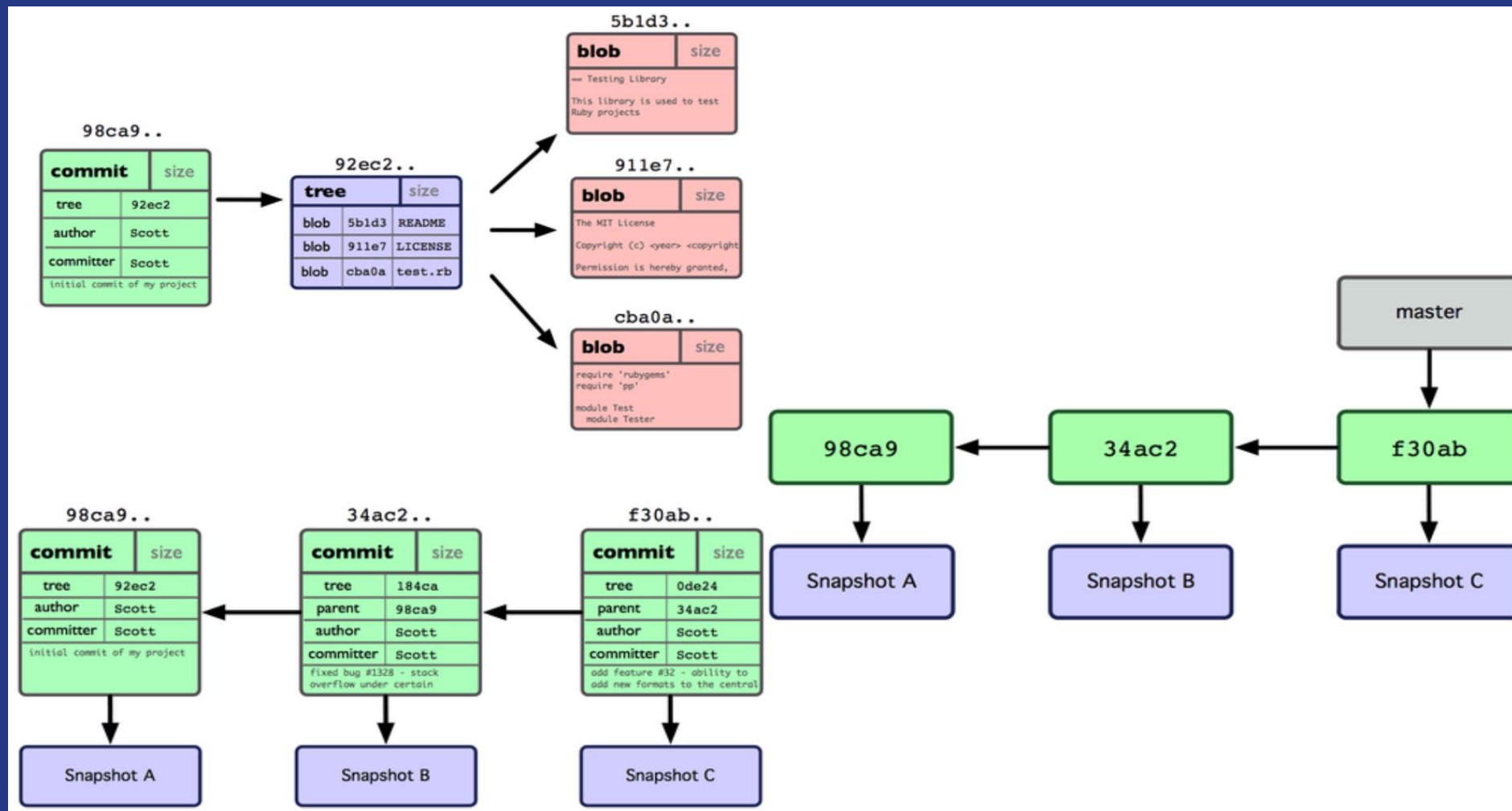
## *Eliminar un archivo del staging area sin perder las modificaciones*

- Si el archivo es nuevo: `git rm --cached <archivo>`
- Si el archivo está modificado: `git reset HEAD <archivo>`

## *Deshacer los cambios en la copia de trabajo y volver al archivo original desde la última instantánea*

- `gitcheckout--<archivo>`

# Estructura interna de un repositorio GIT



# Branches (ramas) en GIT

*Por defecto existe el branch master*

*Listar branches*

- git branch [-v] -a
- La referencia HEAD apunta al branch actual

*Crear un branch (local)*

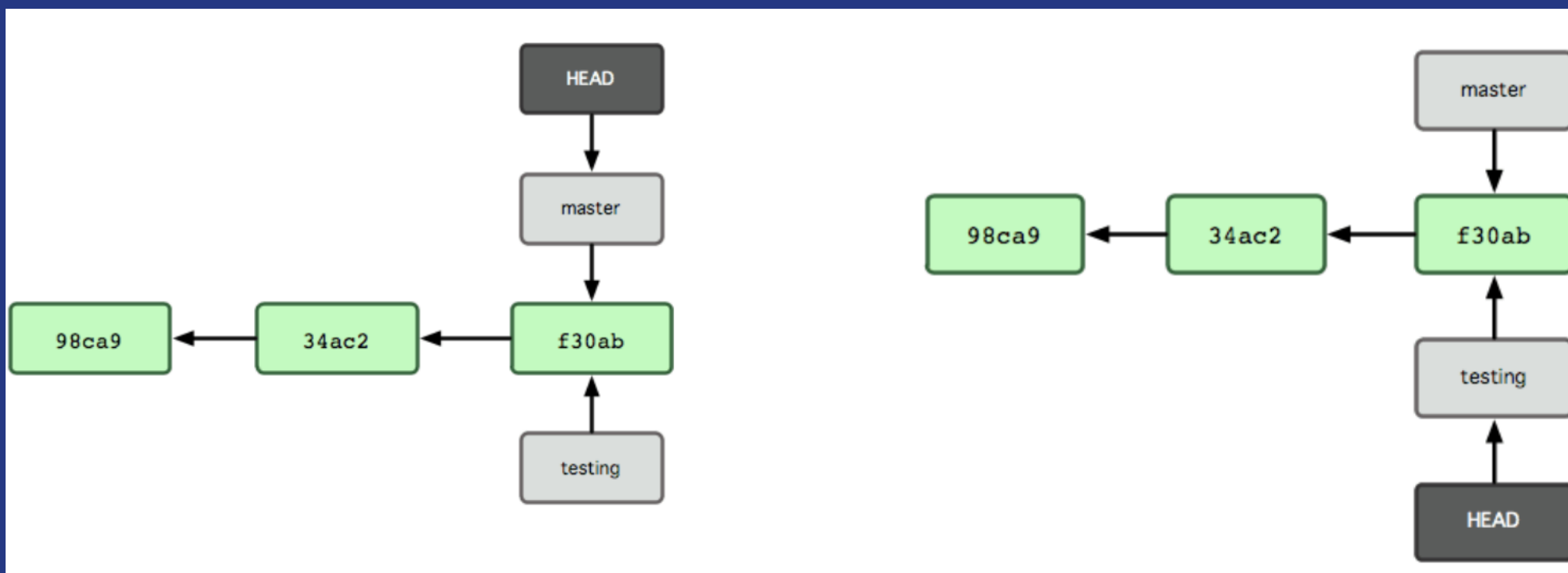
- git branch <nombre branch>
- Crea un branch a partir del branch actual

*Pasar a trabajar a otro branch*

– git checkout <nombre branch>

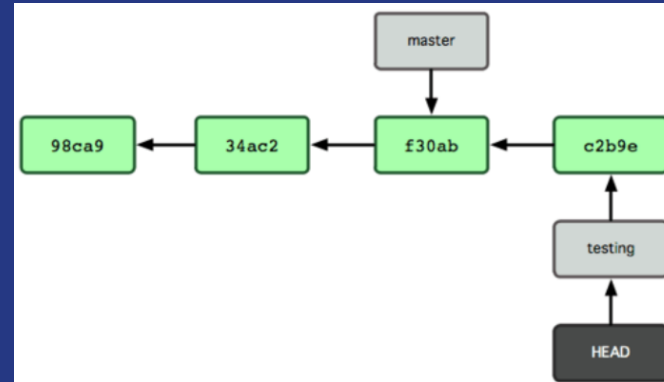


# Branches (ramas) en GIT

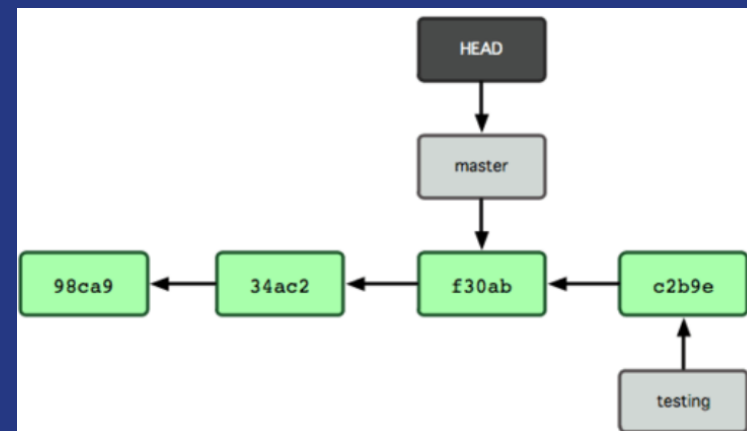


# Branches (ramas) en GIT

*Al hacer commit se realizarán sobre el branch activo*



*Podemos volver al branch master cuando queramos*



# Branches (ramas) en GIT

## *Flujo de trabajo con branches*

- Crear un branch cuando tengo que hacer una tarea o quiero experimentar algo.
- Trabajar sobre el branch (desarrollar, hacer pruebas)
- Nos aseguramos que la copia de trabajo está limpia

*Actualizamos nuestro branch de trabajo con los cambios que haya habido en master*

*Cuando estamos contentos con el trabajo hacemos un merge del trabajo en el branch master*

# Branches (ramas) en GIT

## *¿Cómo hacemos el merge?*

- Checkout del branch donde vamos a integrar los cambios: git checkout master
- Integramos los cambios: git merge tarea

## *Cuando se realizan los merges es posible que haya que resolver conflictos*

- Conflictos: modificaciones sobre un mismo archivo que git no sabe resolver.

# Gestión de Branches en GIT

*¿Cómo averiguo los branches que hay?*

- git branch [-a -v]
- El branch activo aparece con un '\*'

*¿Cómo averiguo que branches NO están integrados con el branch activo?*

- git branch --no-merged

*¿Cómo averiguo que branches SI están integrados con el branch activo?*

- git branch --merged

*Una vez que un branch está integrado puedo eliminarlo (si quiero)*

- git branch -d <branch>

# Colaboración con git: remotes

- La colaboración entre desarrolladores se realiza a través de repositorios remotos
- Desde tu repositorio es posible acceder a otros repositorio para traerte cambios
- No es obligatorio un servidor git

## *¿Cómo me traigo mi repositorio publico a mi máquina?*

- `git clone <url>`
- Automáticamente crea un remote llamado "origin"

# Colaboración con git: remotes

## *Puedo visualizar los remotes que hay en mi repositorio*

- `git remote #` Muestra el nombre del remote
- `git remote -v #` Muestra la URL que se utilizó para crear el remote

## *Puedo añadir más remote*

- De hecho tenemos que añadir el repositorio maestro
- `git remote add <nombre> <URL>`
- `git remote add upstream`  
`ssh://git@git.miempresa.com/var/lib/git/maestro.git`

# Colaboración con git: remotes

- Los repositorios remotos también tienen alojados los branches
- Son referencias a branches en un repositorio remote
- Cuando se clona un repositorio remoto se crea una branch local asociado al branch del master

## *¿Cómo traerme un branch remoto?*

- `git checkout --track <remote>/<branch>`
- `git checkout -b <branch> <remote>/<branch>`



# Colaboración con git: remotes

*¿Cómo me traigo cambios de algún repositorio remoto?*

- git fetch <nombre remote>

*¿Cómo listo los branches que hay en un remote?*

- git ls-remote <remote>

*¿Cómo aplico los cambios que hay en un remote?*

- git merge <remote>/<nombre branch>

*¿Cómo me traigo los cambios y los aplico?*

- git pull [<remote>] [<nombre branch>]

# Colaboración con git: remotes

## *¿Cómo envío cambios a un remote?*

- `git push [<remote>] [<nombre branch>]`

## *¿Cómo "romper" un remote?*

- `git push --force [<remote>] [<branch>]`

## *¿Cómo borro un branch remoto?*

- `git push origin :<branch>`

# Ejercicios

## Crear un repositorio GIT

- Con su equipo de trabajo, configurar un repositorio GIT para trabajar en el desarrollo de su proyecto