

Introducción



Introducción

- Git y Github son herramientas esenciales en el desarrollo de software y hardware, ya que permiten el control de versiones y el trabajo colaborativo de forma efectiva.
- El uso de Git y Github permite el seguimiento de los cambios realizados en el código fuente, lo que facilita la identificación de errores y la corrección de los mismos.
- La utilización de Git y Github permite a los equipos de desarrollo trabajar en paralelo en diferentes ramas de código sin afectar el trabajo del resto del equipo.

Introducción

- Git y Github también facilitan la integración continua, lo que permite detectar errores de manera temprana y asegurar la calidad del código.
- El uso de Git y Github como metodología de trabajo en equipo es esencial para el desarrollo de productos de hardware y software para el Internet de las Cosas, ya que permite una mayor eficiencia y calidad del trabajo.



git



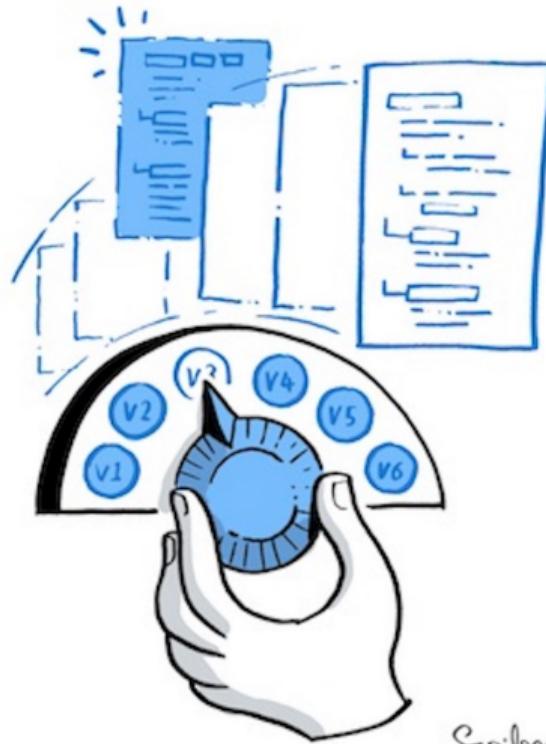
GitHub

¿Qué es el control de versiones?

VERSION CONTROL



TRACK PROJECT HISTORY



Control de versiones

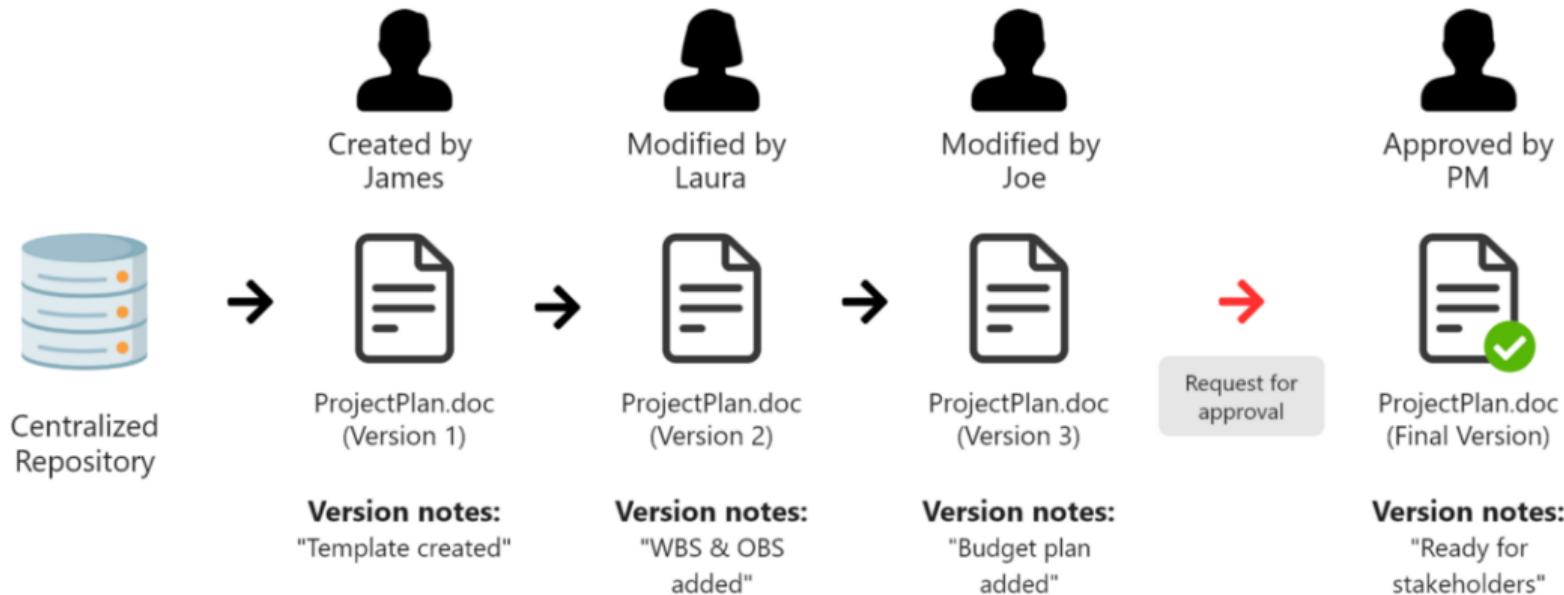
Control de versiones

El control de versiones es un sistema que permite registrar y gestionar los cambios realizados en el código fuente de un software o proyecto, facilitando el seguimiento de las versiones y la colaboración en equipo.

Características principales del control de versiones

- Permite realizar un seguimiento de los cambios realizados en el código fuente.
- Facilita la colaboración en equipo y el trabajo en paralelo.
- Deshacer cambios: Permite volver a versiones anteriores del código fuente en caso de errores o problemas.
- Crear copias de seguridad y restaurarlas
- Sincronizar (mantener al día) a los desarrolladores respecto a la última versión de desarrollo
- Gestionar la autoría del código
- Realizar pruebas (aisladas)

Document Version Control Flow



Ventajas del control de versiones

- Facilita la colaboración en equipo y el trabajo en paralelo.
- Ayuda a detectar errores de manera temprana.
- Permite mantener un registro histórico de los cambios realizados en el código fuente.
- Facilita la integración continua y la automatización de procesos.

Local Computer

Checkout

File

Version Database

Version 3

Version 2

Version 1

Checkout

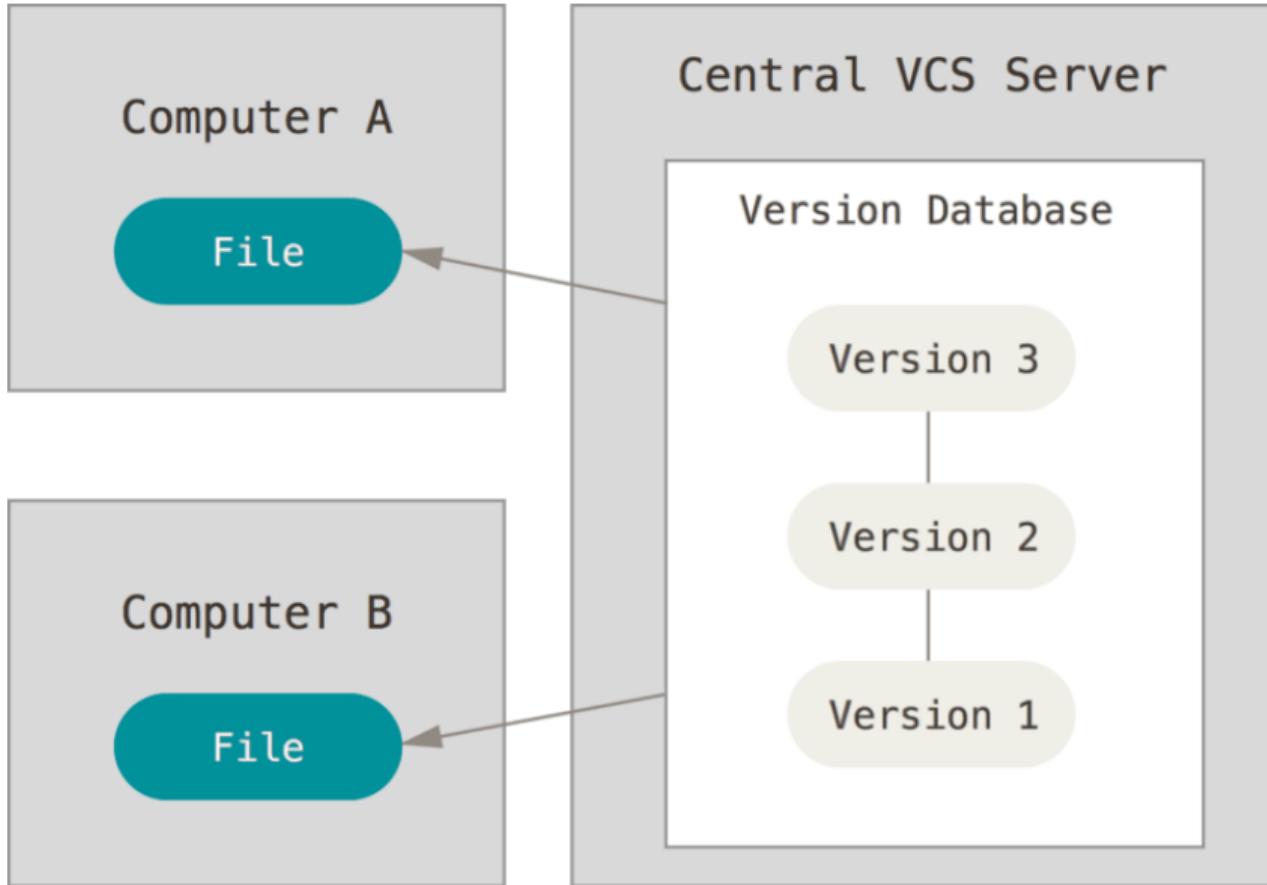
File

Version Database

Version 3

Version 2

Version 1



Casos de uso del control de versiones

- **Casos en los que se puede utilizar:** en proyectos de desarrollo de software y hardware, en proyectos de diseño gráfico y multimedia, en proyectos de documentación y contenido web.
- **Casos en los que no se recomienda utilizar:** en proyectos pequeños con un solo desarrollador, en proyectos con cambios poco frecuentes, en proyectos con pocos recursos disponibles, en proyectos con un plazo muy corto.

¿Qué es Git?

¿Qué es Git?

Git

Git es un software de control de versiones diseñado por **Linus Torvalds**, pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos de código fuente.

- Página oficial de Git: git-scm.com/



Ventajas de utilizar Git como sistema de control de versiones

- Mayor velocidad y eficiencia en la gestión de grandes repositorios de código.
- Mayor seguridad y protección de los datos del proyecto.
- Facilidad para la creación y gestión de ramas para trabajar en diferentes funcionalidades.

Ventajas de utilizar Git como sistema de control de versiones

- Mayor flexibilidad para colaborar en el proyecto en diferentes entornos y plataformas.
- Permite mantener un historial de los cambios y las actualizaciones del proyecto más detallado.
- Soporte de la comunidad y una gran cantidad de recursos y herramientas adicionales disponibles para complementar el trabajo con Git.

Git WorkFlow

Git WorkFlow

Git Workflow es un proceso que describe cómo se utiliza Git para gestionar y colaborar en el desarrollo de un proyecto, desde la creación de una rama para trabajar en una nueva funcionalidad hasta la integración de los cambios en la rama principal.

Glosario Git

Glosario Git

- **Repository (Repositorio):** Es donde se almacena todo el código y los archivos relacionados con un proyecto.
- **Commit (Confirmar):** Es el proceso de guardar los cambios en el repositorio.
- **Branch (Rama):** Es una copia del código que se puede modificar y fusionar con la rama principal.
- **Merge (Fusión):** Es el proceso de unir dos ramas de código en un solo conjunto de cambios.
- **Conflict (Conflicto):** Es el problema que surge cuando dos ramas contienen cambios contradictorios que no pueden fusionarse sin intervención manual.
- **Tag (Etiqueta):** Es una forma de identificar un punto específico en la historia del repositorio para hacer referencia a él más tarde.
- **Head (Cabeza):** Es una referencia al último commit en la rama actual.

Git WorkFlow

- 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

Git WorkFlow

- Crear una nueva rama de trabajo a partir de la rama principal (`git checkout -b [nombre-rama]`).
- Hacer cambios en los archivos del proyecto y guardarlos localmente en la rama de trabajo (`git add [nombre-archivo]` y `git commit -m "[mensaje-de-confirmación]"`).
- Subir los cambios de la rama de trabajo al repositorio (`git push origin [nombre-rama]`).

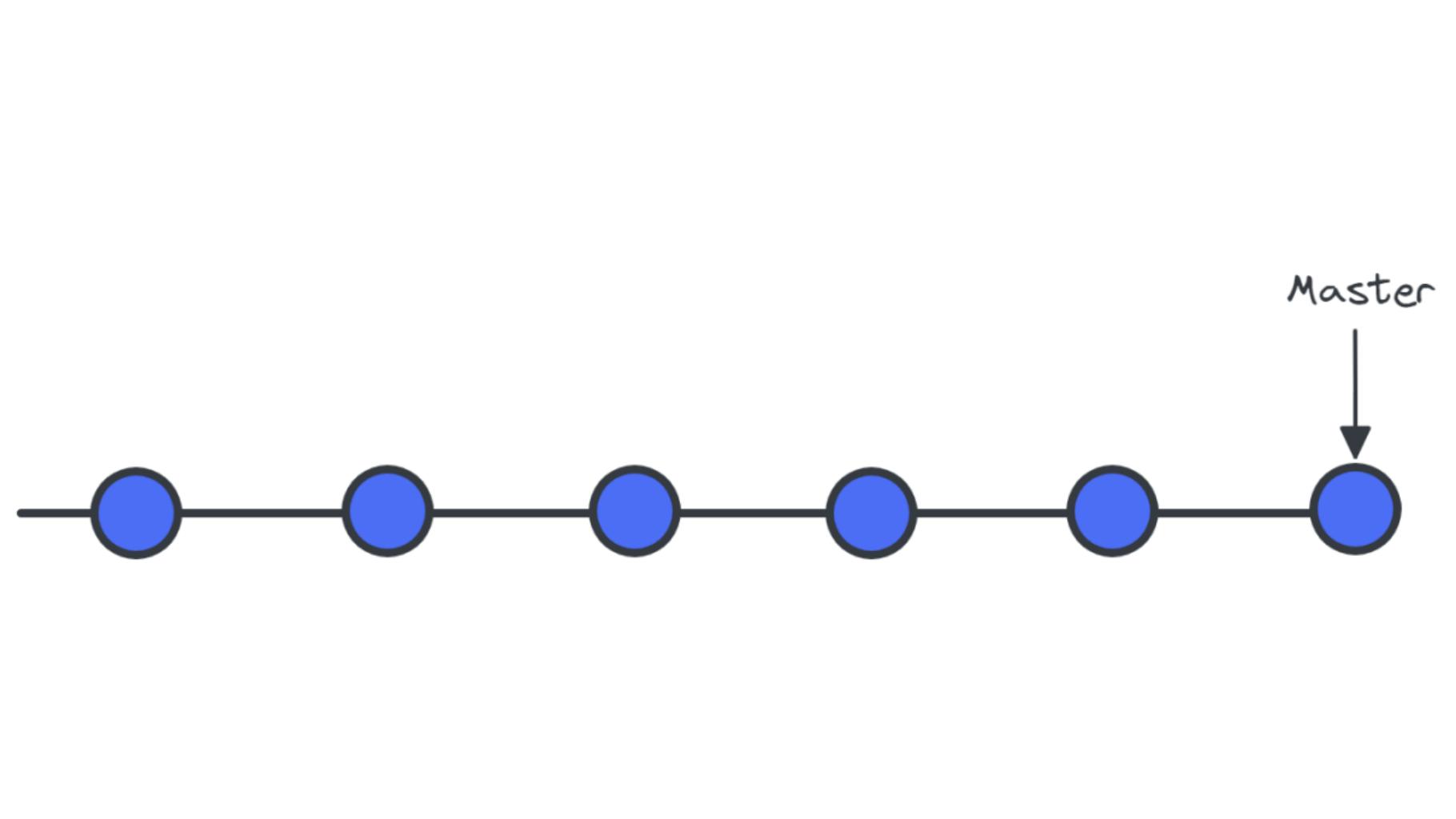
Branch Locales

Branch Locales

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

Branch Locales

- 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*



Master

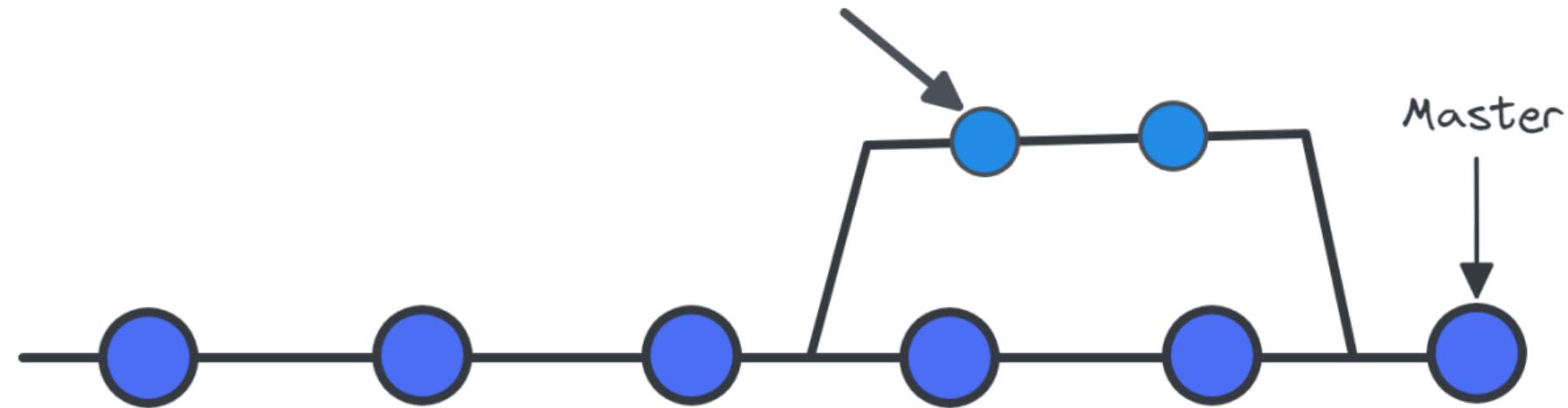
The diagram illustrates a linear sequence of six nodes, each represented by a blue circle with a black outline. The nodes are connected by a single horizontal black line. An arrow points downwards from the word "Master" to the rightmost node, indicating it is the central node of the sequence.

Git WorkFlow

- Hacer cambios adicionales en la rama de trabajo y subirlos al repositorio (`git add [nombre-archivo]` y `git commit -m "[mensaje-de-confirmación]"` y `git push origin [nombre-rama]`).
- Actualizar la rama de trabajo con los cambios realizados en la rama principal (`git merge [rama-principal]` y `git push origin [nombre-rama]`).

New Feature Branch

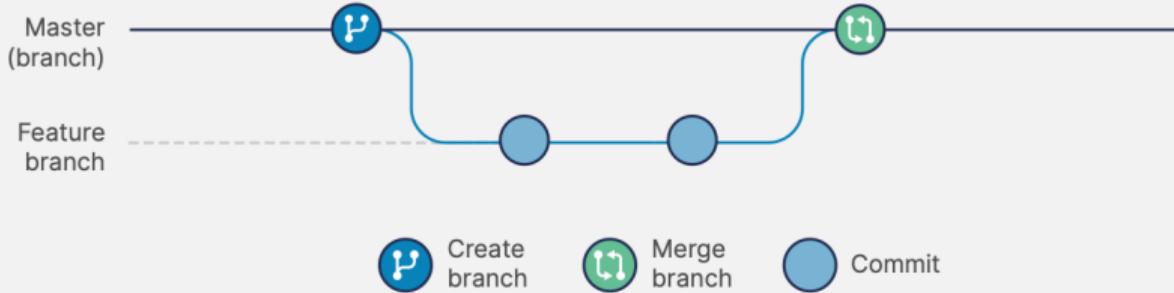
Master



Git WorkFlow

- Resolver los conflictos que surjan al fusionar la rama principal con la rama de trabajo.
- Aprobar y fusionar los cambios de la solicitud de extracción en la rama principal.
- Eliminar la rama de trabajo después de que los cambios se hayan fusionado con éxito en la rama principal (`git branch -d [nombre-rama]`).

Repository

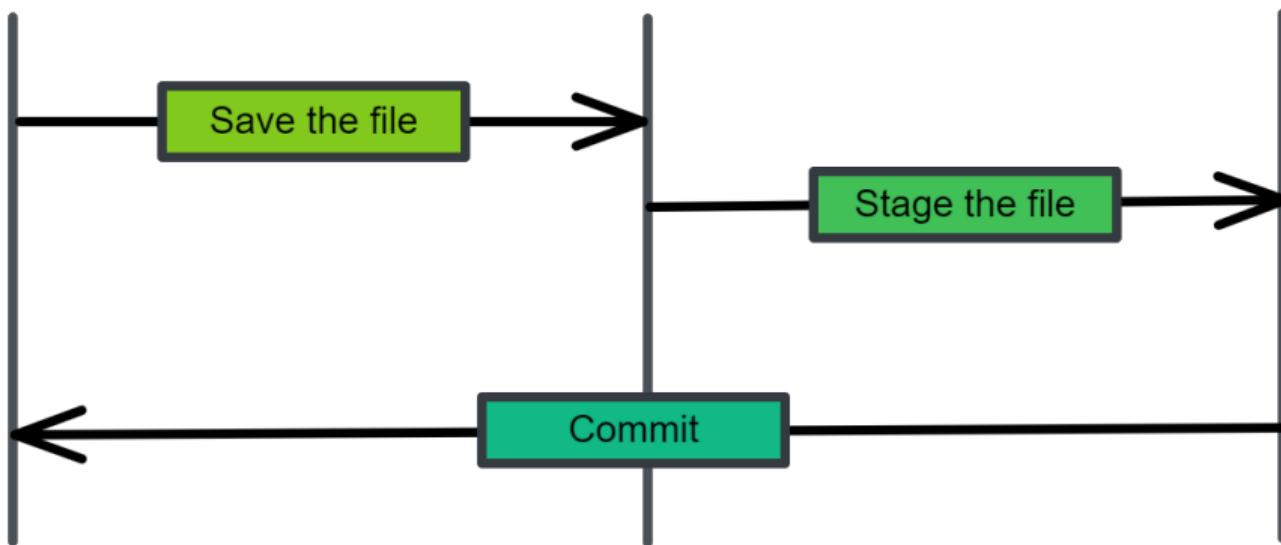


Estado de los archivos

Unmodified

Modified

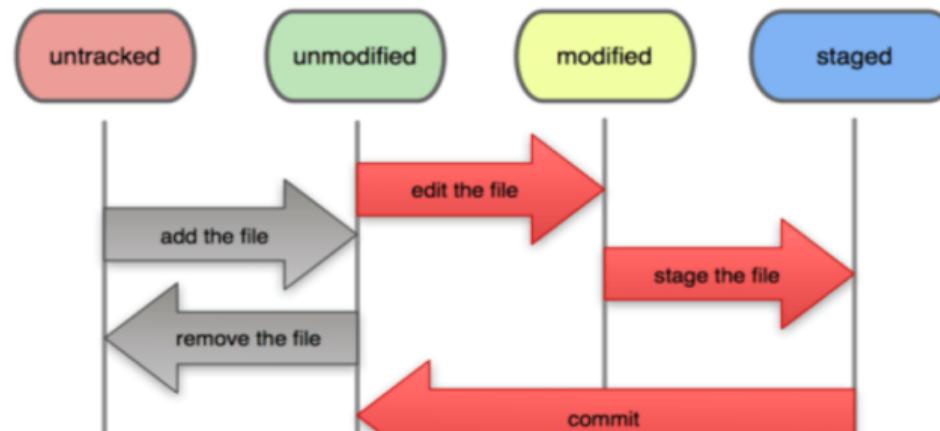
Staged

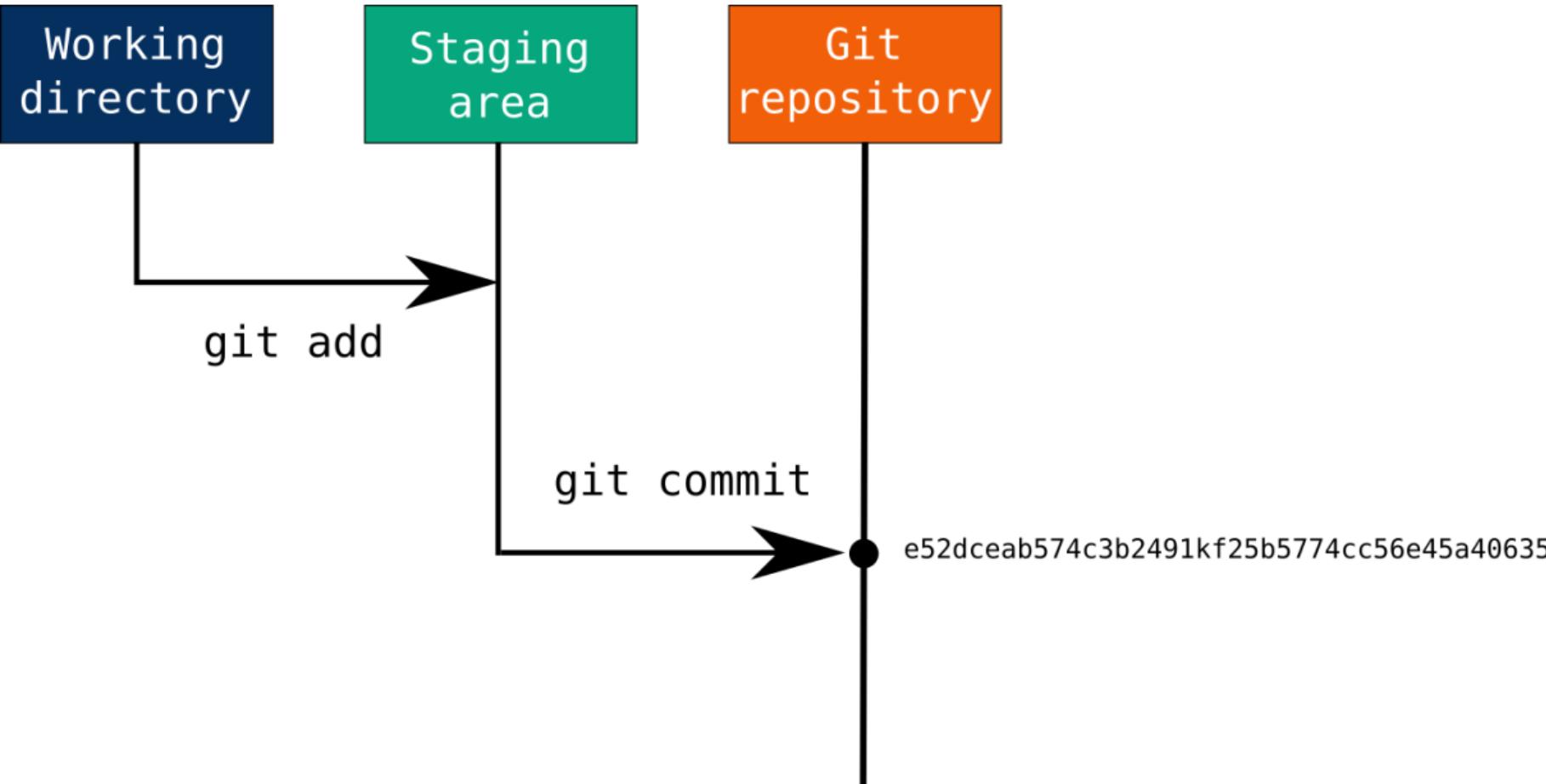


Estado de los archivos

- **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

File Status Lifecycle

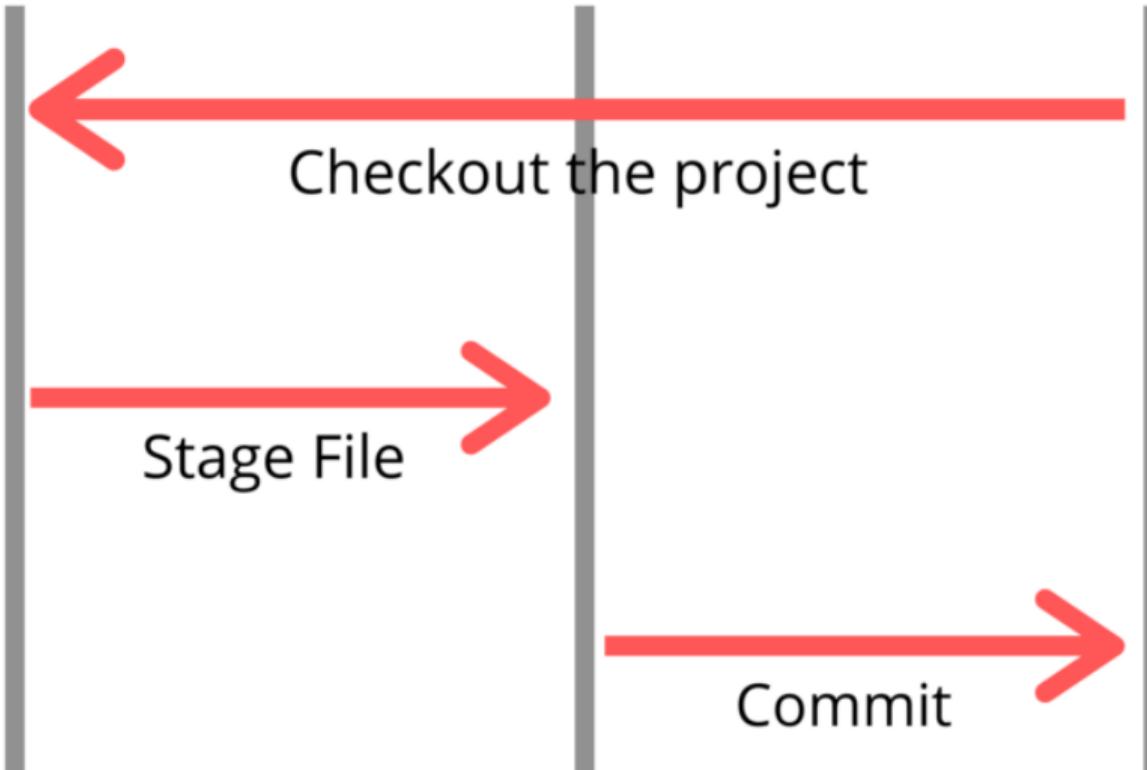




Working
Directory

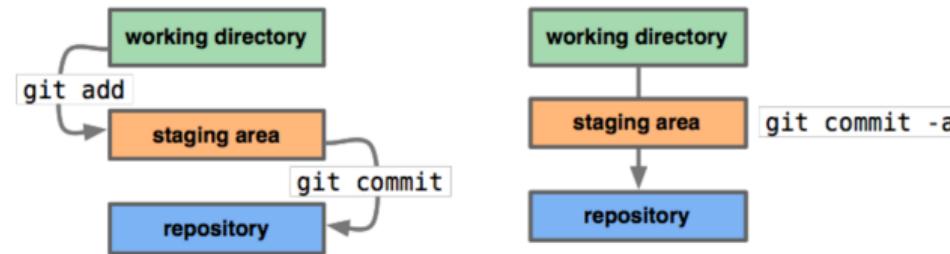
Staging
Area

Git Directory
(Repository)



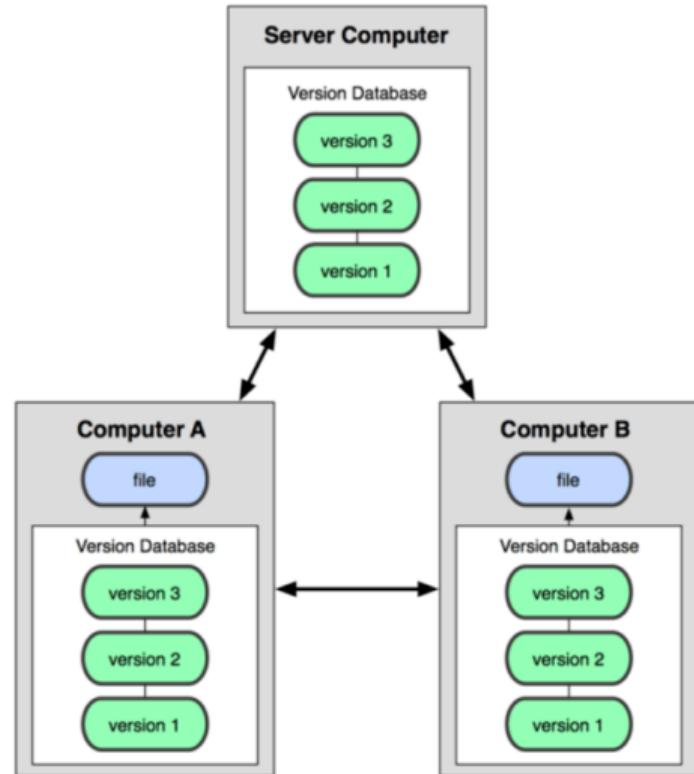
Área de ensayo (Staged area)

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



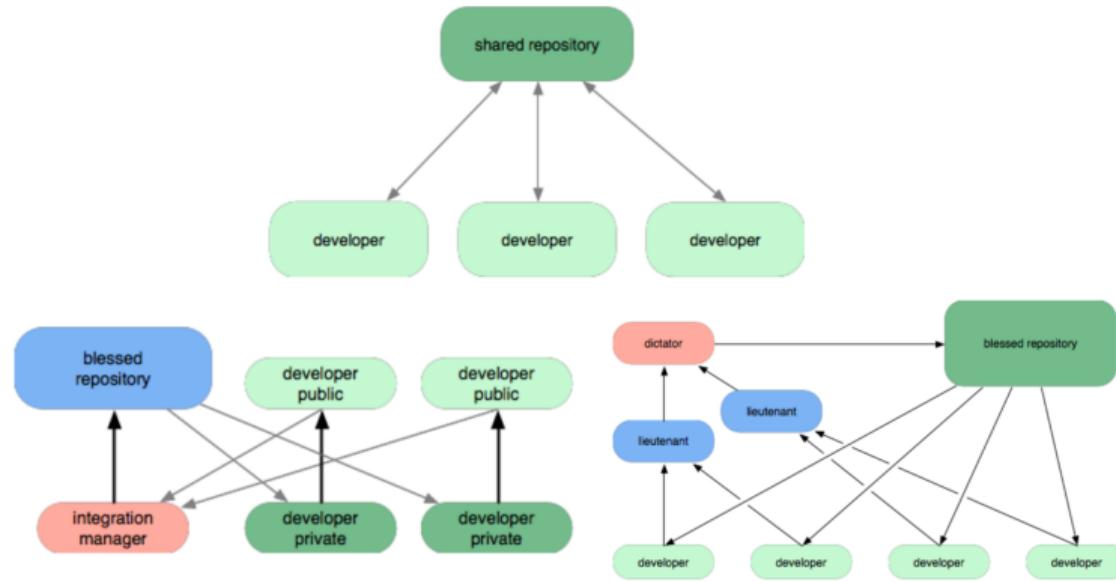
Distribución

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



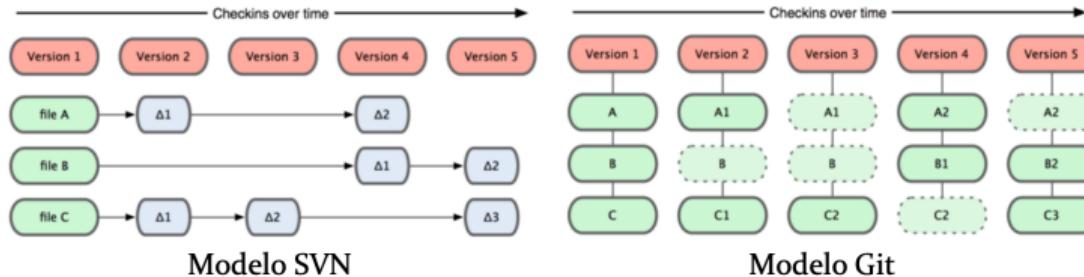
Distribución

- Permite múltiples flujos de trabajo.



Algunos 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



Comandos Git

Configuración

Configuración

Consulta

- `git config --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
- git config --global user.name "FIRST_NAME LAST_NAME"
- git config --global user.email "MY_NAME@example.com"
- Verifique cat .git/config

Parámetros interesantes

- 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

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 --base <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 raiz del proyecto

- NO en el caso de --base

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

Gestión del área de ensayo

Verificar el estado de la *staging area*

- `git status`: Cambios pendientes de commit
- `git diff`: Muestra los cambios de archivos modificados pero NO añadidos al staging area
- `git diff --cached`: Muestra los cambios de archivos modificados que SI están añadidos al staging area

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 realizá

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

Renombrando

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

Git se da cuenta de que estamos renombrando el archivo debido a la firma del archivo.

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

Hash

Hash

Un hash en Git es un identificador único de 40 caracteres alfanuméricos que se utiliza para identificar una confirmación o commit. Puedes identificar un hash al buscarlo en la salida de los comandos git log, git show o git diff.

Buscar el culpable

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

Gestión del área de ensayo

Error en el último commit

- git commit --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

- git checkout -- <archivo>
- git checkout -- index.html

Pasos para regresar a una versión anterior

Para regresar a una versión anterior utilizando Git, sigue estos pasos:

- Identifica el hash del commit al que deseas volver utilizando el comando `git log`. Copia el hash al portapapeles.
- Ejecuta el comando `git checkout <hash>` en la terminal, donde `<hash>` es el hash que copiaste.
- Ahora estás en el commit seleccionado y puedes ver el estado del proyecto en ese punto utilizando el comando `git status`. Puedes hacer cambios o simplemente explorar el proyecto en este estado.
- Si deseas volver a la versión más reciente, ejecuta el comando `git checkout <branch_name>`, donde `<branch_name>` es el nombre de la rama actual.
- Si hiciste cambios en el commit al que volviste y deseas aplicar esos cambios a la rama actual, crea un nuevo commit con los cambios utilizando el comando `git commit -m "mensaje del commit"`.
- Para volver al futuro, simplemente haz un nuevo commit en la rama actual y continúa trabajando normalmente.

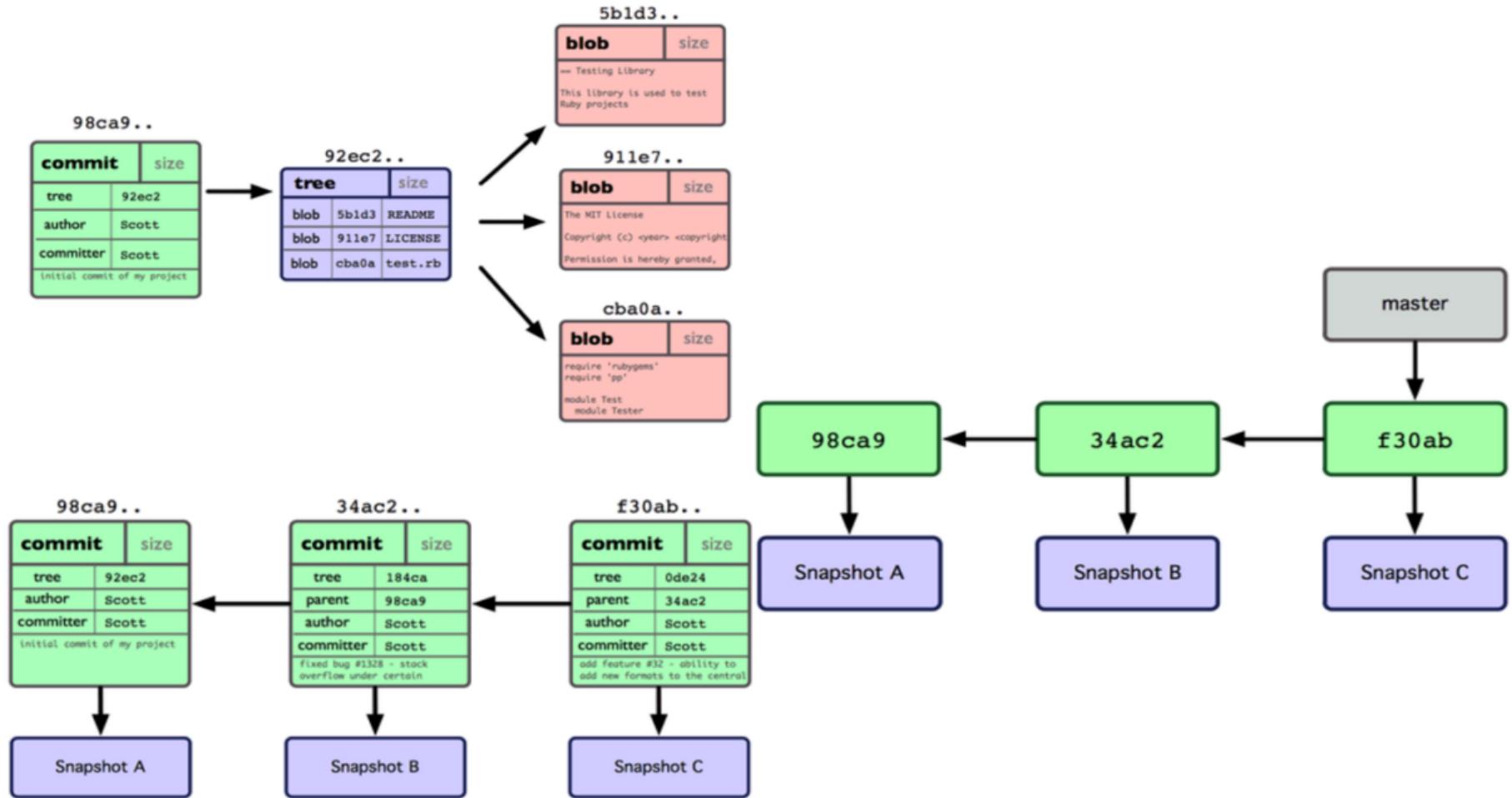
Pasos para regresar a una versión anterior

Es importante tener en cuenta que regresar a una versión anterior descarta todos los cambios realizados después de ese commit. Por lo tanto, debes estar seguro de que deseas volver a esa versión antes de hacerlo.

Descartar todos los cambios en archivos modificados
se han hecho cambios sobre varios archivos, pero no te gustaron y quieres volver al estado del último commit, eliminando de manera permanente todos los cambios a todos los archivos realizados

- `git reset --hard`

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

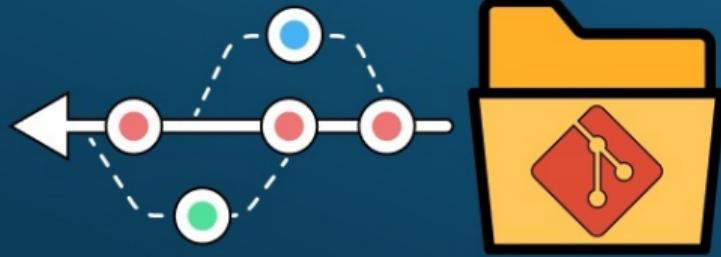
Crear un branch (local)

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

Pasar a trabajar a otro branch

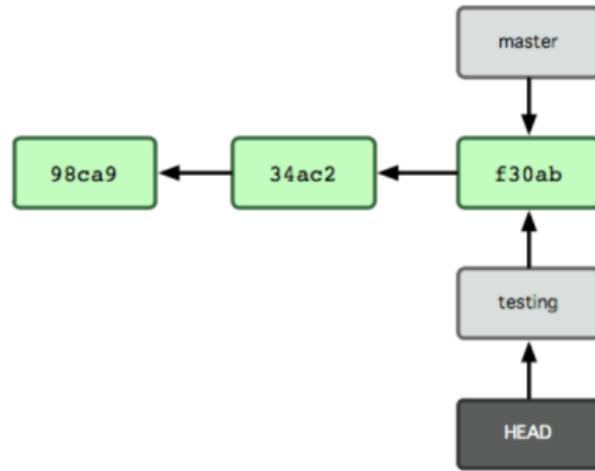
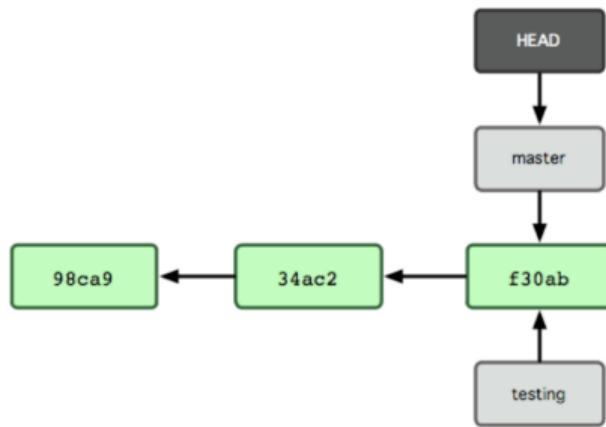
- `git checkout <nombre branch>`

RAMAS CON GIT



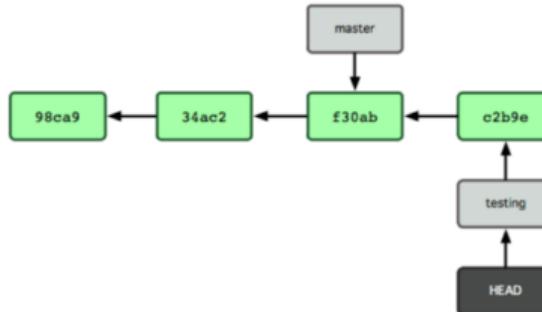
Git Branch workflow

Branches (ramas) en GIT

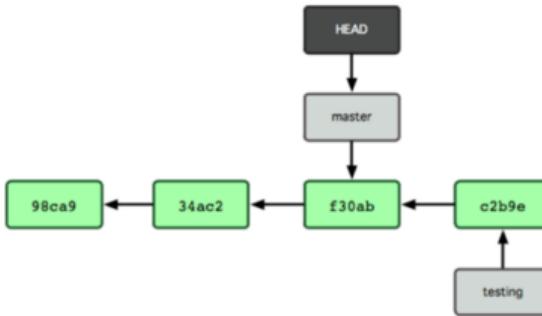


Branches (ramas) en GIT

Al hacer commit se realizarán sobre el branch activo

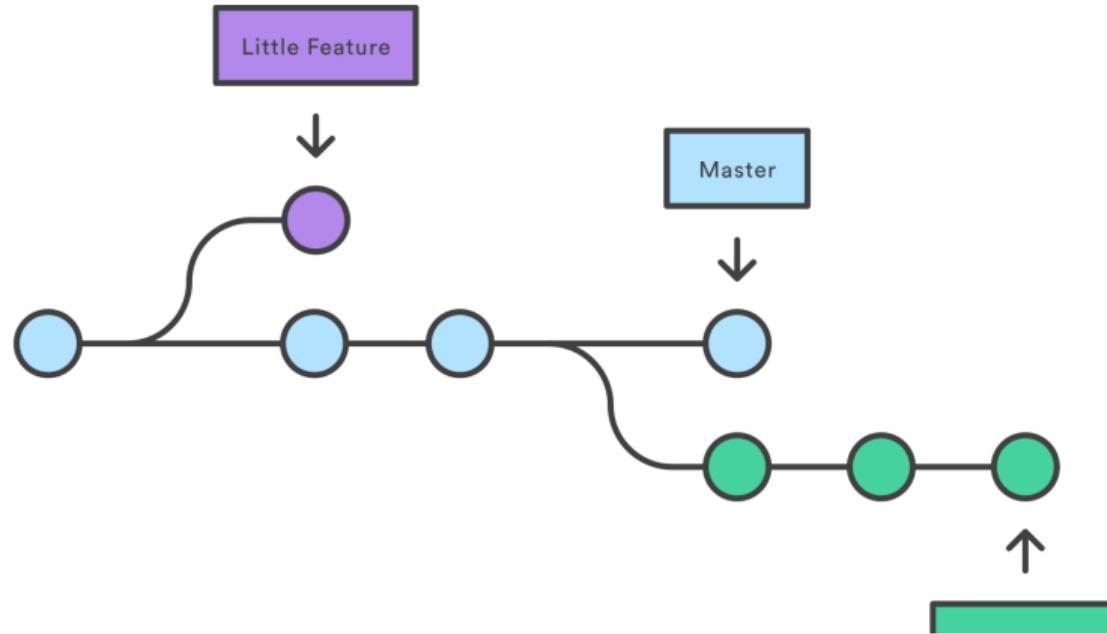


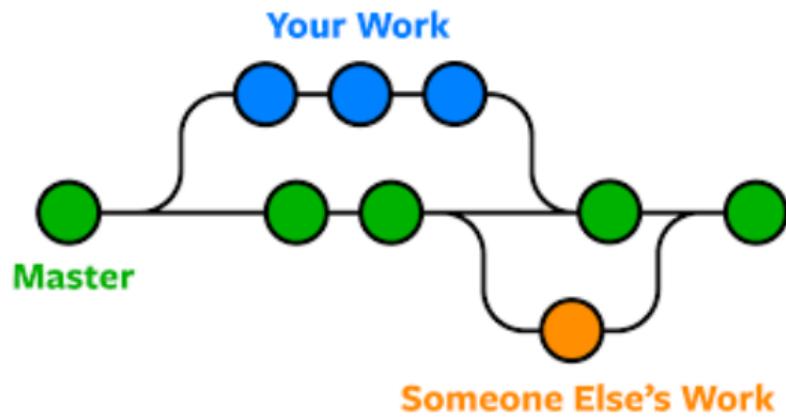
Podemos volver al branch master cuando queramos

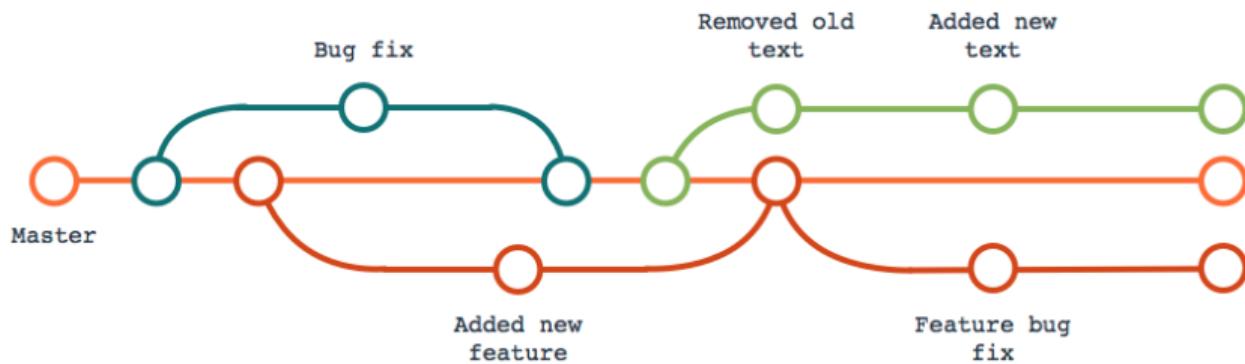


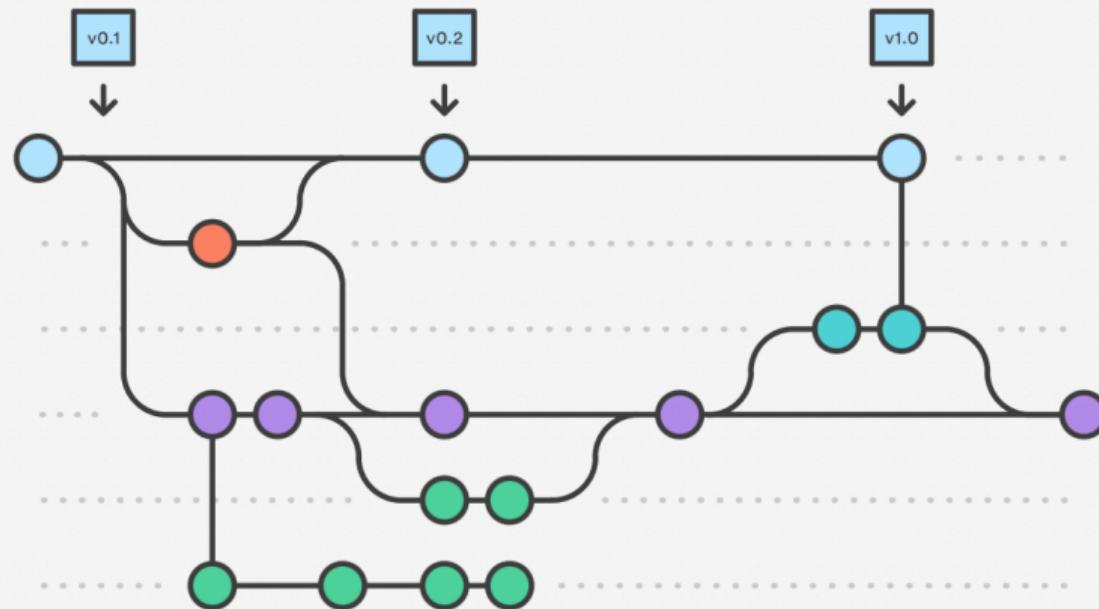
Flujo de trabajo con branches

- 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?

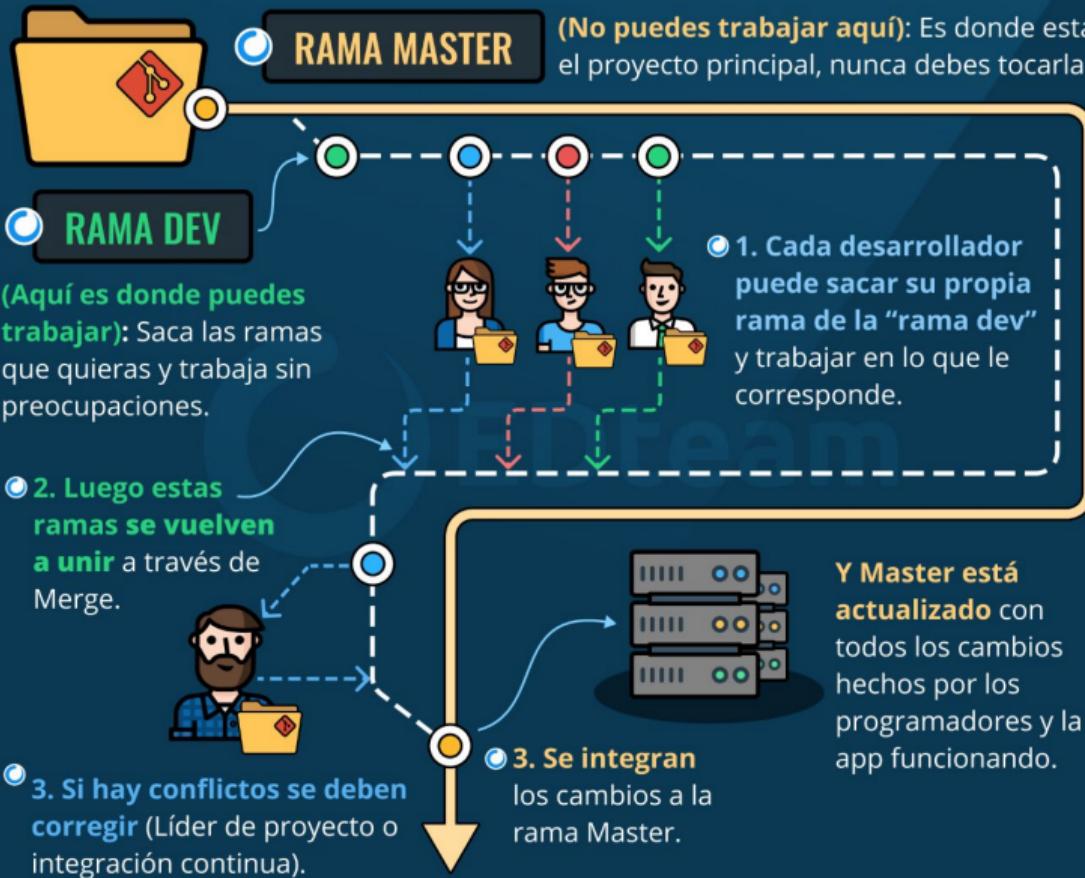
- git branch --merged

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

- git branch -d <branch>

¿Cómo trabajar en equipo con Git?

¿CÓMO TRABAJAR EN EQUIPO CON GIT?



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

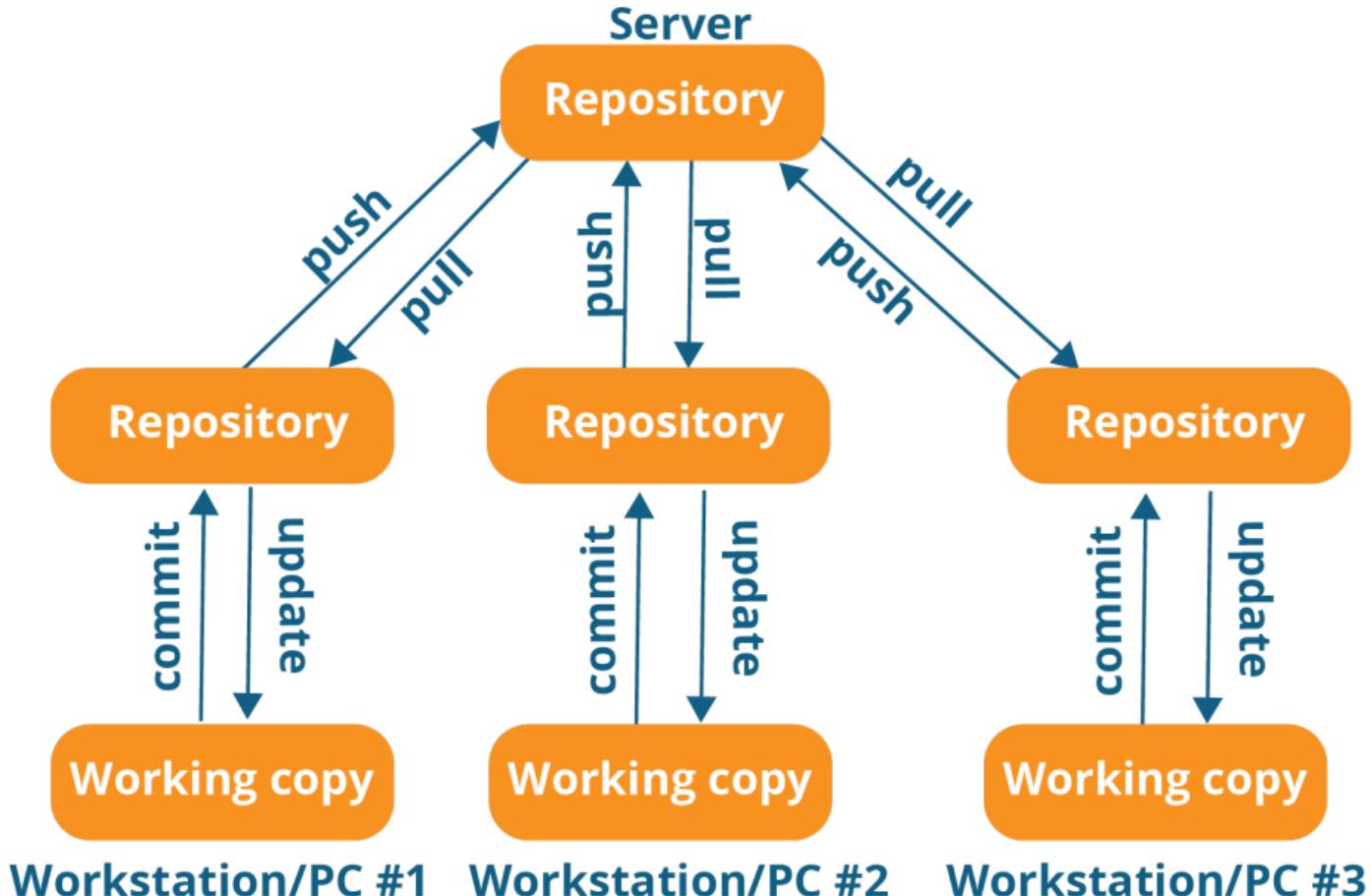
Colaboración con git: remotes

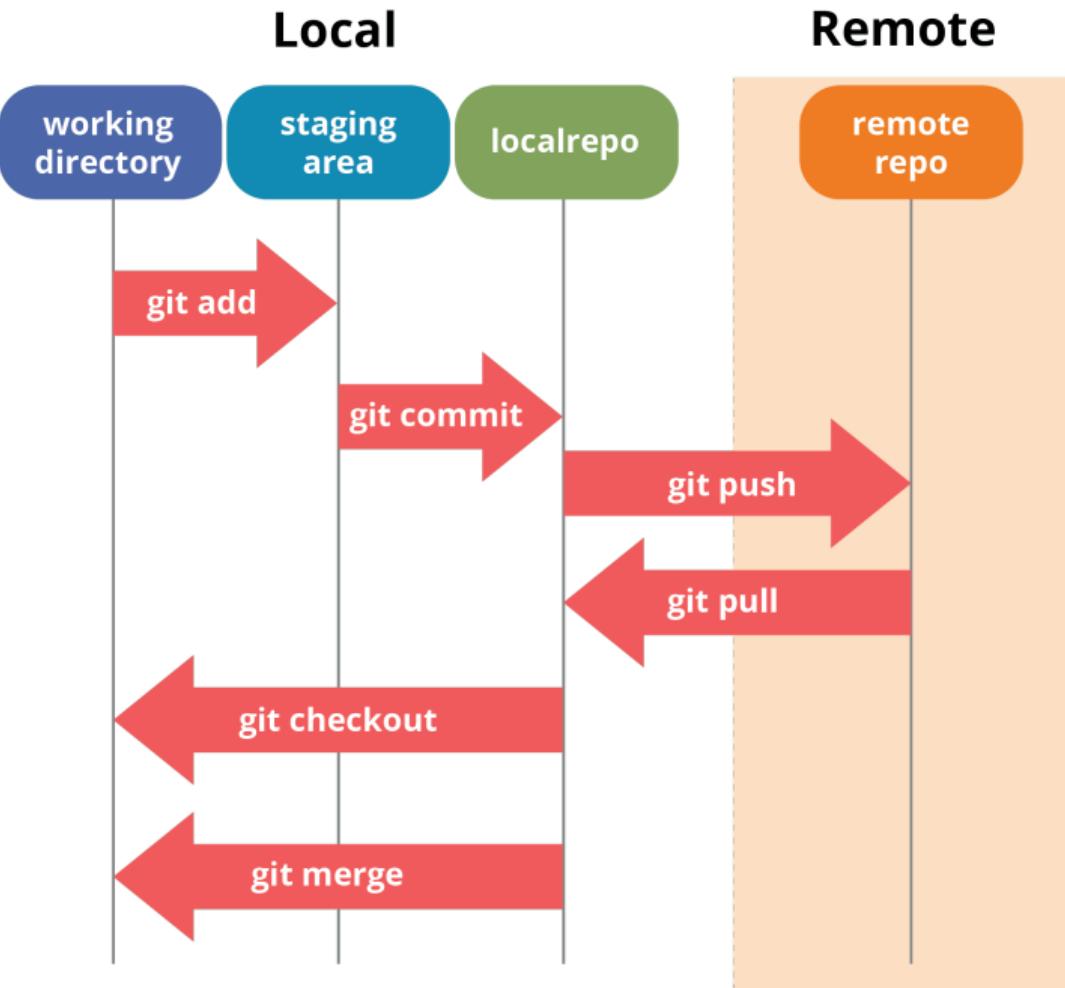
¿Cómo me traigo mi repositorio publico a mi máquina?

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

Ejemplos de flujo de trabajo en equipo

Distributed version control system





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

Colaboración con git: remotes

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.g...`

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

Colaboración con git: remotes

¿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 envio 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>`

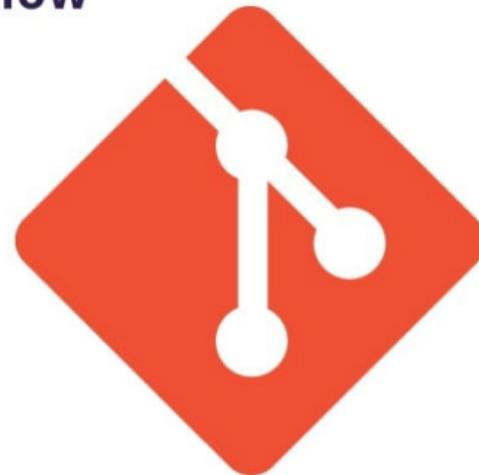
¿Cómo me traigo los cambios y los aplico?

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

Essential git commands every developer

Git Clone should know

- Git branch**
- Git checkout**
- Git status**
- Git add**
- Git commit**
- Git push**
- Git pull**
- Git revert**
- Git merge**
- Git remote**
- Git fetch**



GIT es un sistema de control de versiones

que nos ayuda a llevar el historial completo de modificaciones de un proyecto.

GIT INIT



Inicia un nuevo repositorio.

GIT ADD



Añade un archivo a la zona de montaje. `git add *` añade uno o más archivos a la zona de montaje.

GIT LOG



Se utiliza para listar el historial de versiones de la rama actual.

GIT RESET



Descompone el archivo, pero conserva el contenido del mismo.

GIT CLONE



Clona un repositorio existente.

GIT CONFIG



Establece el nombre del autor, el correo y demás parámetros que Git utiliza por defecto.



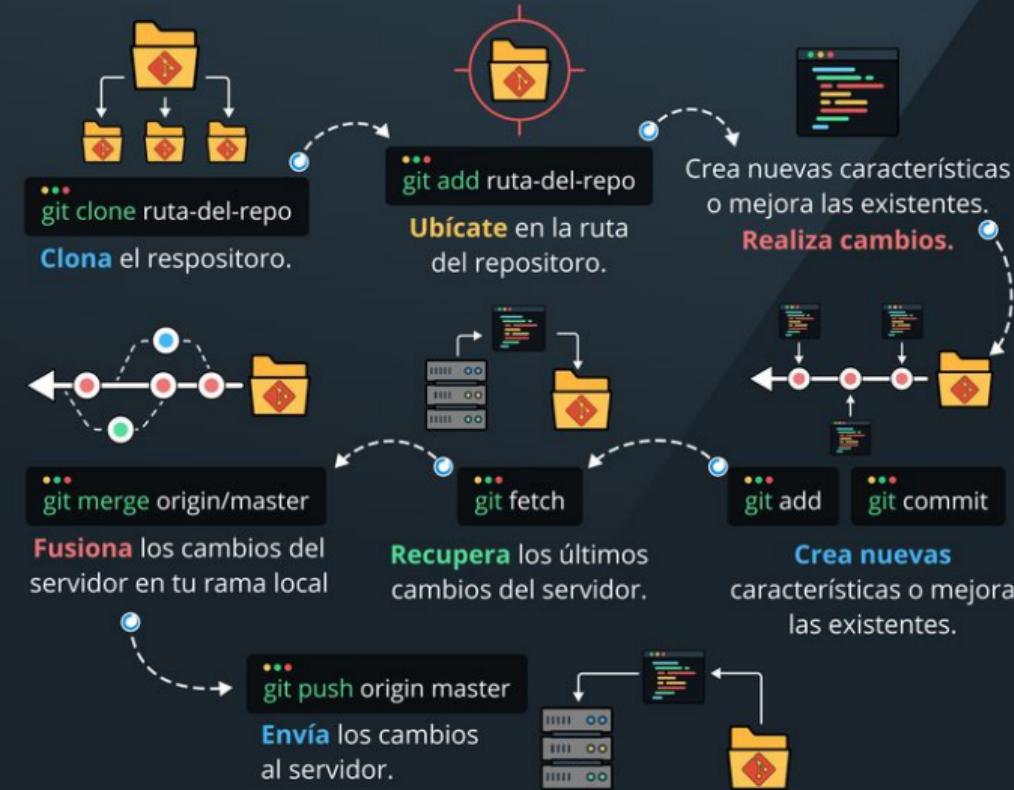
GIT STATUS

Enumera todos los archivos que deben ser confirmados.



Muestra las diferencias de archivo que aún no se ponen en escena.

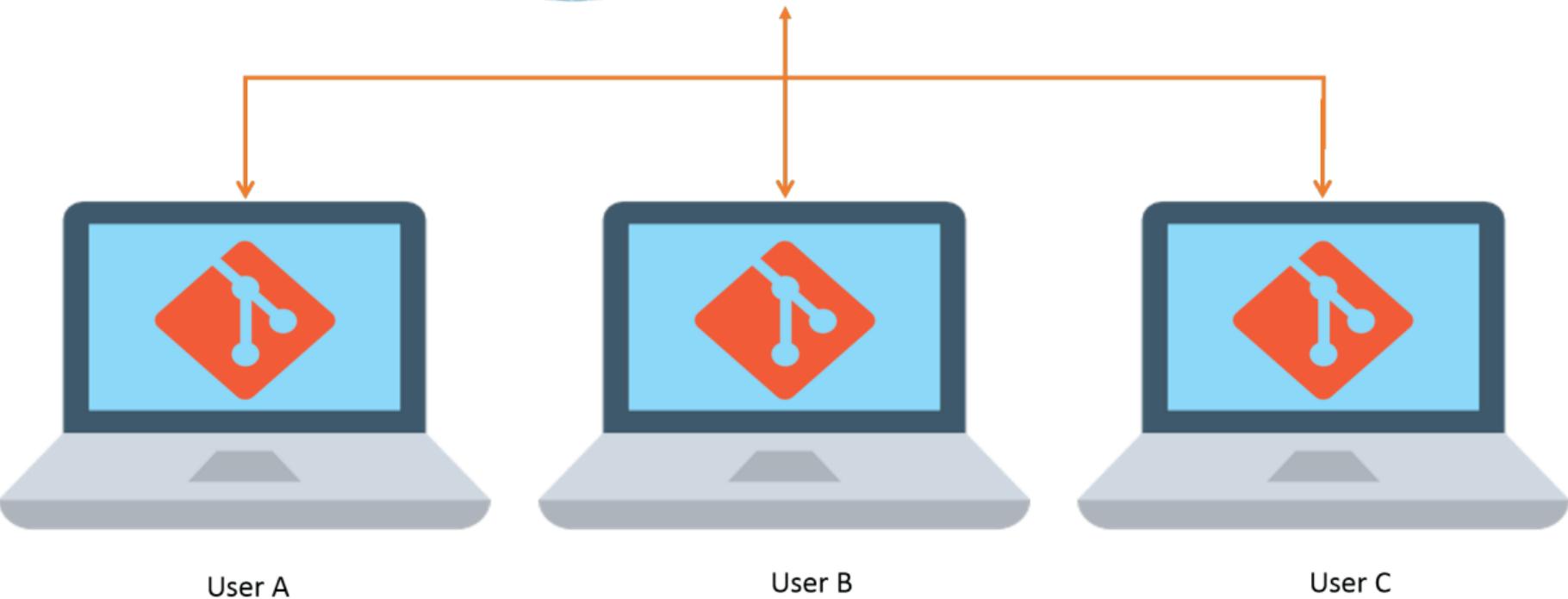
¿CÓMO CONTRIBUIR EN UN PROYECTO CON GIT?



GitHub



GitHub



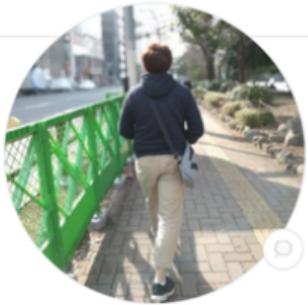


Microsoft

GitHub



Portafolio

**vn7**

vn7n24fkq

[Edit profile](#)

阮 18 followers · 65 following · ⭐ 530



Taiwan

✉ vn7n24fkq@gmail.com

🔗 <https://vn7n24fkq.github.io/>

Highlights

* Arctic Code Vault Contributor



Organizations

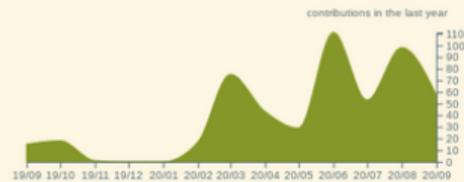


vn7n24fkq / README.md

[Send feedback](#)

vn7n24fkq (vn7)

- 844 contributions on GitHub
- 56 public repos
- Joined GitHub 4 years ago
- vn7n24fkq@gmail.com



Repos per Language (top 5)

- Java
- Rust
- Kotlin
- HTML
- Vim script



Most Commit Language

- Java
- Kotlin
- Rust
- JavaScript
- Vim script



Pinned

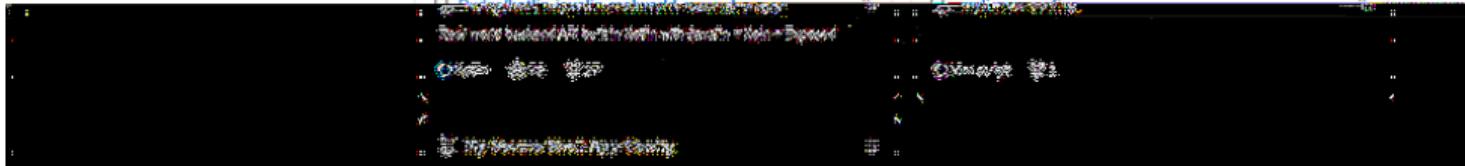
[github-profile-summary-cards](#)

JavaScript ⚡ 1

[tipsy/javalin](#)

A simple and modern Java and Kotlin web framework

Kotlin ⭐ 4.2k ⚡ 367

[Customize your pins](#)



Abid Ali Awan

kingabzpro

Love building machine learning product and working on MLOps. Currently, focusing on content creation, building brand, and startup.

[Edit profile](#)

33 followers · 12 following

eXfinite

Islamabad

kingabzpro / README.md

Hey , I'm Abid!

- 🏆 I am a certified data scientist professional, who loves building machine learning models.
- ✍️ I write blogs on data science and machine learning topics.
- 💻 I'm looking for an experience in the field of technical copywriting.
- ❤️ I'm open for collaborations in MLOps and Applied Machine Learning domains.
- 🎯 Goal (2022): Build an NLP startup (exfinite) for early detection of mental illness.

Competition Performance

[KAGGLE](#) [Z ZINDI](#) [DEVPOST](#) [DRIVENDATA](#) [DS COMPETITIONS](#) [DEEPNOTE](#) [BATTLE SNAKE](#)

Contact me on:



Road to Kaggle Grandmaster



WOLOF-ASR-Wav2Vec2 Public



Audio Preprocessing and finetuning of wav2vec2-large-xlsr model on AI4D Baamtu Datamation - Automatic Speech Recognition in WOLOF Data.

Jupyter Notebook 9 4

French-to-Fongbe-and-Ewe-MT Public



The objective of this challenge is to create a machine translation system capable of converting text from French into Fongbe or Ewe.

Jupyter Notebook 5 1

orchest/orchest Public



Build data pipelines, the easy way

Python 2.8k 150

v1a0/sqllex Public



The most pythonic ORM (for SQLite and PostgreSQL). Seriously, try it out!

Python 72 8

FastAPI-ML-Project Public



Learning and building API using Fast API

Python 7 3

DAGsHub/audio-datasets Public



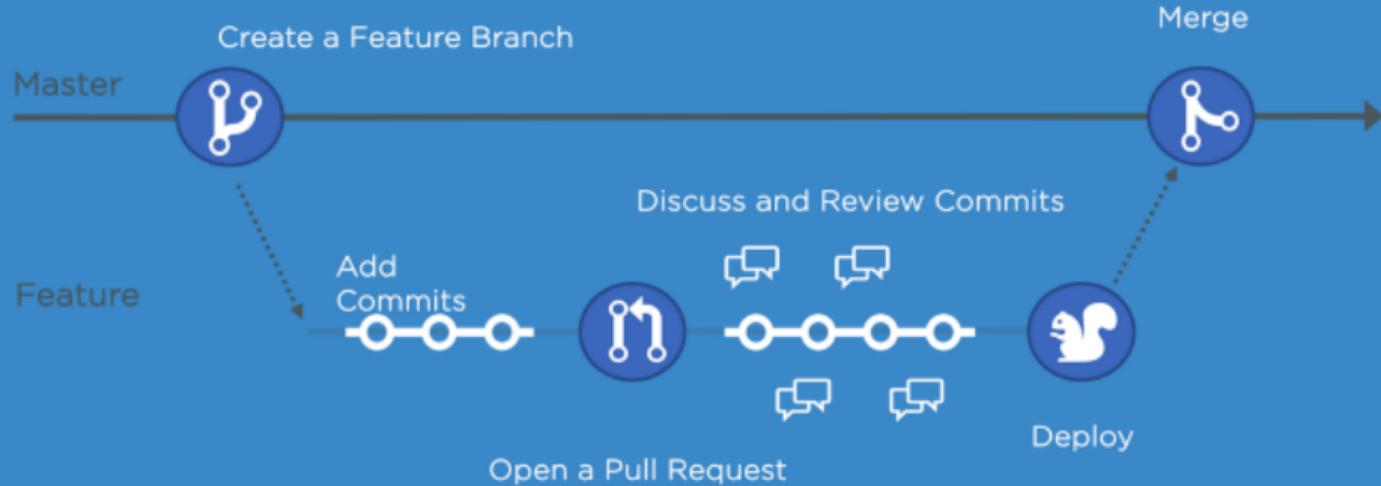
open-source audio datasets

34 16

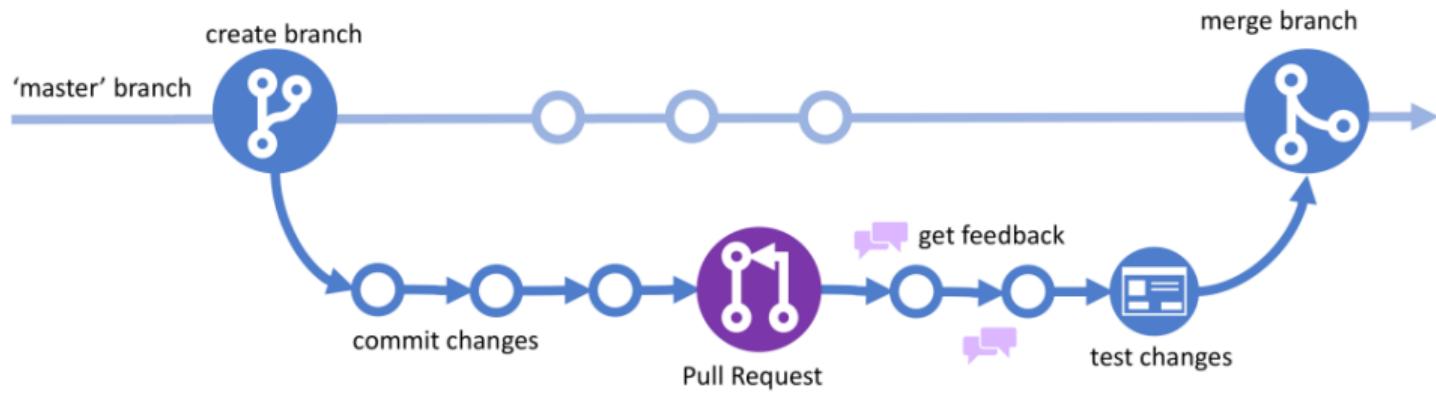
Glosario GitHub

- **Fork (Bifurcar)**: Es la acción de crear una copia independiente de un repositorio existente en una cuenta personal.
- **Pull Request (Solicitud de extracción)**: Es la acción de proponer cambios en un repositorio que se fusionarán después de la revisión.
- **Clone (Clonar)**: Es la acción de copiar un repositorio existente en una nueva ubicación.

GitHub Flow



GitHub Flow





GITHUB



GITLAB



- **Máximo de 3** colaboradores gratis.



- Ofrece **hasta 500 MB** gratis por repositorio.



- **El código se puede descargar** como archivo.



- Opción de **lectura o escritura** por repositorio.



- Supera los **35 millones** proyectos.



- Fue comprada por Microsoft **¿Bueno o malo? ¿Qué opinas?**



- **No hay límite de** colaboradores.



- El límite gratis por repositorio es de **10 GB**.



- Permite incluir **adjuntos en un issue** o problema.



- Puedes **proporcionar acceso** a determinadas partes de un proyecto.



- Más de **100 mil** proyectos.



- Opera bajo **licencia de código abierto**.

Cloning

Cloning

Hay dos métodos principales para clonar un repositorio: la sintaxis HTTPS y la sintaxis SSH.

- Mientras que la clonación SSH se considera generalmente un poco más segura porque tienes que utilizar una clave SSH para la autenticación, la clonación HTTPS es mucho más simple y la opción de clonación recomendada por GitHub.