Sistemas de control de versiones

Requisito indispensable mantener registro y seguimiento de cambios realizados sobre sourceocde a lo largo del tiempo, permitiendo también el trabajo en equipo de diferentes partes de desarrollo de software.

Se puede saber autor, fecha y notas escritas sobre cambios realizados: así como volver a versiones previas para analizar causas y consecuencias de errores.

Creación y fusión de ramas:al tener varias personas trabajando a la vez en un team se beneficia con c/u una tarea y workflow diferente, lo que posteriormente se puede fusionar a una rama principal. Sistemas de ctrl de versiones tienen mecanismos para que los cambios entre ramas no entren en conflicto y asegurar funcionalidad/integración.

Trazabilidad de cambios de software: poder conectar al sist de ctrl de versiones con software de ctrl de proyectos y seguimiento de errores ayuda con análisis de causa de problemas y recopilación de info.

**Concepto de versión – revisión – edición:** de proyecto/source code hace referencia al estado en el que se encuentra el mismo en X momento de su desarrollo.

Los de tipo centralizado almacenan todo el código en un único repositorio o server que contiene todos los archivos versionados, representando un único punto de falla para todo el equipo. Los de tipo distribuido permiten continuar el trabajo aun cuando el repositorio de referencia no está disponible: se descarga la última copia de code y se replica completamente al repositorio con cambios históricos o versiones = sin un server deja de funcionar y los sistemas colaboraban a través de él, cualquiera de los repositorios disponibles de algún cliente puede ser copiado al server para restaurar.

GIT

Sistema de código abierto de control de versiones distribuido que funciona bajo cualquier plataforma (Linux, Win, Mac) e integrado en una amplia variedad de entornos de desarrollo (IDEs <https://www.redhat.com/es/topics/middleware/what-is-ide>).

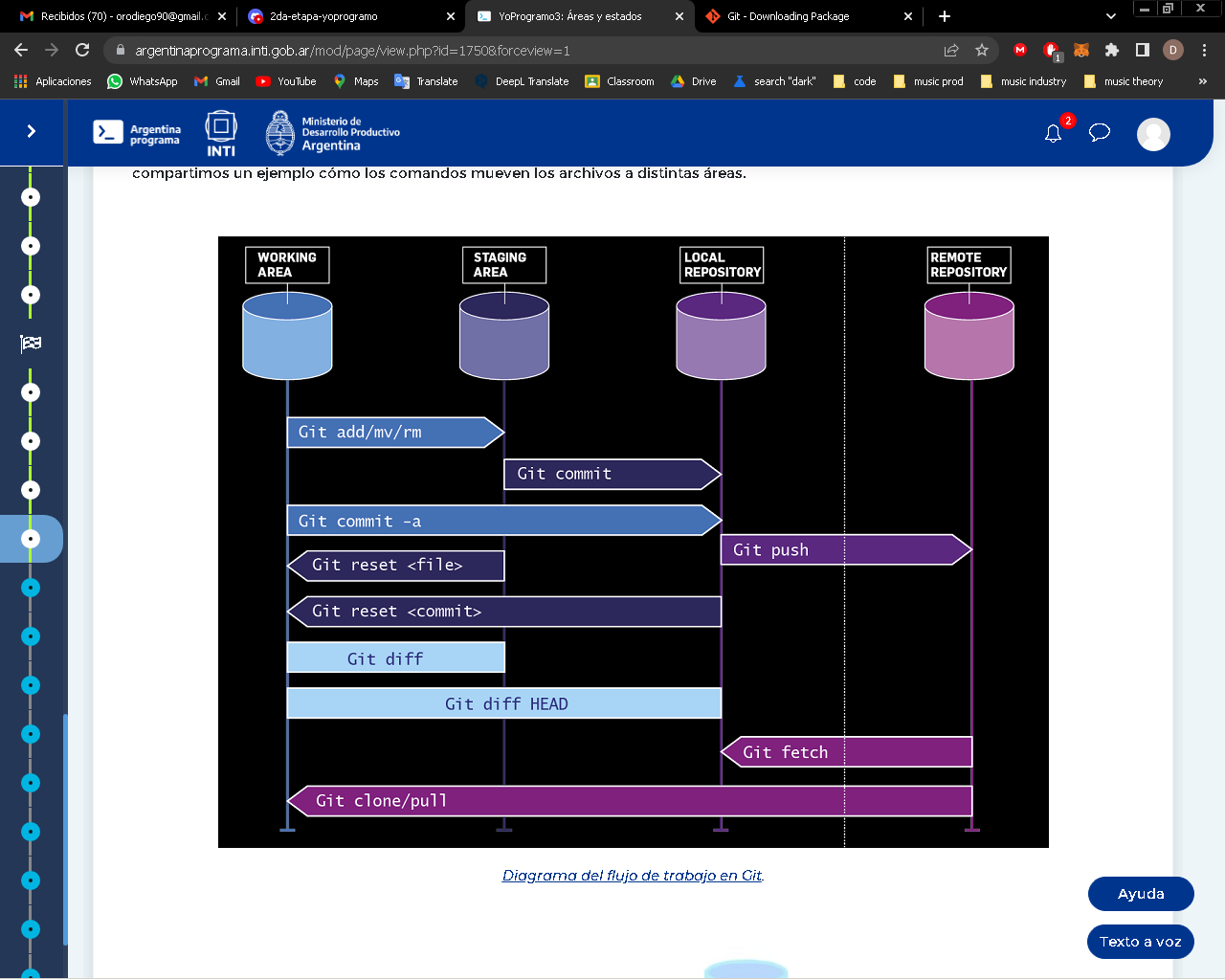
Rendimiento: los algoritmos implementados aprovechan el profundo conocimiento sobre atributos comunes de auténticos árboles de archivos de source code, modificaciones en el tiempo y patrones de acceso. Formato de objetivo de repository file de GIT emplea combo de codificación delta (almacena diferencias de contenido) y compresión, guirdando explícitamente contenido de directorios y objetos de metadatos de versiones.

Seguridad: principal prioridad es conservar integridad de source code gestionado. Contenido de archivo y relaciones con directorios-versiones-etiquetas-confirmaciones están protegidos con algoritmo de hash criptográficamente seguro llamado “SHA1”. Se salvaguarda el code e historia de cambios frente a modificaciones accidentales-maliciosas, garantizando un historial totalmente trazable.

<https://latam.kaspersky.com/blog/que-es-un-hash-y-como-funciona/2806/>

Flexibilidad: en la capacidad para varios workflow de desarrollo no lineal, eficiencia en proyectos grandes como chicos, compatibilidad con sistemas y protocolos. Ideado para posibilitar ramificación y etiquetado como procesos de primera importancia y las operaciones afectantes (fusión-reversión) también son almacenadas en historial.

**ÁREAS Y ESTADOS:** por los que pueden pasar los archivos durante desarrollo. Siempre es importante saber en cuál área están los archivos, para no perder cambios por confusión.



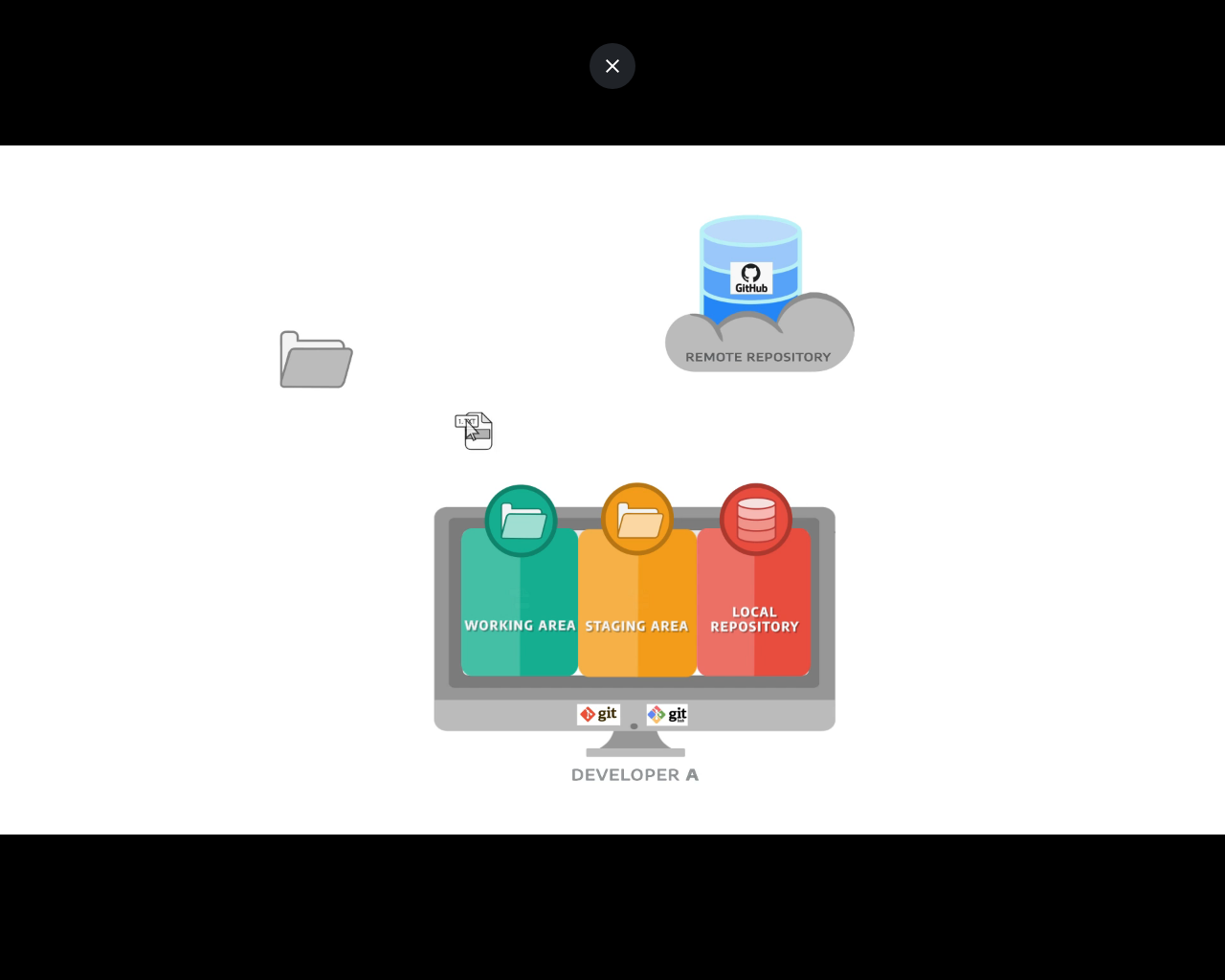
<https://www.reddit.com/r/git/comments/99ul9f/git_workflow_diagram_showcasing_the_role_of>

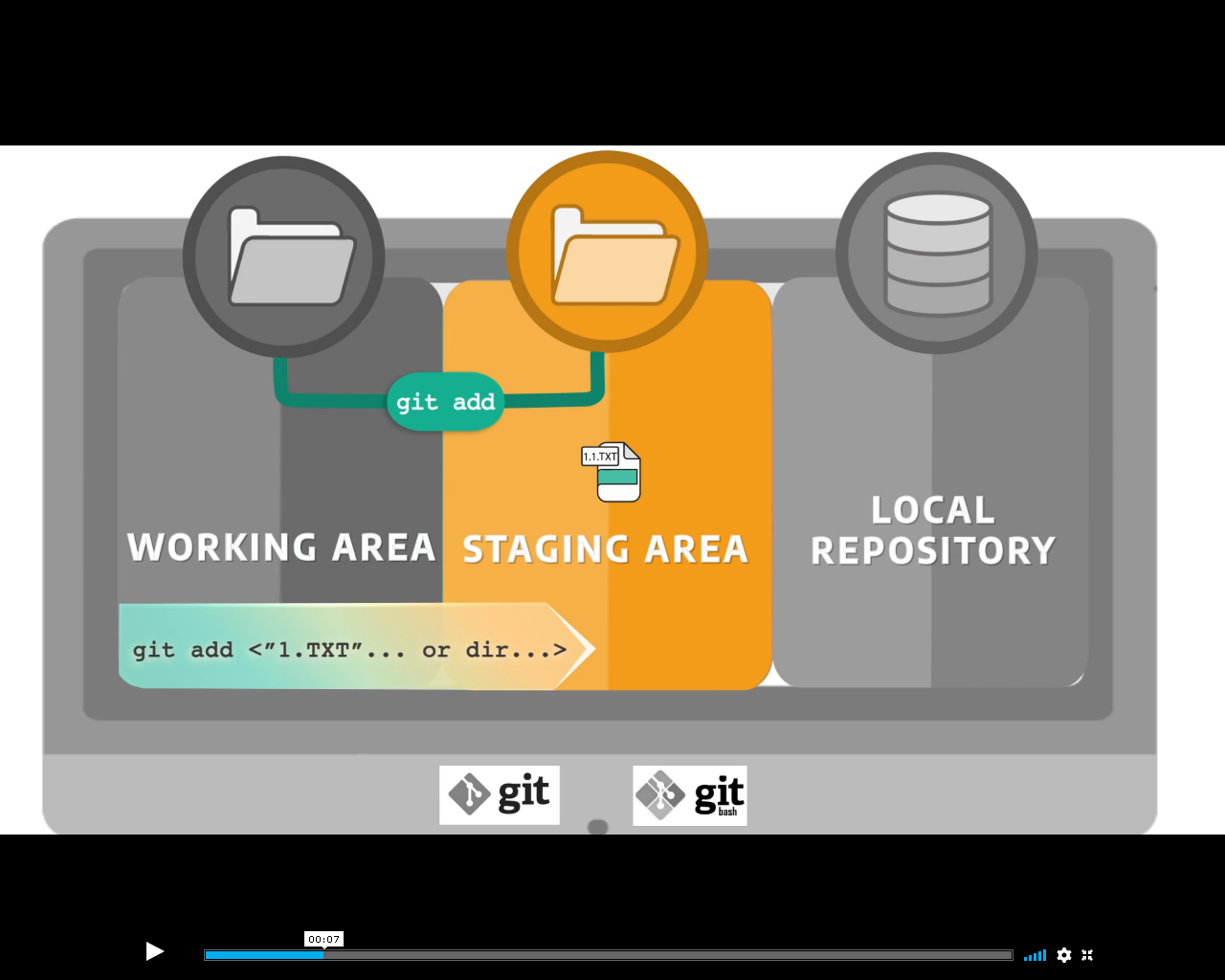
**Directorio de trabajo – working directory:** copia de versión de proyecto, fiels sacados de la DB comprimida y colocados en disco para uso-mod.

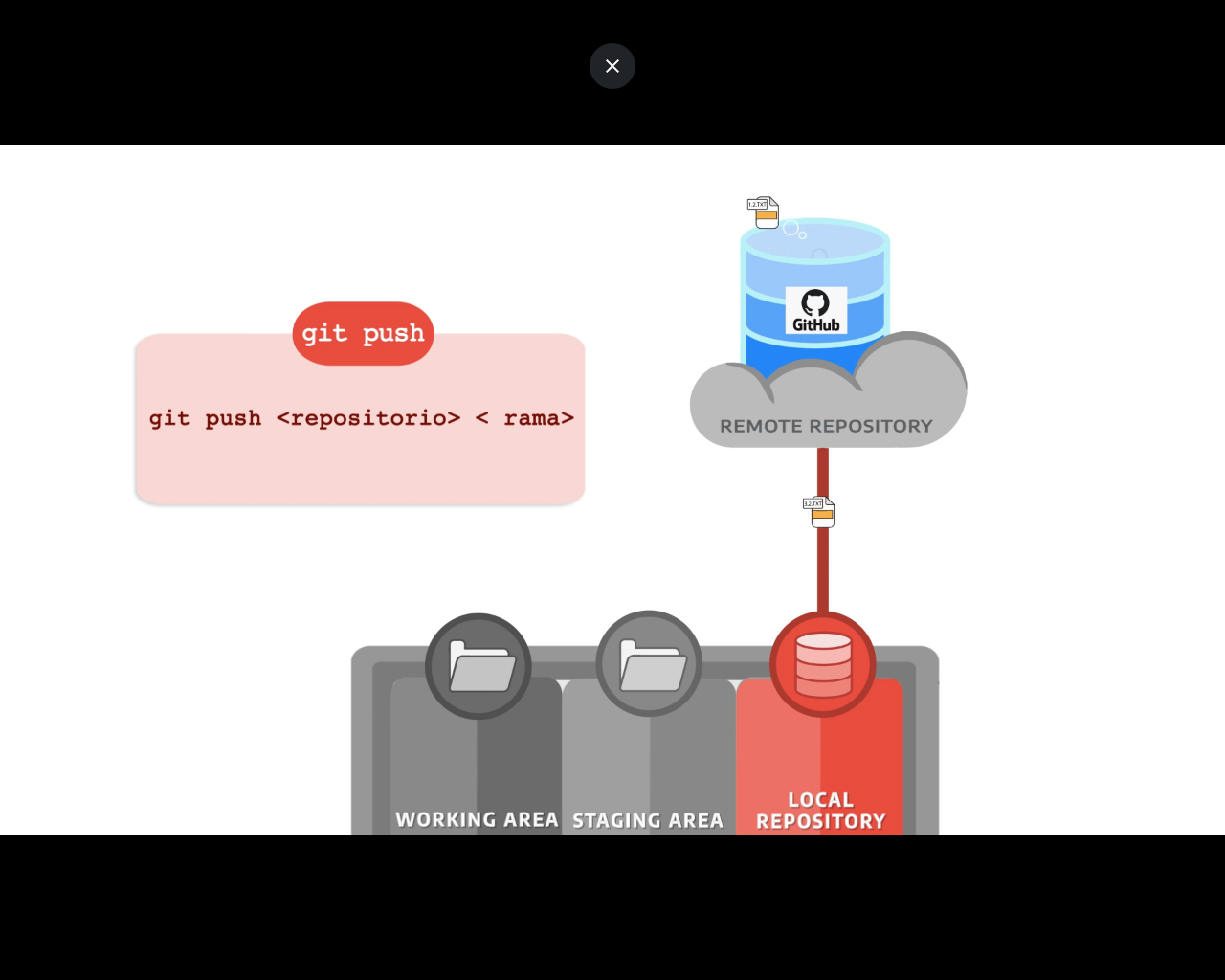
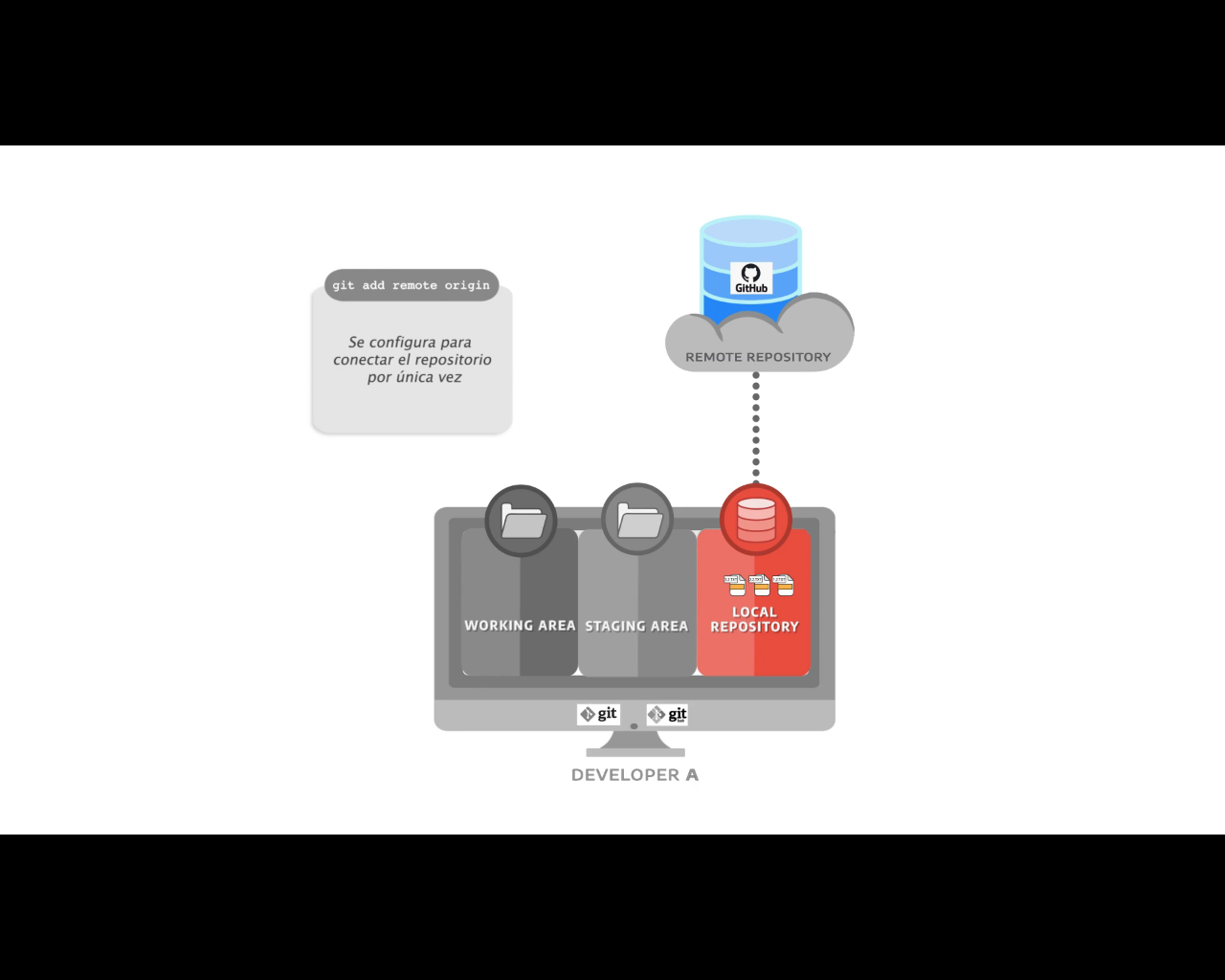
**Área de preparación – staging area:** file dentro de directorio de Git, contiene info de lo que va a ir en la próxima confirmación.

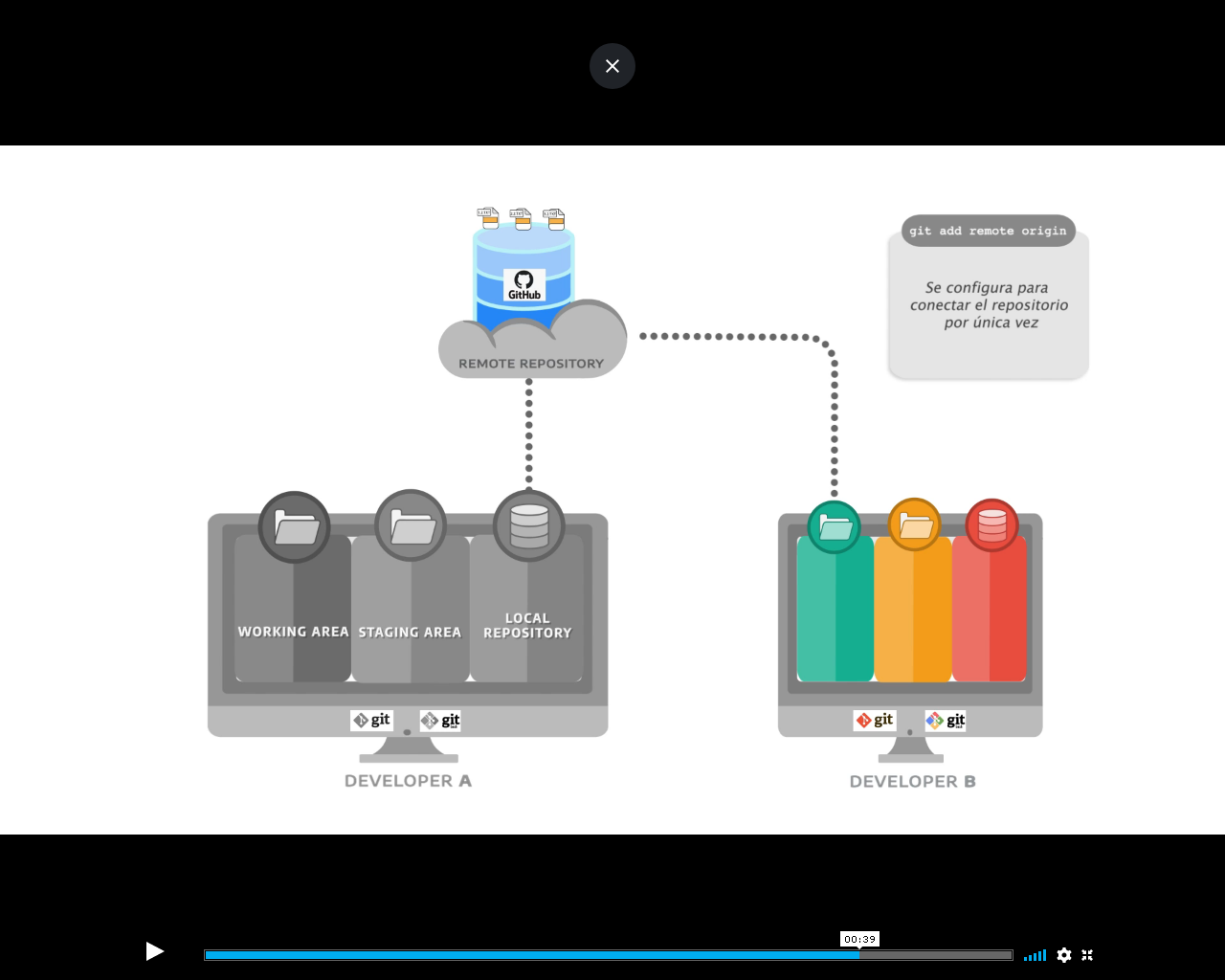
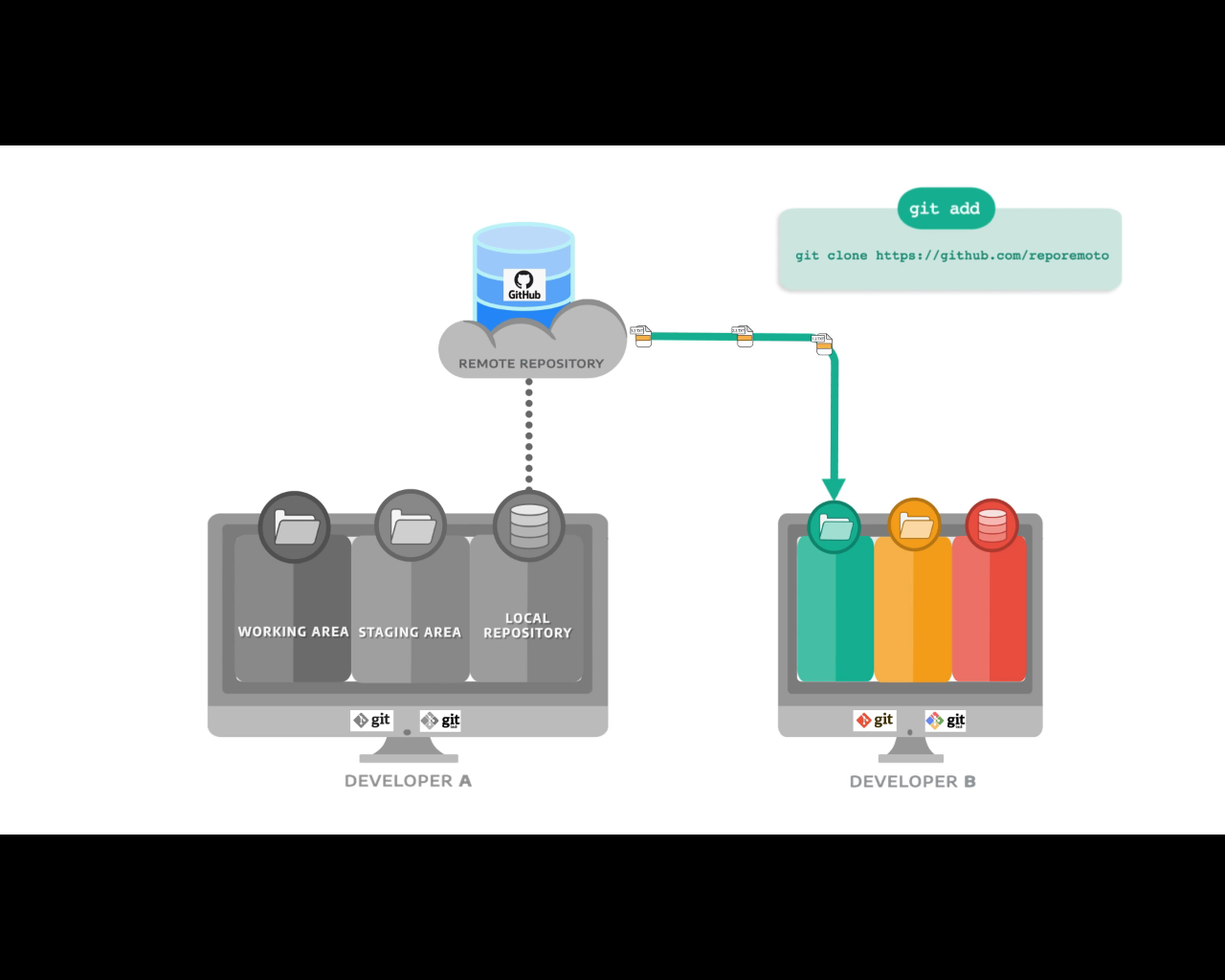
**Directorio de Git – local repository:** lugar de almacenamiento de metadatos y db de objetos del proyecto, copiado al clonar un repositorio de otra fuente. Archivos que ya están funcionando

**Repositorio remoto – remote repository:** repositorio en server remoto, con el que eventualmente se sincronizan trabajos entre integrantes del team.







**Comandos básicos:**

git init: comando para inicializar directorio como repositorio Git, ejecutado dentro de directorio de proyecto y, como resultado, crea subdirect .git que contiene files para realizar seguimiento de cambios, etiquetas, etc.

git add <file>: después de mod., creación o eliminación de file, los cambios queda únicamente en área de trabajo, por lo que se pasan a área de preparación con este comando para incluir en la siguiente confirmación-commit.

git status: permite conocer en qué estado se encuentran los files.

git commit: confirmar cambios registrados en área de preparación, se pasan cambios a repositorio local.

git push: para enviar confirmaciones registradas en repositorio local a remoto.

git pull: inverso de git push, trae cambios a repositorio local y los deja disponibles para su modificación o revisión en área de trabajo. Se usa cuando ya se tiene repositorio local vinculado a uno remoto, al igual que con git push.

git clone: en caso de necesitar “bajar” un repositorio remoto de algún proyecto existente. Genera directorio (con nombre de repositorio o uno especificado) que contiene todo lo del proyecto, y el subdirectorio .git necesario para gestionar cambios y lo pertinente al repositorio Git.