MIND HUB.


GIT
GITHUB

Índice

GIT

Git es un software de control de versiones diseñado por Linus Torvalds, pensando en la eficiencia, la confiabilidad y compatibilidad del mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos de código fuente. Su propósito es llevar registro de los cambios en archivos de computadora incluyendo coordinar el trabajo que varias personas realizan sobre archivos compartidos en un repositorio de código.

Características:

Gestión distribuida: Git le da a cada programador una copia local del historial del desarrollo entero, y los cambios se propagan entre los repositorios locales. Los cambios se importan como ramas adicionales y pueden ser fusionados en la misma manera que se hace con la rama local.

Gestión eficiente de proyectos grandes: dada la rapidez de gestión de diferencias entre archivos, entre otras mejoras de optimización de velocidad de ejecución.

GIT

Todas las versiones previas a un cambio determinado, implican la notificación de un cambio posterior en cualquiera de ellas a ese cambio.
Fuerte apoyo al desarrollo no lineal, por ende rapidez en la gestión de ramas y mezclado de diferentes versiones. Git incluye herramientas específicas para navegar y visualizar un historial de desarrollo no lineal.

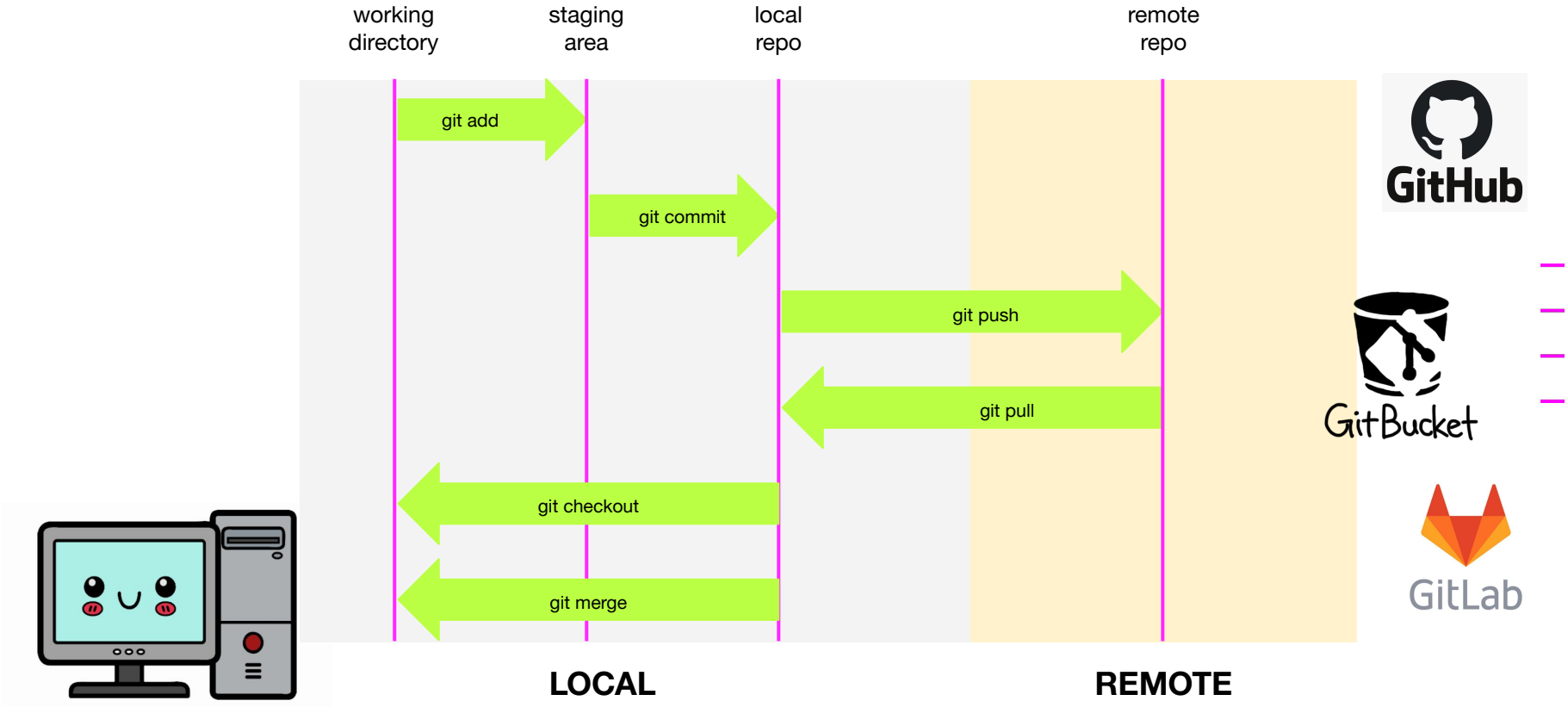
Los renombrados se trabajan basándose en similitudes entre ficheros, aparte de nombres de ficheros, pero no se hacen marcas explícitas de cambios de nombre con base en supuestos nombres únicos de nodos de sistema de ficheros, lo que evita posibles y desastrosas coincidencias de ficheros diferentes en un único nombre.

INSTALAR DESDE: **https://git-scm.com/downloads**

Una vez iniciado git empieza a monitorear nuestros archivos en 3 directorios de nuestra computadora (local):

- Working directory: los archivos que veo en mi directorio de trabajo
- Local repo: base de datos “oculta” que aloja las diferentes versiones de los archivos
- Staging area: archivos preparados para re-versionar.

GIT


working directory
staging area
local repo
remote repo
git add
git commit
git push
git pull
git checkout
git merge
LOCAL
REMOTE
GitHub
GitBucket
GitLab

Configuración

Una vez instalado, desde la consola, debemos configurar por primera y única vez git con los siguientes comandos:

```
git config – global user.name Mindy
git config – global user.email mindy@mindhub.com
```

Esta configuración le informa a git: quién ejecuta los cambios de versiones y las actualizaciones correspondientes.

Luego, si es un proyecto nuevo, debemos inicializar un repositorio en una carpeta con el siguiente comando:

```
git init
```

También podemos traernos un repositorio de algún usuario de git a nuestra computadora con:

```
git clone https://github.com/nombreUsuario/nombreProyecto.git
```

Comandos

git status: nos informa

- la rama en donde estamos trabajando
- si la rama de trabajo está “al día” con la rama maestra o de origen (la rama remata), si no está al día, me avisa cuantas versiones estoy “adelantado”.
- si la rama tiene cambios/modificaciones para re-versionar, en este caso, se habilitan dos opciones:
  git add nombreArchivo1 archivo2 agrega estos archivos al stage area
  git add . agrega todos los archivos al stage area
  git add -A agrega todos los archivos al stage area
  git add -p permite ver los cambios en cada archivo y luego agregarlo al stage area
  git checkout —nombreArchivo1 borra archivos del stage area
- si la rama tiene archivos nuevos para versionar (untracked files) solamente me permite agregarlos (con git add . git add -A git add -p)
- archivos en el stage area, listos para comitear

git commit -m “nombre de la version”: genera una nueva versión de los archivos agregados, los archivos son parte del repositorio local

Comandos

git push origin master: envía las versiones a la rama maestra del repositorio remoto
git push origin nombreRama: envía las versiones a otra rama

git log: para ver todas las versiones del proyecto

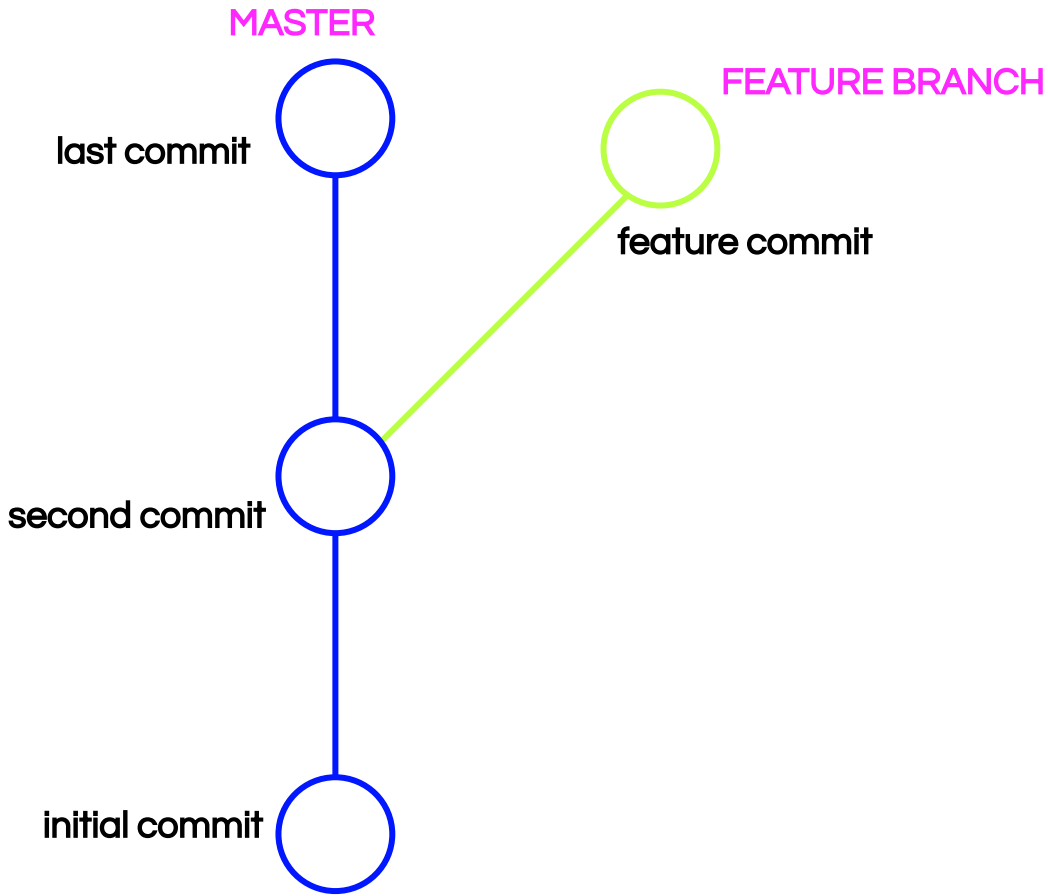
git checkout idDelCommitAntiguo traer una versión antigua al directorio de trabajo y te avisa que no estás trabajando en HEAD (última versión)

git checkout master: vuelvo a la última versión (HEAD).

al eliminar archivos, siguen quedando las versiones “viejas” de ese archivo, para borrarlos definitivamente:

git reset HEAD nombreArchivo.js para quitar el archivo del commit

Ramas


MASTER
FEATURE BRANCH
last commit
feature commit
second commit
initial commit


GIT
GITHUB
trabajo colaborativo

Comandos para trabajar colaborativamente

git branch nuevaRama: crea una nueva rama (el nombre debería ser lo más descriptivo y sencillo posible)

git checkout nuevaRama: me muevo a la nueva rama, con git status verifico que me cambié de rama

git push origin nuevaRama: pushea hacia la nueva rama

git checkout idDelCommitAntiguo traer una versión antigua al directorio de trabajo y te avisa que no estás trabajando en HEAD (última versión)

git checkout master: vuelvo a la última versión (HEAD).

al eliminar archivos, siguen quedando las versiones “viejas” de ese archivo, para borrarlos definitivamente:

git reset HEAD nombreArchivo.js para quitar el archivo del commit

¡Muchas gracias!

MIND
HUB.