Academia Java

Investigación Git

Capacitador:

Miguel Angel Rugerio

Capacitado:

Hernandez Soledad Angel Agustin

Fecha de entrega:
20 de Julio del 2024

Índice

Introducción a Git 2

Comandos Básicos 4

Trabajando con repositorios en github 6

Branches, Merge y Conflicts 7

Bibliografía 10

Introducción a Git

En la presente investigación se va a hablar un poco acerca de lo que es Git, algunos comandos básicos para utilizarlo a través de consola, como se trabaja con este en GitHub, y como realizar Branches, Merges, así como resolver conflictos derivados de estos.

Comenzaremos con hacernos la pregunta, ¿Que es git?, **git** es un sistema de control de versiones distribuido, diseñado para manejar proyectos de todo tamaño con rapidez y eficiencia. Fue creado por $Linus\ Torvalds$ [1] en 2005 para el desarrollo del kernel de Linux, pero ha sido adoptado ampliamente por diversos proyectos de software.

1. The Editors of Encyclopaedia Britannica. (2024). Linus Torvalds [Fotografía]. Fuente. https://acortar.link/7WGWzs

El diseño de Git se inspiró en BitKeeper y Monotone. Inicialmente, fue concebido como un motor de control de versiones de bajo nivel sobre el cual otros podrían desarrollar interfaces frontales, como Cogito o StGIT. Desde entonces, el núcleo de Git ha evolucionado hasta convertirse en un sistema de control de versiones completo y directamente utilizable.

Linus Torvalds buscaba un sistema distribuido similar a BitKeeper, pero ningún sistema de software libre disponible cumplía con sus requisitos, especialmente en términos de rendimiento. El diseño de Git permite gestionar una gran cantidad de código distribuido por muchas personas, lo que afecta varios aspectos de rendimiento y responde a la necesidad de rapidez en su implementación inicial.

Entre las características más destacadas de Git se encuentran:

- **Desarrollo no lineal eficiente:** Git ofrece un sólido soporte para el desarrollo no lineal, permitiendo una rápida gestión de ramas y la mezcla de diferentes versiones. Incluye herramientas específicas para navegar y visualizar un historial de desarrollo no lineal. Se asume que los cambios se fusionarán con más frecuencia de lo que se escriben originalmente, ya que pasan por varios programadores para su revisión.

- **Gestión distribuida:** Similar a sistemas como Darcs, BitKeeper, Mercurial, SVK, Bazaar y Monotone, Git proporciona a cada programador una copia local completa del historial del desarrollo. Los cambios se propagan entre los repositorios locales, importándose como ramas adicionales que pueden fusionarse de la misma manera que las ramas locales.

- **Publicación versátil de repositorios:** Los repositorios de Git pueden publicarse a través de HTTP, FTP, rsync o mediante un protocolo nativo, usando conexiones TCP/IP simples o cifrado SSH. Git también puede emular servidores CVS, lo que permite el uso de clientes CVS y módulos IDE preexistentes para acceder a repositorios Git.

- **Compatibilidad con Subversion y svk:** Los repositorios de Subversion y svk pueden utilizarse directamente con git-svn.

- **Gestión eficiente de proyectos grandes:** Git es muy rápido en la gestión de diferencias entre archivos, además de incluir mejoras de optimización para la velocidad de ejecución, lo que lo hace eficiente para proyectos de gran tamaño.

- **Autenticación criptográfica del historial:** Cada versión previa a un cambio está notificada por cambios posteriores, asegurando la autenticación criptográfica del historial, una característica presente también en Monotone.

- **Gestión de archivos renombrados:** Git maneja los renombrados basándose en similitudes entre archivos, en lugar de nombres únicos de nodos del sistema de archivos, evitando posibles conflictos desastrosos de nombres.

- **Reempaquetado periódico:** Git realiza un reempaquetado periódico en paquetes de archivos. Esto es eficiente para la escritura de cambios, aunque puede ser ineficiente para la lectura si el reempaquetado no se realiza regularmente.

- **Compatibilidad con *GitHub*** [2] **y Microsoft Visual Studio Code:** Git es compatible con estas populares plataformas de desarrollo, facilitando su integración en flujos de trabajo modernos.

2. Thiago Marsal Farias. (2023). GitHub [Logo]. Fuente. https://acortar.link/qR1UbB

Comandos Básicos.

Al comenzar a utilizar Git los primeros comandos que deberíamos utilizar serán los que nos permiten realizar la configuración inicial:

- **git config --global user.name “nombre usuario”**
- **git config --global user.email "tuemail@ejemplo.com"**

Después sería normal utilizar los comandos para iniciar un nuevo repositorio en la ruta en donde nos encontremos:

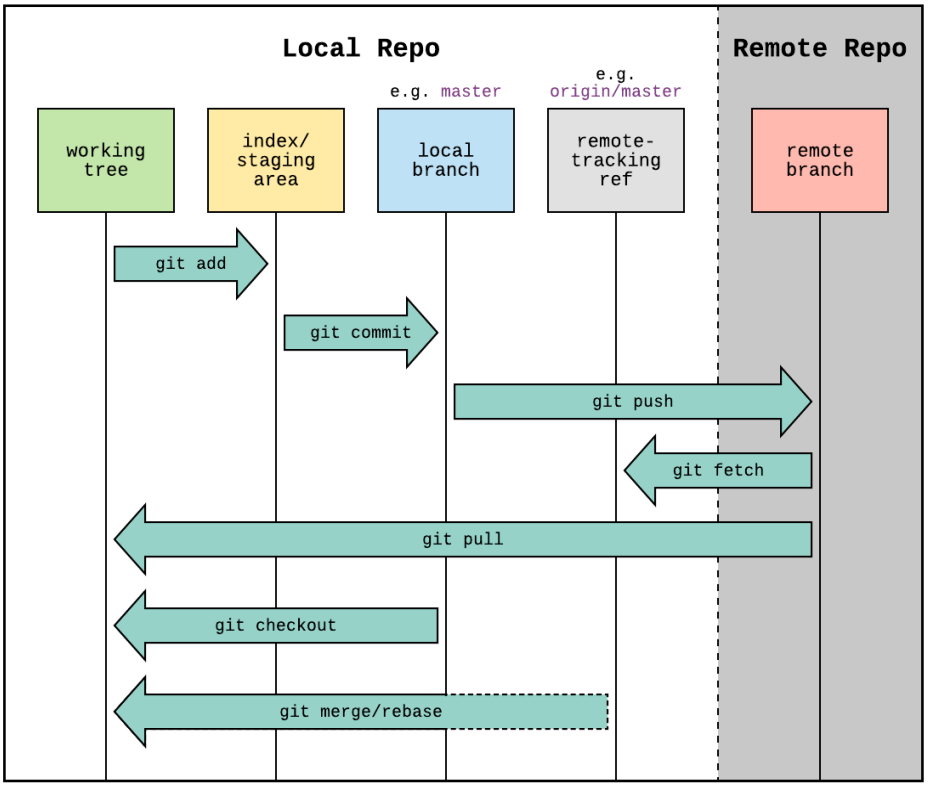
- **git init**

O descargar un repositorio de forma remota

- **git clone www.example.com**

Algunas operaciones básicas que se pueden hacer son:

- **git status**: Nos permite ver el estado del repositorio
- **git add ejemplo.txt**: Nos permite añadir archivos al área de preparación (staging)
- **git add .** Nos permite añadir todos los archivos modificados al área de preparación
- **git commit -m "Mensaje del commit"** Nos permite hacer un commit
- **git log**: Nos permite ver el historial de commits



3. MurkyCartographer525. (2018). Reddit [Gráfico]. Fuente. https://acortar.link/Ktkvxg

Para trabajar con ramas (branches) podemos utilizar:

- **git branch <nombre-de-la-rama>**: Nos permite crear una nueva rama
- **git checkout <nombre-de-la-rama>**: Nos permite cambiar a una rama
- **git checkout -b <nombre-de-la-rama>**: Nos permite crear y cambiar a una nueva rama
- **git merge <nombre-de-la-rama>**: Nos permite fusionar una rama con la rama actual

Para operaciones remotas se realizan principalmente con:

- **git remote -v**: Nos permite ver los repositorios remotos
- **git remote add <nombre> <URL>**: Nos permite añadir un repositorio remoto
- **git fetch**: Nos permite obtener cambios del repositorio remoto
- **git pull**: Nos permite obtener y fusionar cambios del repositorio remoto
- **git push**: Nos permite enviar cambios al repositorio remoto

Otras operaciones que podrían resultar útiles:

- **git diff**: Nos permite ver las diferencias entre archivos
- **git diff --staged**: Nos permite ver las diferencias entre el área de preparación y el último commit
- **git checkout -- <archivo>**: Nos permite revertir cambios en el área de trabajo
- **git reset --soft HEAD~1**: Nos permite revertir el último commit manteniendo los cambios en el área de preparación
- **git reset --hard HEAD~1**: Nos permite revertir el último commit descartando los cambios

4. Anonimo. (2020). Horizonte [Ilustracion]. Fuente. https://n9.cl/wjk75

Trabajando con repositorios en github

Existen diferentes repositorios remotos con los que podemos trabajar, que están abiertos de forma pública y “gratuita” y que nos facilitan mucho la integración de equipos a la hora de realizar un proyecto, ya que nos permiten poder subir y bajar el código de forma sencilla, uno de los más populares es **GitHub** de Microsoft, en el cual podemos seguir unos sencillos pasos para utilizar.

- Para comenzar deberemos tener una cuenta.

- Después al estar en la página de inicio tendremos que crear un repositorio, siguiendo los siguientes pasos:
    - Haz clic en el botón "+" en la esquina superior derecha y selecciona "New repository".
    - Rellena el formulario:
    - Agrega un nombre al repositorio.
    - Agrega una descripción (opcional).
    - Elige si el repositorio será público o privado.
    - Puedes inicializar el repositorio con un README (opcional).
    - Haz clic en "Create repository".

- Después la forma más sencilla de poder trabajar con él de forma local es clonándolo con el comando “git clone www.example.com” que se mostró en la sección anterior.

- Y después para subir cambios se deberán seguir los comandos add, commit y push, que se mostraron arriba y en ese orden en específico.
    - Se recomienda realizar un pull si es un repositorio colaborativo para evitar conflictos.

5. Sourav Kumar Nanda. (2018). medium [Gráfico]. Fuente. https://acortar.link/VxElkQ

Branches, Merge y Conflicts

Branches

Las ramas (branches) en Git son una característica fundamental que permite a los desarrolladores trabajar en diferentes versiones de un proyecto de manera simultánea. Una rama es esencialmente una línea de desarrollo independiente que parte de un punto específico en la historia del proyecto.

Una rama es un puntero a una confirmación (commit) específica en la historia del proyecto. La rama principal en la mayoría de los repositorios se llama main o master, y es la línea de desarrollo principal. Puedes crear ramas adicionales para desarrollar nuevas características, corregir errores o experimentar con ideas sin afectar la rama principal.

Ventajas de usar ramas

- **Desarrollo paralelo**: Permite a varios desarrolladores trabajar en diferentes características o correcciones de errores al mismo tiempo sin interferir unos con otros.
- **Historial limpio**: Facilita mantener un historial de commits claro y organizado.
- **Seguridad**: Permite probar y validar cambios antes de fusionarlos en la rama principal.
- **Flexibilidad**: Facilita la experimentación sin riesgo, ya que siempre puedes volver a la rama principal si algo sale mal.

Operaciones comunes con ramas

Crear una nueva rama:

- `git checkout -b nombre-de-la-rama`

Este comando crea una nueva rama y cambia a ella.

Cambiar de rama:

- `git checkout nombre-de-la-rama`

Este comando cambia a una rama existente.

Listar ramas:

- `git branch`

Este comando lista todas las ramas en el repositorio. La rama actual se marcará con un asterisco (*).

Fusionar una rama:

- `git checkout main`
- `git merge nombre-de-la-rama`

Este comando fusiona los cambios de `nombre-de-la-rama` en la rama `main`.

Borrar una rama:

- `git branch -d nombre-de-la-rama`

Este comando elimina una rama que ya no necesitas.

Flujo de trabajo típico con ramas

Crear una nueva rama para una nueva característica o corrección de errores:

- `git checkout -b nueva-caracteristica`

Trabajar en la nueva característica y hacer commits de los cambios:

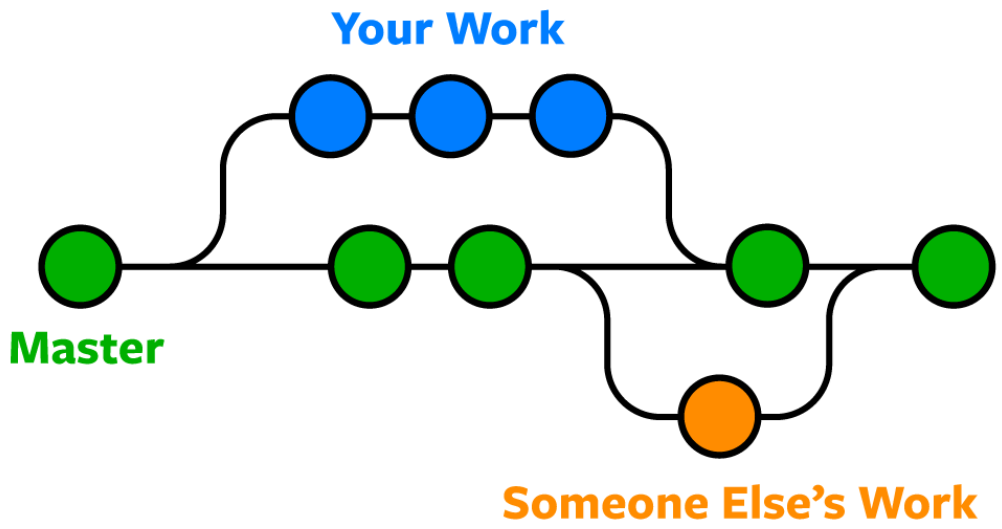
- `git add archivo-modificado`
- `git commit -m "Descripción del cambio"`

Cambiar a la rama principal y fusionar la nueva característica:

- `git checkout main`
- `git merge nueva-caracteristica`

Borrar la rama si ya no es necesaria:

- `git branch -d nueva-caracteristica`



6. Anonimo. (2021). nobledesktop [Gráfico]. Fuente. https://acortar.link/zxNAtQ

Pull Requests y ramas

En plataformas como GitHub, es común usar pull requests para gestionar la fusión de ramas. Un pull request permite revisar los cambios propuestos antes de fusionarlos en la rama principal. Esto es especialmente útil para proyectos con múltiples colaboradores.

Ejemplo práctico

Imagina que estás trabajando en un proyecto y quieres añadir una nueva funcionalidad sin interrumpir el desarrollo actual:

Crear una rama para la nueva funcionalidad:

- `git checkout -b nueva-funcionalidad`

Hacer cambios y commits en la nueva rama:

- `git add .`
- `git commit -m "Añadida nueva funcionalidad"`

Fusionar la nueva funcionalidad en la rama principal una vez que esté lista:

- `git checkout main`
- `git merge nueva-funcionalidad`

Borrar la rama si ya no es necesaria:

- `git branch -d nueva-funcionalidad`

Bibliografía

- Atlassian. (s. f.). Qué es Git | Atlassian Git Tutorial. https://www.atlassian.com/es/git/tutorials/what-is-git
- Mijacobs. (2023, 5 octubre). ¿Qué es Git? - Azure DevOps. Microsoft Learn. https://learn.microsoft.com/es-es/devops/develop/git/what-is-git
- Git - Fundamentos de Git. (s. f.). https://git-scm.com/book/es/v2/Inicio---Sobre-el-Control-de-Versiones-Fundamentos-de-Git
- Conceptos básicos de Git: Commits, Branches y Merges. (s. f.). desarrolladoraweb.com. https://desarrolladoraweb.com/blog/conceptos-basicos-de-git-commits-branches-y-merges
- Acerca de los conflictos de fusión - Documentación de GitHub. (s. f.). GitHub Docs. https://docs.github.com/es/pull-requests/collaborating-with-pull-requests/addressing-merge-conflicts/about-merge-conflicts