Introducción a GIT y Control de Versiones

Table of Contents

- What is version control and why should we care?
- GIT's graph model
- Git VS Github
- Terminology
- Commands
- Hands on tutorial!
  - How do we create a repository?
    - We can create a repository locally and then link it
    - We can create a remote repository and then download it
  - Forking a repo
  - Working with branches
    - Contribute to a forked repository
- Summary □□
- Further materials

¿Qué es el control de versiones y por qué debería importarnos?

¿Qué clase de magia negra es esta?

El control de versiones es un sistema que registra los cambios realizados en un archivo o conjunto de archivos a lo largo del tiempo, de modo que se puedan recuperar versiones específicas más adelante. Para los ejemplos en este cuaderno, utilizaremos el código fuente del software como los archivos que están bajo control de versiones, aunque en realidad puedes hacer esto con casi cualquier tipo de archivo en una computadora.

Si eres un diseñador web o gráfico y quieres mantener todas las versiones de una imagen o diseño (lo cual ciertamente querrás), un sistema de control de versiones (VCS) es algo muy inteligente de hacer. Te permite revertir archivos seleccionados a un estado anterior, revertir el proyecto entero a un estado anterior, comparar cambios a lo largo del tiempo, ver quién fue el último en modificar algo que podría estar causando un problema, quién introdujo un problema y cuándo, y mucho más. Usar un VCS también significa generalmente que si las cosas se desordenan o los archivos se pierden, se pueden recuperar fácilmente. Además, todo esto se logra con muy poco gasto.

GIT es un sistema VCS diseñado con los siguientes objetivos:

- velocidad
- Diseño simple
- Fuerte soporte para el desarrollo no lineal (miles de ramas paralelas)
- Completamente distribuido #copia completa
- Capaz de manejar proyectos grandes como el kernel de Linux de manera eficiente (velocidad y tamaño de datos)

Es un estándar absoluto de la industria

- **Nota:** Documentación principalmente tomada de docs

El modelo gráfico de GIT

Git modela la historia de tus proyectos como un grafo acíclico dirigido (un DAG). Entendamos estas tres palabras:

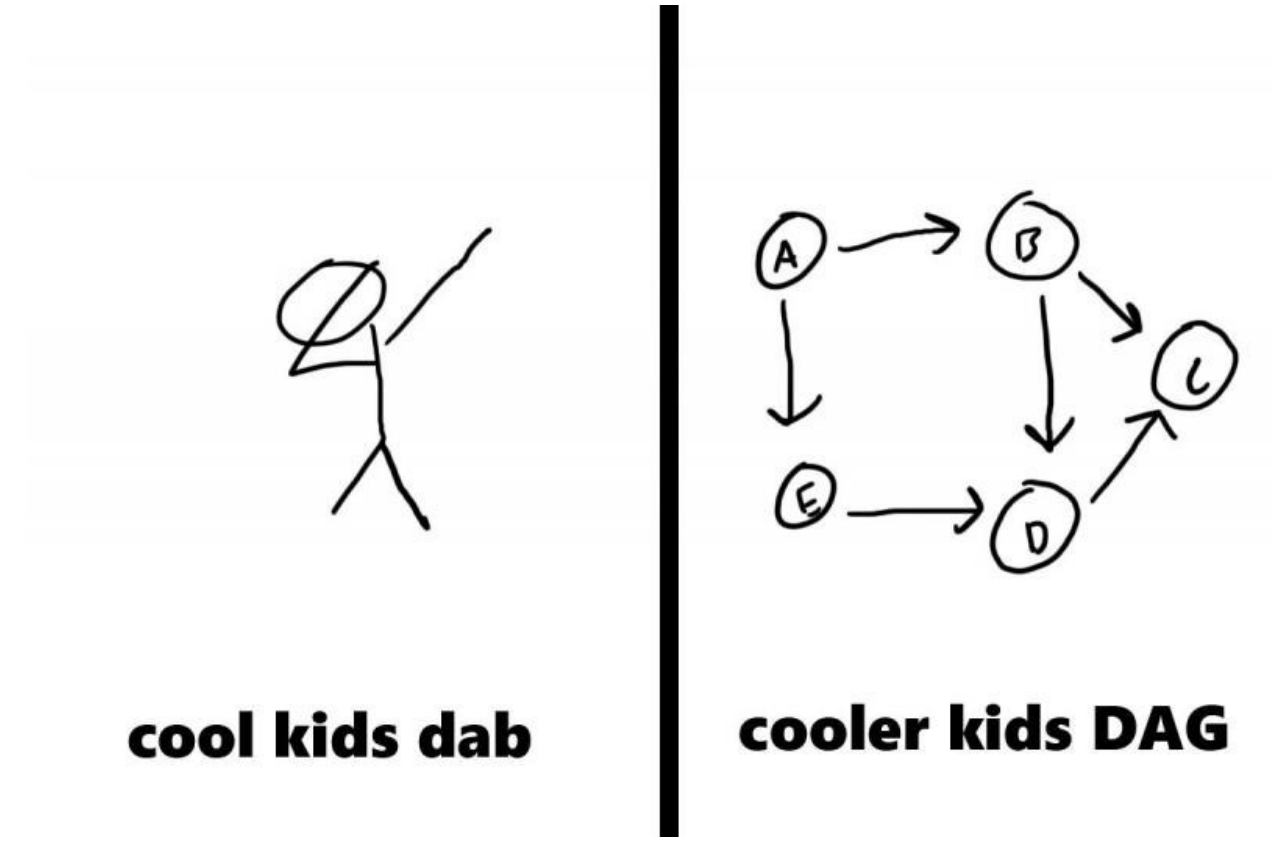
- Un **Grafo** es una forma de modelar cosas conectadas. Más técnicamente, un grafo es una colección de "nodos" conectados por "aristas". Piensa, por ejemplo, en una red social, en la que los individuos son los nodos y las relaciones son las "aristas" (también llamadas enlaces).
- **Acíclico** significa que el grafo no contiene círculos. Esto significa que no puedes encontrar un camino que comience y termine en el mismo nodo siguiendo la dirección de las aristas.
- **Dirigido** significa que las aristas solo pueden ir en una dirección. Piensa, por ejemplo, en "padre", "hijo", y la relación "es hijo de" --- no, no puedes ser padre e hijo de la misma persona.

Además:

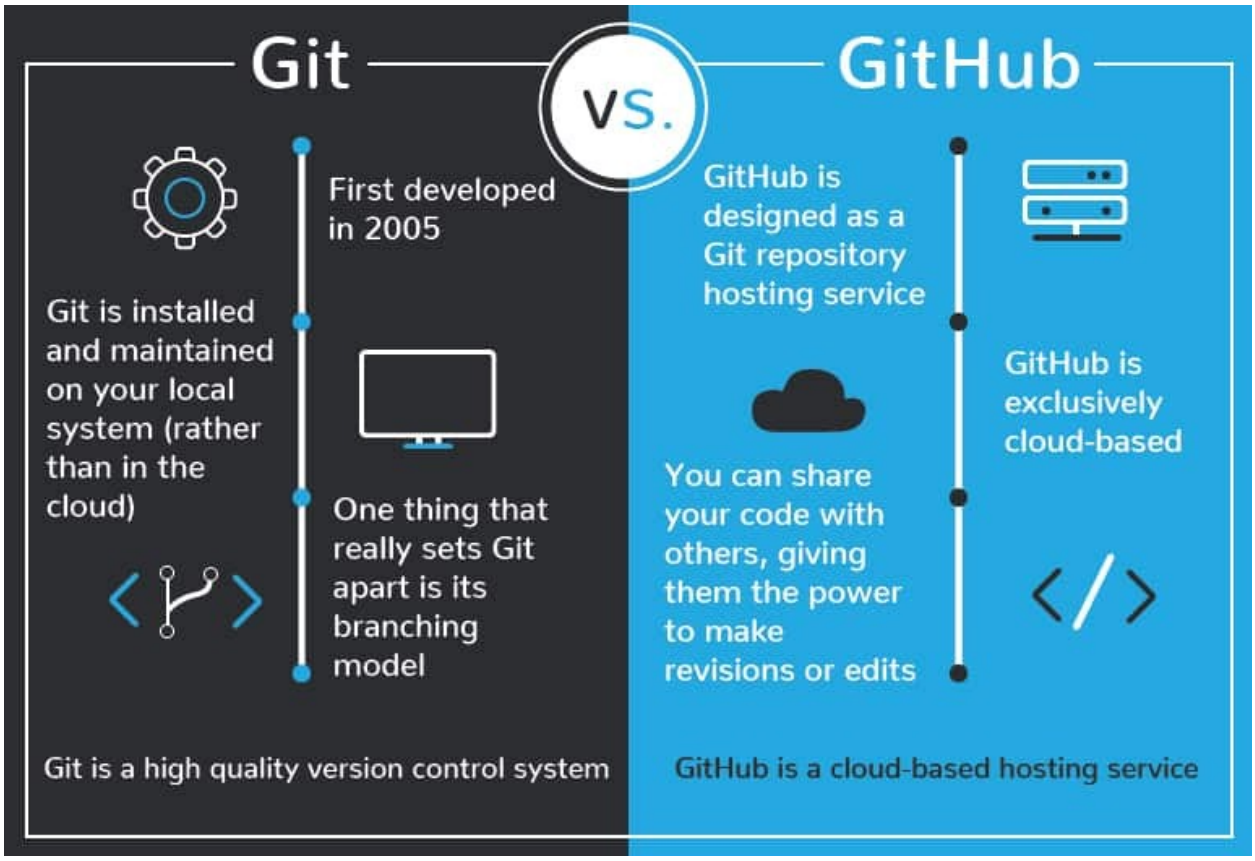
*El **gráfico en su totalidad** contiene una historia del proyecto.

- Los **Nodos** en Git representan commits (recuerda: instantáneas de tu proyecto)
- Las **Aristas** apuntan de un commit a sus padres.
- Una **rama** ocurre si un commit tiene más de un hijo.
- Una **fusión** ocurre cuando un commit tiene más de un padre.

¡Enlace a una visualización súper genial!



Git VS Github



- **Git** es un software de VCS local que permite a los desarrolladores guardar instantáneas de sus proyectos a lo largo del tiempo. Generalmente es mejor para uso individual.
- **GitHub** es una plataforma basada en la web que incorpora las características de control de versiones de git para que puedan ser utilizadas de manera colaborativa. También incluye características de gestión de proyectos y equipos, así como oportunidades para el networking y la codificación social.

Terminología

- `repositorio` | `repo` : ubicación de almacenamiento de un paquete de software ("una carpeta rastreada por git")
- `untracked` : un archivo o directorio no rastreado por git, no pertenece al repositorio, pero puede estar dentro de la carpeta.
- `tracked` : un archivo que git ha ordenado rastrear (a través del comando `add`.)
- `commit` : un punto en la línea de tiempo de un repositorio. Una "instantánea" de todos los cambios que hemos "oficializado", o el último estado de los archivos escaneados antes de hacer el commit. Cada commit tiene un identificador único y debe tener un mensaje de commit. Cuanto más descriptivo sea el mensaje, mejor.
- `local repo` : Un repositorio de código en la computadora que estás usando actualmente.
- `remote` : Un repositorio en una máquina diferente (por ejemplo: Github).
- `forking a repo` : crear una línea de tiempo alternativa para un repo, usualmente el repo de alguien más que queremos tener como una copia nuestra.

- `cloning a repo` : hacer una copia local de un repo remoto.
- `push` : Empujar cambios desde el repositorio local al remoto.
- `pull` : traer cambios de un repositorio remoto a uno local.
- `.gitignore` : archivo de texto con los archivos/carpetas que no queremos que git rastree o consulte.
  - p.ej.:

    logo.jpg `# git ignora archivo con nombre específico`

    develop_test/ `# git ignora el directorio especificado con nombre y todo su contenido. Nota el / después del nombre.`

    *.jpg `# git ignora todos los archivos con una extensión dada`
- `branch` : las diferentes `líneas de tiempo` de un repositorio. Permite el desarrollo paralelo.
- `merge` : Fusionar cambios de una rama a otra. Es lo opuesto a branching, en lugar de separar, une dos "líneas de tiempo".
- `pull request` : el acto de pedir al dueño de un repositorio que fusione cambios en su fork/rama. Es literalmente la solicitud de que `tires` tus cambios.

Comandos Git

- `git init` - Convierte una carpeta en un repositorio de Git.
- `git add <archivo>` - Agrega un archivo al área de preparación (staging area) del repositorio.
- `git restore --staged <archivo>` - Elimina un archivo del área de preparación. Lo "desetapa".
- `git commit -m "<mensaje>"` - Guarda los cambios con un mensaje descriptivo. Es como una "instantánea" en la línea de tiempo del proyecto.
  - `-a` - Realiza un commit de todos los cambios en archivos que ya han sido añadidos previamente al repositorio.
  - `-m` - Permite escribir el mensaje del commit directamente en la línea de comando.
  - `-am` - Combina `-a` y `-m`, agregando todos los cambios y permitiendo el mensaje en un solo comando.
- `git log` - Muestra el historial de commits del repositorio.
- `git reset --hard <commit_id>` - Restaura el repositorio al estado de un commit anterior de manera irreversible.
- `git remote add <nombre> <url>` - Conecta el repositorio local a uno remoto en la URL especificada, asignándole el nombre indicado.
- `git clone <url>` - Clona un repositorio remoto a tu máquina local.
- `git pull <remote> <branch>` - Trae los cambios del branch remoto especificado al branch actual (ejemplo: `git pull origin main`).
- `git push <remote> <branch>` - Envía los commits del repositorio local al branch remoto indicado (ejemplo: `git push origin main`).
- `git branch` - Lista todas las ramas del repositorio.

- `git branch <nombre>` - Crea una nueva rama con el nombre indicado.
- `git checkout <branch>` - Cambia a la rama especificada. Solo una rama puede estar activa a la vez.
- `git checkout -b <branch>` - Crea una nueva rama y cambia a ella en un solo paso.
- `git merge <branch>` - Fusiona la rama especificada con la rama activa actual.

:warning: **Git rebase**: Asegúrate de mantener una versión segura antes de realizar un rebase, ya que modifica la historia de commits.

¡Tutorial práctico!

Creemos nuestro propio repositorio y juguemos con él Siempre debemos tener presente esta maravilla... -->> **¡OH SHIT, GIT!** 🙃

¿Cómo creamos un repositorio?

Hay dos maneras:

Podemos crear un repositorio localmente y luego enlazarlo

```
cd somewhere # si necesitas
mkdir first_repo
cd first_repo
git init
```

Extra: ¡Vamos más allá! --> .git
comprobamos que .git se creó al hacer ls -a

```
ls -a
```

Para subir nuestro repositorio local a un repositorio en línea, primero debemos crear un repositorio en un sitio en línea como GitHub.

```
git remote add origin <url of repo>
```

Podemos crear un repositorio remoto y luego descargarlo

```
git clone <url of repo>
```

Más fácil y menos dañino, si?

Hacer un Fork de un repo

Hacer un fork de un repositorio significa copiarlo. Copiar un repositorio nos permite experimentar libremente sin afectar el proyecto original, obviamente porque nuestro fork es una copia, que se encontrará en nuestra cuenta de github.

Trabajando con ramas

Una rama en Git es simplemente una copia de cómo está el repositorio en el momento que la haces. El nombre de la rama por defecto en Git es main. Cuando comienzas a hacer commits, se te da una rama principal que apunta al último commit que hiciste. Cada vez que haces un commit, el puntero de la rama principal se mueve hacia adelante automáticamente. O la rama a la que haces commit.

- https://medium.com/faun/branching-with-git-the-multiverse-theory-83d3d1372746

Contribuir a un repositorio forkeado

Un pull request (PR) es una solicitud enviada a un repositorio de GitHub para fusionar código en ese proyecto. El PR permite a los revisores solicitados ver y discutir el código propuesto. Una vez que el PR pasa todos los estándares de revisión y se han hecho todas las revisiones necesarias, puede ser fusionado en la base de código. ¡Vamos!

Áreas de Git

Ejemplos de flujos de trabajo:

- Trabajando en **mi propio repo** que empecé localmente:
  - `git init` para iniciar el repo.
  - `git add <archivos>` para rastrear archivos.
  - `git commit -m "<mensaje>"` para hacer el commit de los cambios.
  - `git remote add <nombre> <url>` para conectar el repo a uno remoto (crear un repo vacío en github para evitar conflictos).
- Trabajando en un **remoto** al que tengo privilegios:
  - `git clone <url>` para hacer una copia loca
  - `git add <archivos>` y `git commit -m "<mensaje>"` para hacer el commit de los cambios.
  - `git pull` ANTES de hacer push. Queremos traer los intercambios extranjeros antes de enviar los nuestros para evitar conflictos.
  - `git push` para empujar nuestros cambios.
  - También es bueno hacer `git pull` antes de empezar a trabajar en cada sesión para asegurarnos de tener la última versión del código.
- Trabajando en un **remoto al que no tengo privilegios**:
  - `fork` en github para crear tu propia alternativa
  - `git clone` para copiar nuestro fork al local
  - `git commit`, `git add`, `git push` para que nuestros cambios se reflejen en nuestra copia remota
  - `pull request` para ver si el dueño acepta nuestros cambios.

Resumen □□

Ahora es tu turno: ¿Qué hemos aprendido?

Extra: práctica hands-on, learngitbranching

RECAP

Materiales adicionales

- Github para Dummies aquí
- Conoce a Linus Torvalds, la mente detrás de git Charla TED
- Consulta estos documentos de GIT
- Un poco más de información
- O esta guía simple
- Y no olvides tu Hoja de trucos