Introducción a GIT

Mateos Wenceslao

Clase 1

¿Cómo comparten código?

- Cuando hacen un TP que involucra programación, ¿Cómo trabajan en conjunto?
- ¿Qué problemas encuentran?

¿Qué problemas tienen?

- Se pisan el código.

- Usan versiones desactualizadas

Solución - GIT

GIT nace en el 2005 por la necesidad de mantener el desarrollo de kernel de Linux. Tomando como referencia un DVCS propietario llamado BitKeeper.

- Algunos de los objetivos del nuevo sistema fueron los siguientes:
  - Compatible con protocolos existentes (Git puede utilizarse con HTTP, SSH, FTP, etc)
  - Fuerte apoyo al desarrollo no lineal (miles de ramas paralelas)
  - Completamente distribuido
  - Capaz de manejar grandes proyectos (como el núcleo de Linux) de manera eficiente (velocidad y tamaño de los datos)

Instalación

Por repositorio

- apt install git

Desde source

- https://git-scm.com/downloads
- https://www.kernel.org/pub/software/scm/git

Configuración

Comando git config

- $ git config --global user.name "Mi Nombre" → Configura el nombre
- $ git config --global user.email Mimail@mail.com → Configura el mail.

Es importante configurar el nombre y el email, ya que cada nueva “versión” de los archivos que modifiquemos, contará con esta información.

Comando git config

- git config --global core.editor editorpordefecto → Configura el editor por defecto.
- git config -l → Visualiza la configuración

Comandos

Inicializando un repositorio en un directorio existente

- git init

Agregando Archivos

- git add *.c
- git add README

El comando “add” sirve para marcar los cambios que se van a incluir en el próximo commit. Luego de ejecutar este comando, y antes de hacer el commit, se dice que los cambios están en área de “Staging”.

Comandos

- git commit –m 'versión inicial del proyecto'

Guarda los cambios que están en el área “Staging” en el repositorio local.

A cada commit se le asocia mediante la opción “-m” un mensaje para describir los cambios que incluye.

Compartir

Para lo siguiente, es necesario crear una cuenta en algún servicio de Git en la nube, nosotros vamos a usar Github.

Luego:

- git remote add origin https://github.com:<user>/<repo>.git

“remote” se le denomina a un repositorio común que todos los usuarios utilizan como punto de sincronización. “add” es el comando para agregar un nuevo “remote” a nuestro repositorio local. “origin” es un argumento, y es el nombre de nuestro repositorio remoto. Puede ser cualquier otro nombre, aunque por convención se utiliza popularmente “origin”. Todo lo que sigue después del nombre del remoto, es la URL que contiene el repositorio. Puede tener distintos protocolos.

Compartir

- git push -u origin master

“push” es el comando para enviar todos los cambios que están en los commits locales que no se encuentren en el repositorio remoto. Se pueden enviar más de 1 commit a la vez. “-u” es una opción que sí está presente, determina que si no existe la rama en el remoto la cree. “origin” es el nombre del repositorio remoto tal cual lo configuramos en el paso anterior. “master” es el nombre de la rama que queremos actualizar en el repositorio remoto.

Compartir

Clonando un repositorio existente

- git clone [url] [directorio destino]

Por defecto el directorio destino es el nombre del proyecto.

Para obtener ese nuevo cambio, ejecutar:

- git pull

Busca y descarga los commits desde el remote configurado (ej. origin) y los fusiona con los archivos locales.

Tarea para el hogar

Realizar la clonación de un repositorio de algún compañero. El dueño del repositorio original, debe realizar algún cambio y volver a hacer commit.

Clase 2

Desarrollo en paralelo

- Supongamos que dos personas tienen que trabajar en dos características distintas que se van a superponer en algún aspecto.
- ¿Cómo desarrollamos sin pisarnos entre nosotros?

Ramas

Git permite desarrollar en “paralelo”, permitiendo distintas líneas de desarrollo dentro de un mismo repositorio.

- git checkout -b dev

El comando checkout sirve para cambiar de rama. La opción ‘-b’ es para crear una nueva rama, y lo que sigue después es el nombre de la misma. Realizar algún cambio, y luego hacer un commit.

- git checkout master

Ramas

La rama master se crea por defecto con el primer commit. Verificar si el cambio está hecho:

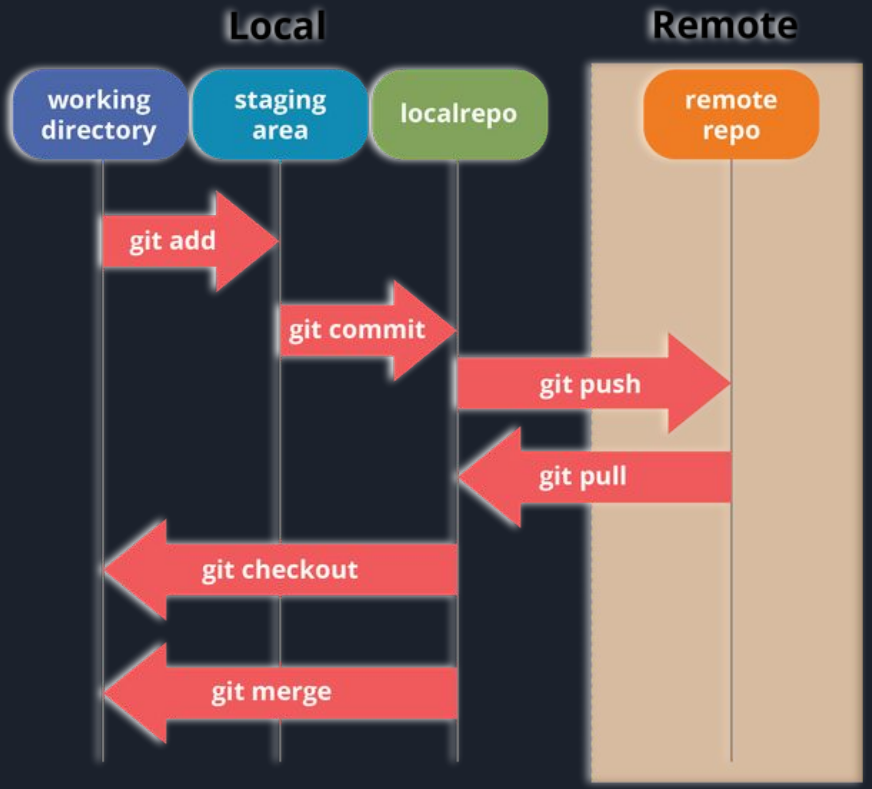
- “git branch” Lista todas las ramas en el repositorio local
- “git merge dev” Fusiona los cambios de las 2 ramas (la rama actual absorbe los cambios de la rama “dev” en este caso).

Ramas

A tener en cuenta:

- No puedo irme de una rama sin antes haber resuelto todos los cambios
- “git status” muestra el estado actual del árbol de trabajo.

Conceptos generales - 3 Árboles



Conceptos generales - Commit

Un commit es un conjunto de cambios de 1 o más archivos realizados desde el último commit y agrupados atómicamente.

A cada commit, Git lo identifica con un número hexadecimal de 40 caracteres. El número es un checksum que toma como argumento el conjunto de cambios en ese commit, incluyendo mensaje de commit, autor y checksum del commit anterior. Como todo checksum, cada vez que se inserta la misma entrada se obtiene la misma salida. Por esta razón, el número de commit es único, y además permite verificar la integridad del commit.

Conceptos generales - HEAD

Es una variable de referencia que apunta a un commit determinado (por defecto al último). El pointer HEAD puede ser movido a cualquier commit. Si se ubica en un commit previo, una vez en él, cada vez que se haga commit se pisarán los commits posteriores.

Tarea

En un mismo repositorio en github, hacer dos branches (distintos del master) generar cambios, subirlos a github y combinarlos en el master.

¿Qué problemas encontraron? ¿Cómo los solucionaron?

Clase 3

Problemas de la tarea

- ¿Qué problemas encontraron?
- ¿Cómo los resolvieron?

Deshacer

- git commit --amend

si te olvidaste algún archivo en la última confirmación, lo llevas al staging area y haces el ammend

- git reset HEAD <archivo>

quita el archivo del staging area.

Deshacer

- git checkout -- <file>

Deshaces un archivo modificado. No usar a menos que estén totalmente seguros.

- git reset <commit>

Vuelve a un commit previo, desde donde luego cada commit pisará los que le seguían a ese commit.

- git revert <commit>

Crea nuevos commit que deshacen los cambios entre el commit actual y el pasado en el argumento

Push (avanzado)

Ver los distintos múltiple sitios remoto:

- git remote -v

Traer y Combinar Remotos:

- git fetch [remote-name]

Trae ramas y tags de un repositorio remoto.

Enviar los cambios:

- git push [nombre-remoto] [nombre-rama]
- git push origin master → Envía todos los cambios que se han hecho commit.

Pull y push

Pull y push son dos comandos que juntan una serie de comportamientos. Cuando hacemos un fetch, git trae los cambios remotos a nuestros branches remotos, pero eso no altera nuestros branches locales, por lo que tenemos que también hacer un merge.

Lo mismo sucede con push, es en realidad un merge de nuestros branches locales con los remotos y luego hacer un fetch.

Tags

Las etiquetas en Git son hitos dentro del proyecto que deseamos marcar para luego encontrar fácilmente. Suelen estar compuestos por varios commits.

Git utiliza dos tipos principales de etiquetas:

- Ligera: es muy parecido a una rama que no cambia - simplemente es un puntero a un commit específico.
- Anotadas: se guardan en la base de datos de Git como objetos enteros. Tienen un checksum; contienen el nombre del etiquetador, correo electrónico y fecha; tienen un mensaje asociado; y pueden ser firmadas y verificadas con (GPG). Normalmente se recomienda que crees etiquetas anotadas, de manera que tengas toda esta información; pero si quieres una etiqueta temporal o por alguna razón no estás interesado en esa información, entonces puedes usar las etiquetas ligeras.

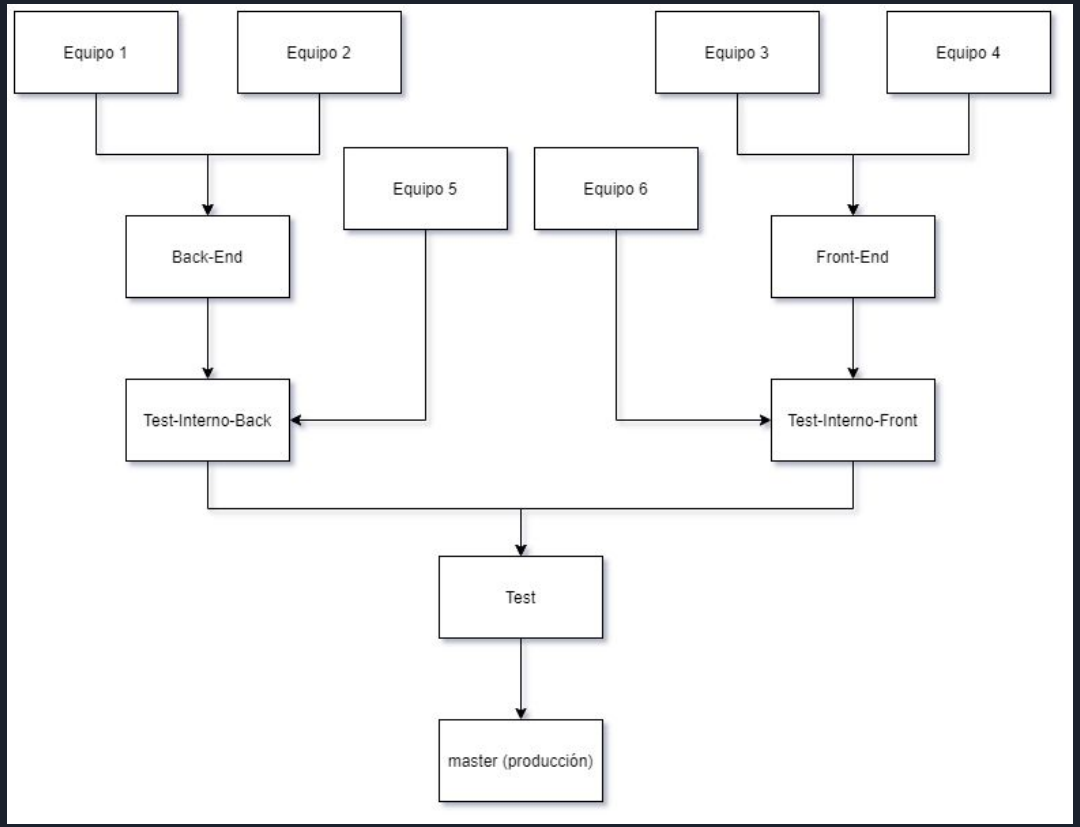
Tags

- git tag → VIsualización
- git tag <nombre> → etiqueta ligera.
- git tag -a v1.4 -m 'my version 1.4' → etiqueta anotadas.

Git push no transfiere la etiqueta hay que hacerlo en forma explícita.

- git push origin v1.5
- git push origin –tags → Enviar todas las etiquetas creadas.

Workflow - Provisorio



Conflictos

Un conflicto ocurre cuando hay modificaciones dispares en una misma porción de un mismo archivo en las dos ramas distintas que se pretenden fusionar.

Para identificar un conflicto, ejecutar:

- git status

Lo que hay entre <<<<<<< HEAD y ======= es el contenido donde apunta el HEAD. Lo que está entre ======= y

>>>>>>> es el contenido nuevo que queremos fusionar.

En el archivo que indique el conflicto se verá algo como lo siguiente:

<<<<<<< HEAD:index.html

contact : email.support@github.com

=======

  please contact support@github.com

>>>>>>> iss53:index.html

Conflictos

Para resolver el archivo, editarlo, guardarlo y luego volver a fusionar.

- git diff

El diff es un comando útil para prevenir conflictos. Muestra la diferencia entre lo que tenemos localmente sin estar en “staging”.

Pull Request

Si bien Git permite que colaboradores compartan código de diversas formas, está popularmente aceptado como buena práctica el hecho de utilizar “Pull Requests” o PR.

Un PR es una petición que hace un usuario a otro para que revise y fusione modificaciones sobre un proyecto. Supongamos que el usuario Juan está encargado de mantener el proyecto A, y para ello tiene un repositorio Git.

Si bien hay formas de realizar un PR desde línea de comando con la orden request-pull, la mayoría de los servicios en la nube de Git ofrecen una forma más amigable de revisar código, comentarlo y aceptar o rechazar.

Pull Request

Pedro quiere colaborar con el proyecto A.

El primer paso para Pedro es clonar el repositorio del proyecto en un repositorio local. Luego, realizará las modificaciones en su repositorio local. Una vez que considere que sus modificaciones son un aporte significativo al proyecto, enviará un PR al repositorio desde originalmente clonó.

Juan podrá entonces revisar las modificaciones y decidir si las acepta o las rechaza. Si las acepta, fusionará el código enviado por Pedro en el proyecto. Si las rechaza le dará a Pedro una explicación del por qué la rechazó y Pedro así podrá seguir trabajando para hacer efectiva su colaboración en un futuro cercano.

Tarea Final

En los grupos designados un integrante debe crear un repositorio en github con dos branches asociadas a los integrantes.

Crear una página web (con HTML, CSS y JavaScript) que sirva como tutorial de GIT. En la misma deben haber videos, imágenes e información.