# GIT - GITHUB

Lorena Baigorria
Fabrizio Riera
Brian Paez

DÍA 2

# Ejercicio Práctico

Cada alumno de manera individual debe:

1. Crear un proyecto en lenguaje C o Java llamado “calculadoraConGit”. La calculadora se compondrá de las funciones/métodos “suma” y “resta”. Crear al menos 3 versiones en las cuales se puedan dividir las funcionalidades en:
   a. Saludo de la calculadora básica con un pequeño menú de opciones.
   b. Crear la función/método suma y añadir al menú de la calculadora.
   c. Crear la función/método resta y añadir al menú de la calculadora.

# Volviendo en el Tiempo en Git - Git Reset y Git Revert

Para quienes se inician en el manejo de versiones con Git, comandos como git reset y git revert se vuelven herramientas indispensables, ya que permiten deshacer errores y ajustar el historial de cambios sin complicaciones. ¿Cuál es la diferencia entre Git Reset y Git Revert?

• **Git Reset**: mueve el puntero de los commits a uno anterior, permitiendo “volver en el tiempo” y explorar el historial de cambios. Es útil para deshacer actualizaciones recientes o revisar lo que se hizo en cada commit.

• **Git Revert**: crea un nuevo commit que revierte los cambios de un commit específico, permitiendo conservar el historial original sin eliminaciones. Es ideal para regresar a un estado anterior sin afectar los commits de otros usuarios.

# Gestión de versiones: tag y checkout

Los comandos **git tag** y **git checkout** son piezas clave para una gestión eficiente y ordenada de los cambios en el código.

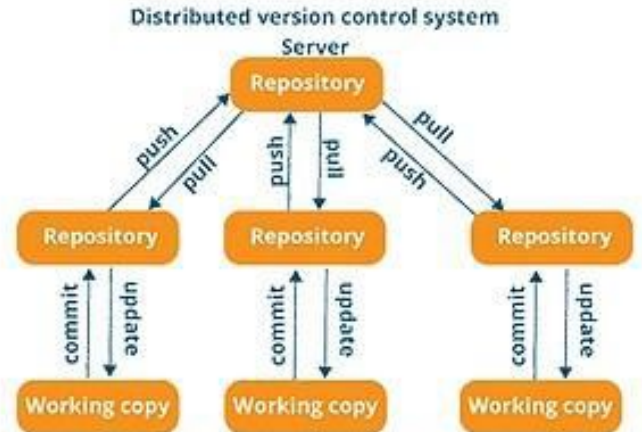
- El comando **git tag** permite marcar un commit con una etiqueta descriptiva, ideal para señalar versiones estables o hitos importantes en el proyecto.
- El comando **git checkout** permite revisar commits previos para explorar o probar cambios sin alterar la rama principal. Puedes regresar a un punto específico en el historial y evaluar cómo afectaban los cambios al proyecto en ese momento. Tiene usos más amplios además, cómo cambiar entre ramas que lo veremos la próxima clase.

# GitHub

Hay dos tipos de sistemas de control de versiones:

Sistemas de control de versiones centralizados.

**Sistemas de control de versiones distribuidos.**

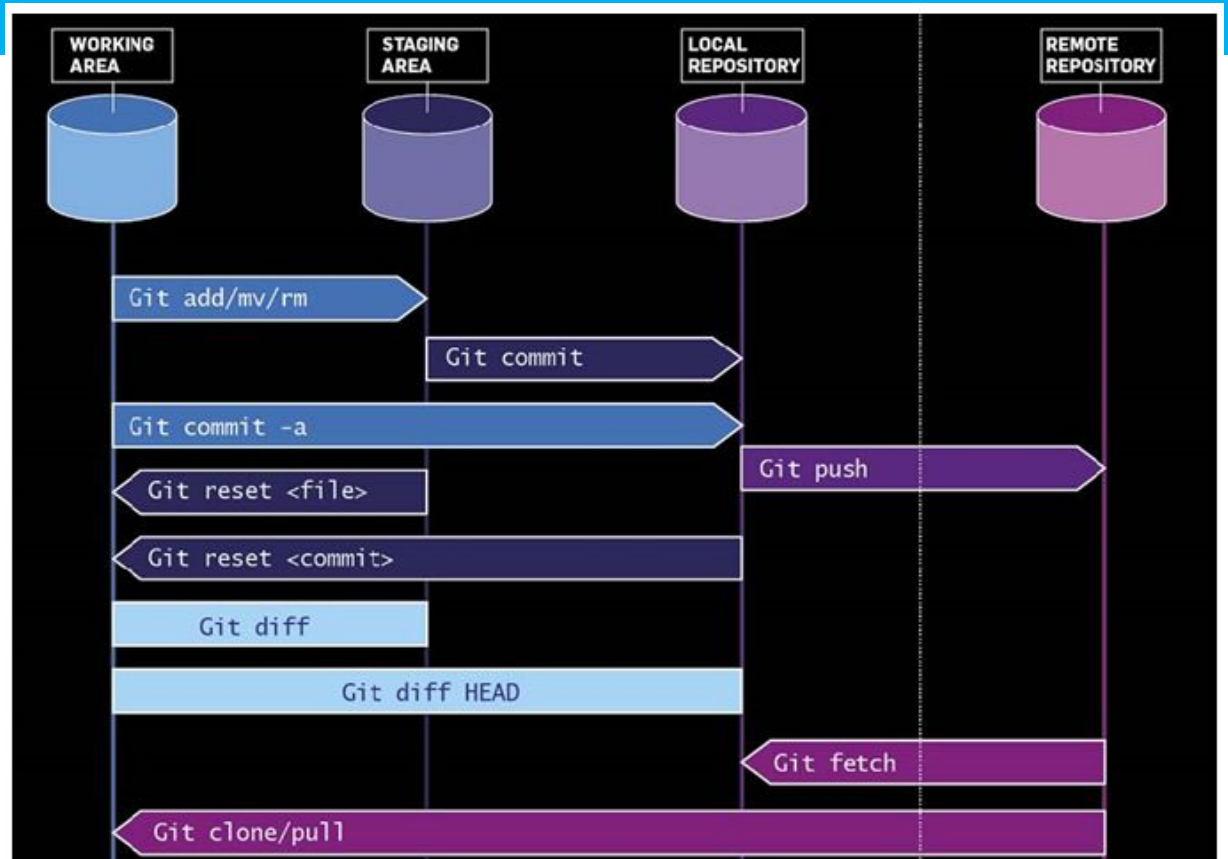



# GitHub  Sistema de Control de Versiones Distribuido

Los sistemas de control de versiones distribuidos (SCVD) no dependen necesariamente de un servidor central para almacenar las versiones de los archivos del proyecto.

- Cada programador tiene una copia local o clon del repositorio principal
- Todos pueden operar con su repositorio local sin ninguna interferencia.
- Todos los programadores pueden:  actualizar sus repositorios locales con nuevos datos del servidor central con una operación llamada ‘pull‘ y  persistir cambios en el repositorio principal con una operación llamada ‘push‘ desde su repositorio local.

# Git - GitHub Áreas y Estados



# Github

- GitHub es una plataforma de colaboración formal e informal de desarrollo de software (conocida también como plataforma de social coding). en esta se pueden publicar repositorios remotos que funcionan bajo el sistema de control de versiones Git. La plataforma configura los proyectos nuevos como de código abierto, por lo que cualquier persona puede verlos, pero esto es configurable,

- **Incidencias** Una incidencia o asunto (problema) es una unidad de trabajo designada para realizar alguna mejora en un sistema informático. Puede ser el arreglo de un fallo, una característica pedida, una tarea, una solicitud de documentación específica y todo tipo de ideas o sugerencias al equipo de desarrollo.

# Github - Intro

**Hitos** Los hitos (Milestones) son grupos de incidencias y que ayudan a seguir el progreso de estas. Desde la página de detalle de un hito se pueden observar los siguientes datos:

• Una descripción del hito proporcionado por el usuario, que puede incluir información como una descripción general del proyecto, equipos relevantes y fechas de vencimiento proyectadas.

• La fecha de vencimiento del hito

• El porcentaje de finalización del hito

• La cantidad de incidencias abiertas y cerradas asociadas con el hito. • Una lista de incidencias abiertas y cerradas asociadas con el hito.

# Github - Proyectos

**Etiquetas** Cada incidencia puede ser etiquetada bajo categoría (error, documentación, duplicado, inválido, etc) de manera que estas etiquetas pueden identificar rápidamente a qué grupo o tipo de tarea corresponde cada incidencia. Además del conjunto por defecto de etiquetas que tiene GitHub, se pueden crear otras que sean específicas de la organización o del proyecto.

Los proyectos en GitHub permiten organizar incidencias y notas en categorías mediante tarjetas en columnas. Estas tarjetas se pueden arrastrar entre las columnas según los estados en los que se encuentran las tareas que representan.

En GitHub existen tres tipos de tableros de proyectos:

- Pertenecientes al usuario: pueden tener incidencias de cualquier repositorio personal.
- De proyectos a nivel de organización: pueden contener incidencias de cualquier repositorio que pertenezca a una organización.
- Tableros por repositorio: están enfocados en las incidencias de un único repositorio. Pueden incluir referencias o notas a incidencias de otros repositorios.

# Github ¿Cómo y para qué se utiliza?

Creación de usuario

Creación de repositorio

Acceder a repositorio

Compartir un repositorio

Desplegar un proyecto

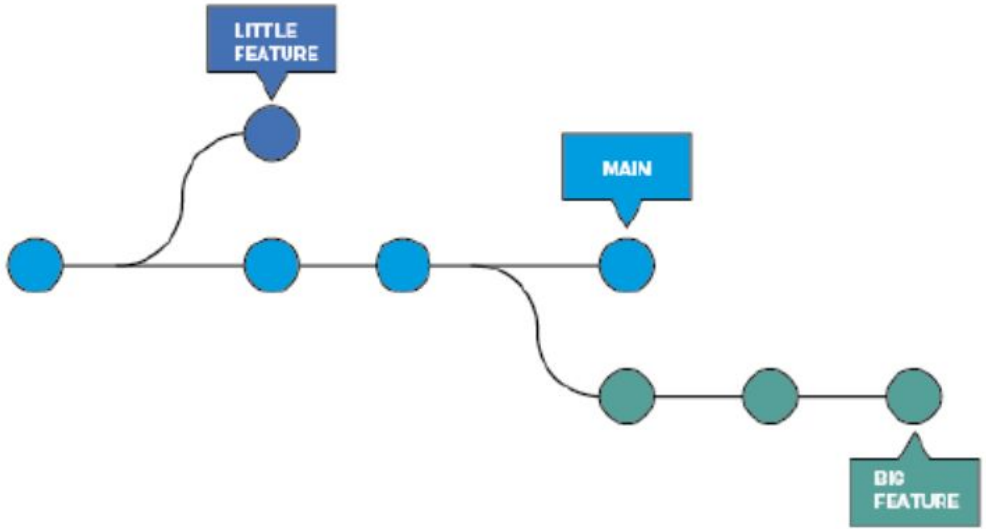
**https://github.com/**

# Github - Flujos de Trabajo

**Ramas:** En Git, las ramas son parte del proceso de desarrollo diario. Las ramas de Git son un puntero eficaz para las instantáneas de tus cambios.

**comando git branch**

**Una rama representa una línea independiente de desarrollo. Las ramas sirven como una abstracción de los procesos de cambio, preparación y confirmación**



# GitHub - Ejercicio Ramas

¿Hacemos una prueba? Iniciamos haciendo una rama en forma local
Para crear una nueva rama y saltar a ella, en un solo paso, puedes utilizar el comando `git checkout` con la opción `-b`:

$ git checkout -b nombrerama

Para cambiar de rama

$ git checkout nombre rama

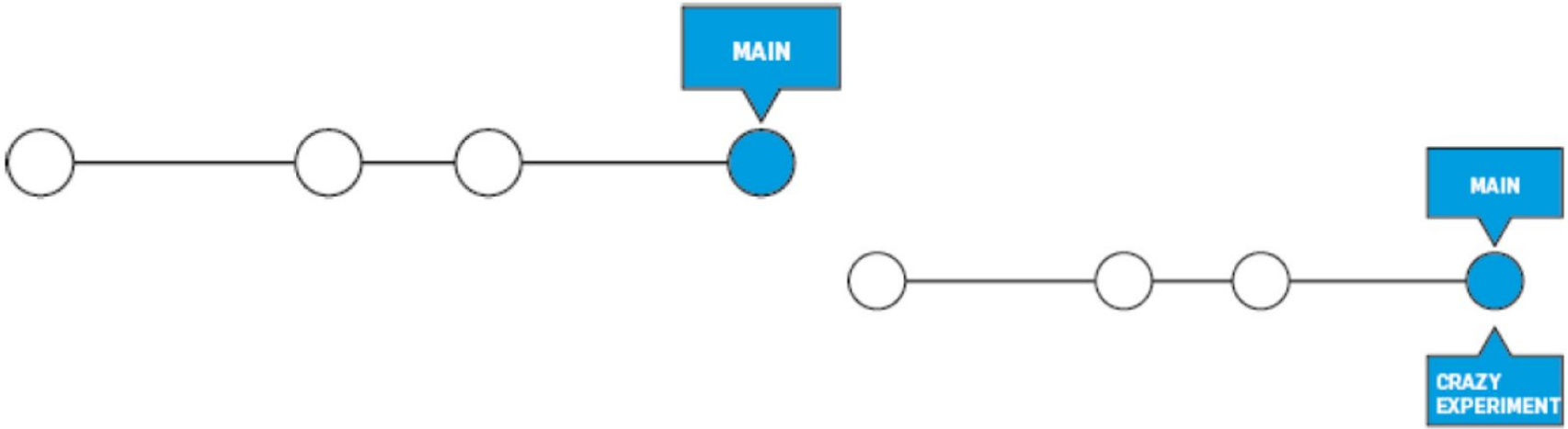
$ git switch nombrerama

Para ir a la rama main: $ git checkout -

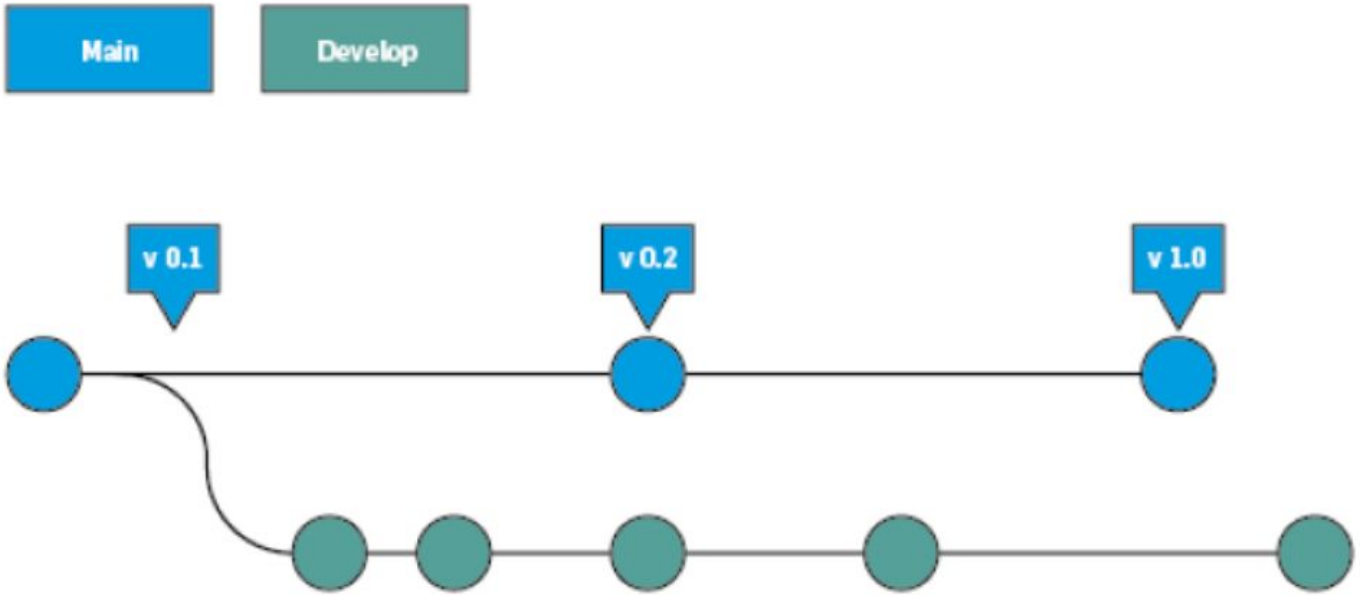
Para unir las ramas me paro en main y  $ git merge nombrerama

# Github - Ramas

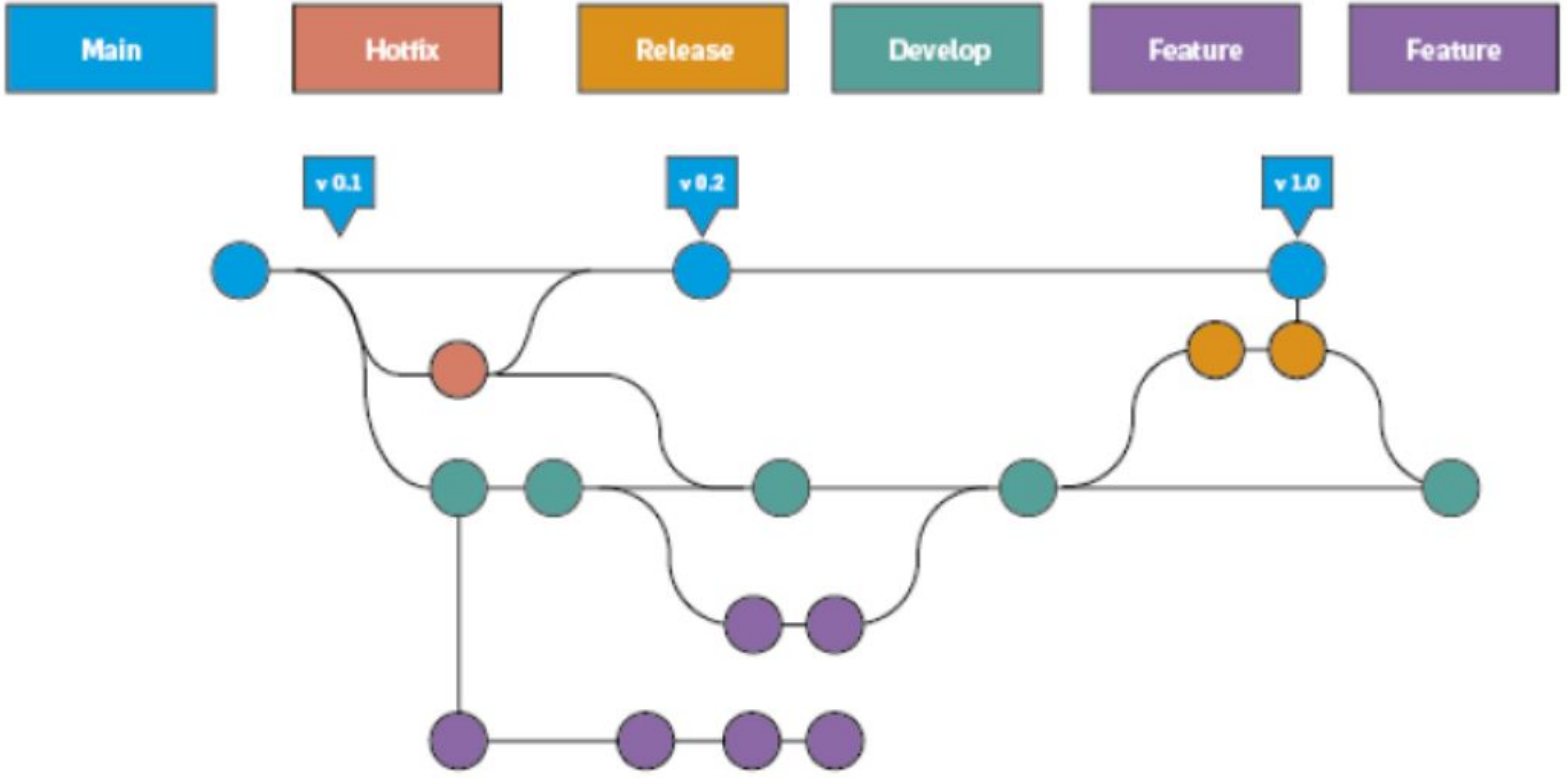
Es importante comprender que las ramas son solo punteros a las confirmaciones. Cuando se crea una rama, todo lo que Git tiene que hacer es crear un nuevo puntero, no modifica el repositorio de ninguna otra forma. Si empiezas con un repositorio que tiene esta forma



# Github - Ramas



# Github - Ramas



# GitHub - ejercicio

- Será necesaria la demostración para generar la creación del primer proyecto de Github a través de https://github.com/ y deberá contener lo siguiente:
    - Clone a repositorio local
    - Commit de cambios en el archivo readme como mínimo
    - Push de los cambios al servidor de github
    - Pull para confirmar los cambios en el repositorio local

Material Adicional:
https://www.atlassian.com/es/git/tutorials/comparing-workflows/gitflow-workflow
https://www.makeareadme.com/
https://bluuweb.github.io/tutorial-github/01-fundamentos/#ramas-o-branch

# GitHub - Trabajo en grupo

En un ejemplo de C o Java realizar un proyecto dividido en modulos

Distribuir módulos a cada integrante del grupo

Subir y Compartir el proyecto con el grupo

Bajar el proyecto

Hacer la tarea asignada

Subir el proyecto