

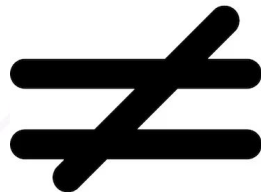
Git y GitHub

DEV.F
DESARROLLAMOS(PERSONAS);

dev



git



Git y github no son lo mismo, cada que piensas eso muere un gatito.

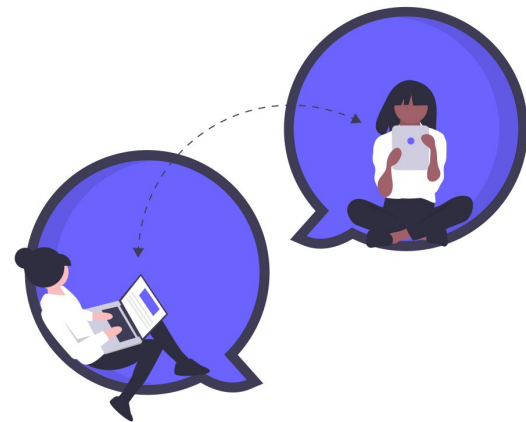
The background of the slide is a dark blue color with a subtle, light blue circuit board pattern. The pattern consists of various geometric shapes, including lines, circles, and polygons, arranged in a way that suggests a complex electronic circuit. The text is centered on the slide in a white, sans-serif font.

Git

Control de versiones

¿Qué podemos hacer con git?

- **Respaldar** nuestro código de forma profesional.
- Trabajar **colaborativamente**.
- Controlar **historial** de cambios de código.
- **Saber que, cuando y quien modificó** el código.
- **Controlar** las **versiones** que se van a liberar y las que están en desarrollo.
- **Recuperar** cambios **perdidos**.
- Seguimiento y avances trackeables.
- Integración de cambios controlable y semiautomática.



¿Qué podemos hacer con git?

```
commit 2c9d512d7896b7865d544fc3dce23aeb7ec8ad83 (HEAD -> develop)
```

```
Merge: 72ff934 df5cd35
```

```
Author: montoyaguzman <montoyaguzman7@gmail.com>
```

```
Date: Mon Aug 29 19:39:42 2022 -0500
```

```
Merge branch 'develop' of github.com:montoyaguzman/js-avanzado-g17B into develop
```

```
commit df5cd3578928f12753104c6efb9eb8d57fd029f5 (origin/develop)
```

```
Author: danielgloria <daniel.gloria@gmail.com>
```

```
Date: Thu Aug 25 22:40:38 2022 -0500
```

```
feat: Conection to database fixed
```

```
commit c29359424ad448149f1c3646b1c356491510dbf8
```

```
Author: danielgloria <daniel.gloria@gmail.com>
```

```
Date: Thu Aug 25 22:34:08 2022 -0500
```

```
feat: updateProduct query fixed
```

```
commit e7fd538db5db19562377a08c72da68610c08bcd3
```

```
Author: danielgloria <daniel.gloria@gmail.com>
```

```
Date: Thu Aug 25 22:29:51 2022 -0500
```

```
feat: reordenamiento de carpetas
```



Github

Repositorio remoto

Repositorio remoto

Github funge como **la plataforma en internet** a donde subimos el código para **compartirlo con otros** desarrolladores. Es un **repositorio remoto** para **compartir el código** que tenemos en git.



GitHub

montoyaguzman

montoyaguzman

Find a repository...Type to search

OverviewRepositories78ProjectsPackagesStars2

montoyita

montoyaguzman

Viviendo una linea de código a la vez

Edit profile

192 followers · 209 following

BBVA

Change your avatar

Find a repository...

Type

Language

Sort

New

g26a-intro-javascriptPublic

Repo de intro a js de la g26a de dev.f

☆ 1 Updated last month

g25c-javascriptPublic

Kata de Javascript para la generacion 25

JavaScript ☆ 5 2 Updated on May 28

g24b-computer-sciencePublic

JavaScript ☆ 3 1 Updated on May 28

ecmascript6-13Public

JavaScript ☆ 1 Updated on Apr 29

☆ Star

☆ Star

☆ Star

☆ Star

DEV.F.

***¿Por qué necesito un
controlador de versiones?***



dev

¿Qué significa controlar una versión?

- segunda revision >
- tesis-corregida >
- tesis-corregida copy >
- tesis-revisada >
- tesis-v-finañ >
- tesis1 >



Sistemas de Control de Versiones

Sistema de control de versiones (VCS)

El **Version Control System** es aquel que nos permite llevar un historial y control de cambios a medida que una o más personas colaboran en un proyecto.

¿Que cambios se hicieron?

¿Quién hizo los cambios?

¿Cuando se hicieron los cambios?

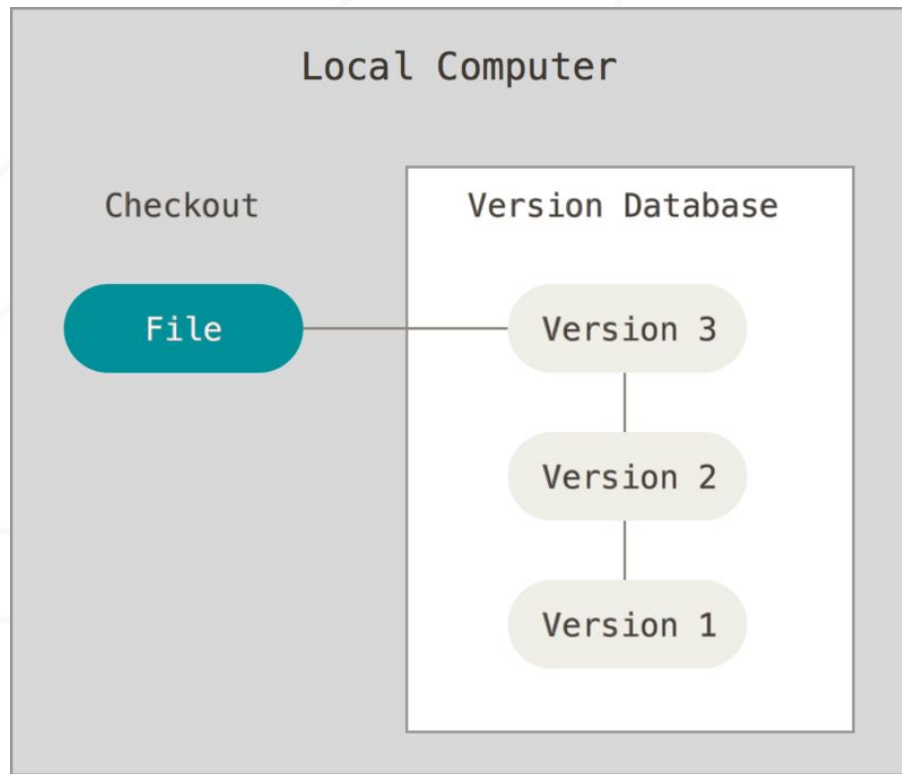
¿Por qué fueron requeridos los cambios?

Tipos de controladores de versiones

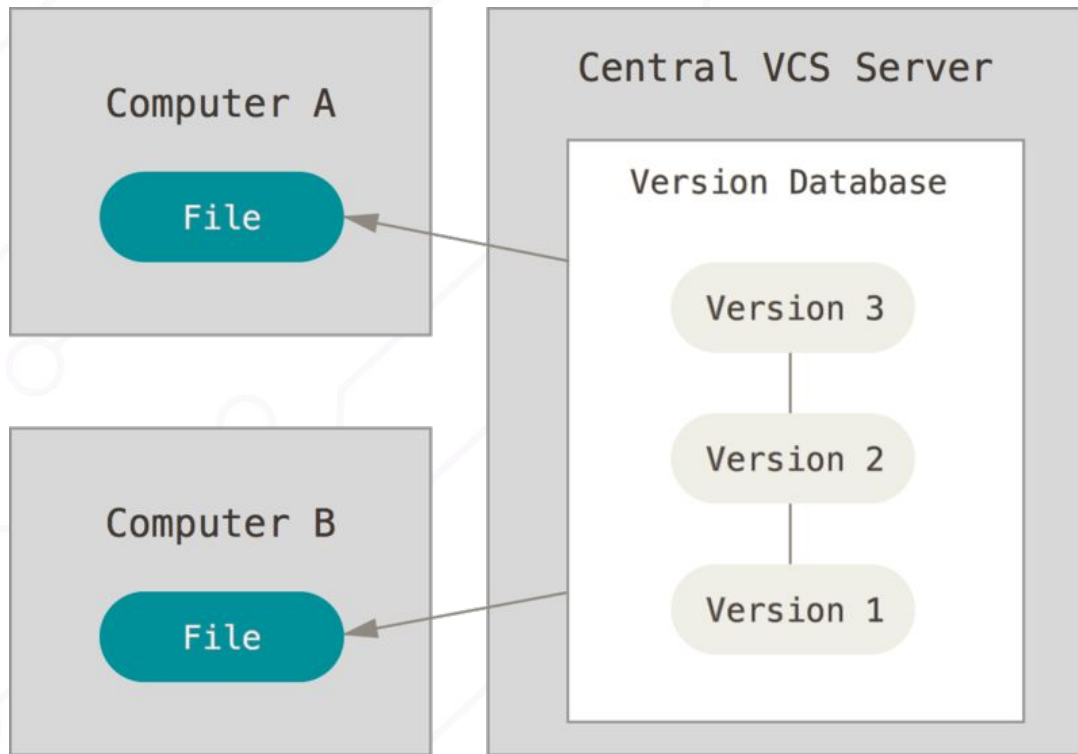
Tipos de controladores de versiones

- **Locales:** Todos los datos del proyecto se almacenan en una sola computadora y los cambios realizados en los archivos del proyecto se almacenan como revisiones.
- **Centralizados:** Utiliza un servidor central para almacenar todos los archivos y permite el trabajo colaborativo de un equipo. Trabaja sobre un repositorio único al que los usuarios pueden acceder desde un servidor central.
- **Distribuidos:** Aparecen para superar el inconveniente del sistema de control de versiones centralizado. Los clientes clonan completamente el repositorio, incluido su historial completo. Si algún servidor está inactivo o desaparece, cualquiera de los repositorios del cliente se puede copiar en el servidor para restaurarlo.

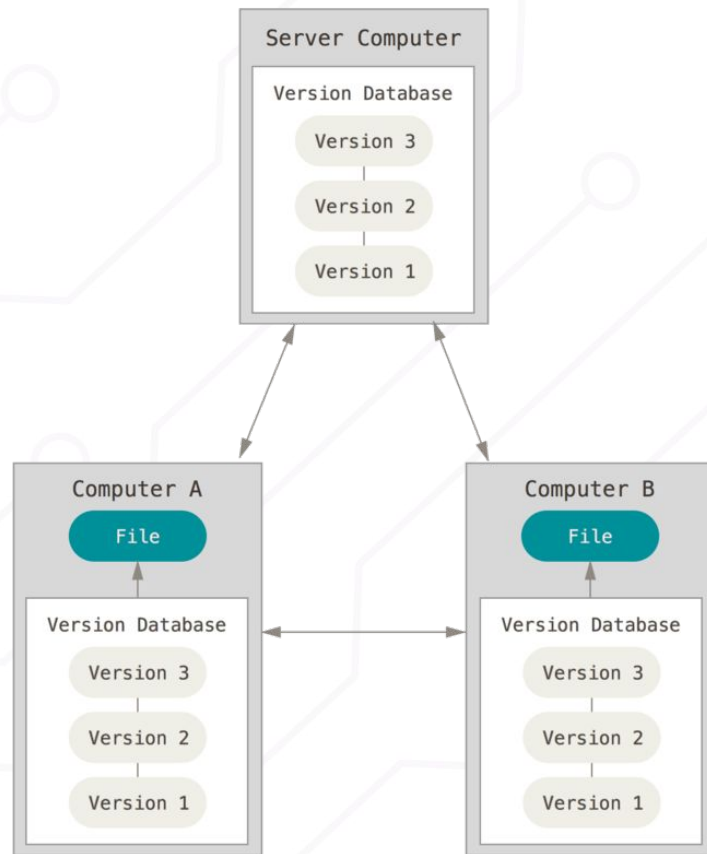
Local



Centralizado



Distribuido



A large, dark blue diamond shape with a white border, containing the text 'DEV.FL' in white. The 'F' is stylized with a grid pattern.

DEV.FL

Git

dev



Git es un (VCS) de tipo **distribuido** de código abierto y actualmente el más usado por los desarrolladores gracias

a sus beneficios para individuos y equipos de trabajo.

- Acceso detallado a la historia del proyecto.
- Colaboración en cualquier momento y lugar.

Su uso principal es mediante Interfaz de línea de comandos (**CLI - Command line interface**).

¿Qué es un repositorio?



Un **repositorio** es un espacio de almacenamiento donde se organiza, mantiene y difunde información.

El **repositorio** es la carpeta del proyecto donde estará la colección de archivos y carpetas junto al historial de cambios.



Instalación y manejo de Git

DEV.F
DESARROLLAMOS(PERSONAS);

dev

Instalación

- [Windows.](#)
- [Mac OS.](#)
- [Linux.](#)

Configuración de GIT

DEV.F
DESARROLLAMOS(PERSONAS);

dev

Configuración inicial de Git

- Desde consola, se puede ver a la configuración de **Git** con el comando:

```
git config --list
```

- Después de instalar git, lo primero que debe hacer es establecer una **identidad** en **Git**, para ello se usan los comandos:

```
git config user.name "mi nombre"
```

```
git config user.email miemail@algo.com
```

- Usando el flag "**--global**" podemos establecer la configuración de forma global y realizarla una sola vez. Sin el, se hace solo a nivel de carpeta.



Comandos de configuración inicial

Posterior a la instalación es necesario indicar a git el usuario y correo que se utilizara para registrar cada uno de los commits realizados en el equipo.

Configurar el nombre de usuario

```
git config --global user.name montoyaguzman7
```

Configurar el correo global.

```
git config --global user.email montoyaguzman7@gmail.com
```

NOTA: Esta configuración se realiza solo una vez por equipo y/o carpeta.
Este usuario y nombre es el que se verá en cada commit que hagamos.

Conceptos

DEV.F
DESARROLLAMOS(PERSONAS);

dev

Conceptos clave

- **Repositorio local:** Es la carpeta que inicializamos con git y donde vamos haciendo “commits”.
- **Áreas de git:** Son espacios virtuales que sirven para diferenciar de forma interna entre lo que estamos trabajando y lo que se agrega al repositorio.
- **Commit:** Es la acción de crear un punto de historia en el tiempo, se realiza mediante stashear y confirmar cambios.
- **Stashear:** Agregar modificaciones al staging área.
- **Repositorio remoto:** Es cuando el código ya está en una plataforma en internet disponible para compartirlo.



Trabajando con el repositorio local

DEV.F
DESARROLLAMOS(PERSONAS);

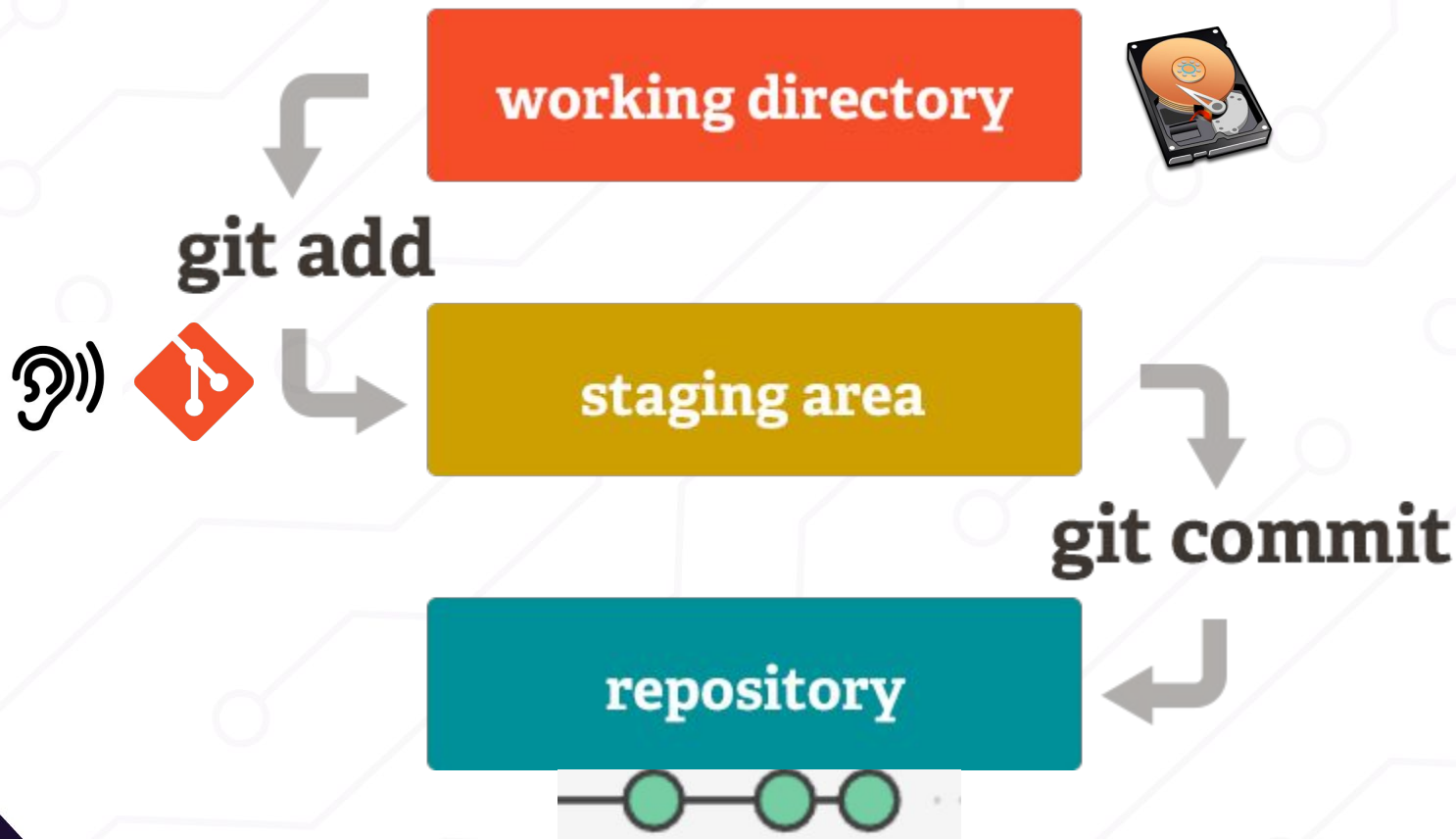
dev

Repositorio local

El repositorio local es una carpeta de nuestro proyecto en local que fue inicializada con el comando `git init` y consta de 3 zonas virtuales fundamentales:

- **Working directory:** Nuestro disco duro o sistema de archivos.
- **Staging area:** Es una área especial donde podemos colocar lo que está listo para agregarse al historial (indexado de git).
- **Repositorio local:** Es lo que se encuentra en el historial de cambios.

Zonas de Git





Creación de un repo y status

Inicializa una carpeta como un repositorio local

- `git init`

Nos muestra el estado de working y staging area

- `git status`

Ver los commits que hemos realizado

- `git log`
- `git log --oneline`
- `git log --graph`

El profe: Quedo claro o tienen alguna duda?

Yo que estuve dormido toda la clase:

refasil



Listos?

Commit



Hacer un commit

- `git add .`
- `git commit -m "Comentario"`

Stage / Unstage

Agregamos todos los archivos al staging area

`git add .`

Agregamos el archivo.txt al staging area

`git add archivo.txt`

Quitamos el archivo.txt del staging area

`git rm --cached archivo.txt`

Quitamos el archivo.txt del staging area

`git restore --staged archivo.txt`

Commits

Commits

Se crea un punto en la historia con un mensaje.

- `git commit -m "Comentario"`

Staging y commiteamos al mismo tiempo.

- `git commit -am "Comentario"`

Actualiza el último mensaje del commit

- `git commit --amend -m "Comentario"`

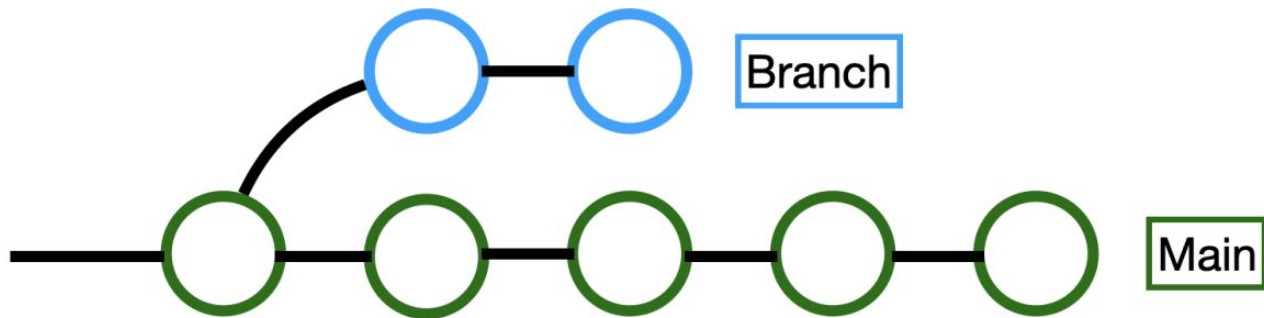
Administración de branches

DEV.F
DESARROLLAMOS(PERSONAS);

dev

Ramas (Branch)

La rama por defecto de Git es la rama **main** (antes **master**). En cada confirmación de cambios que realicemos, la rama irá avanzando automáticamente.



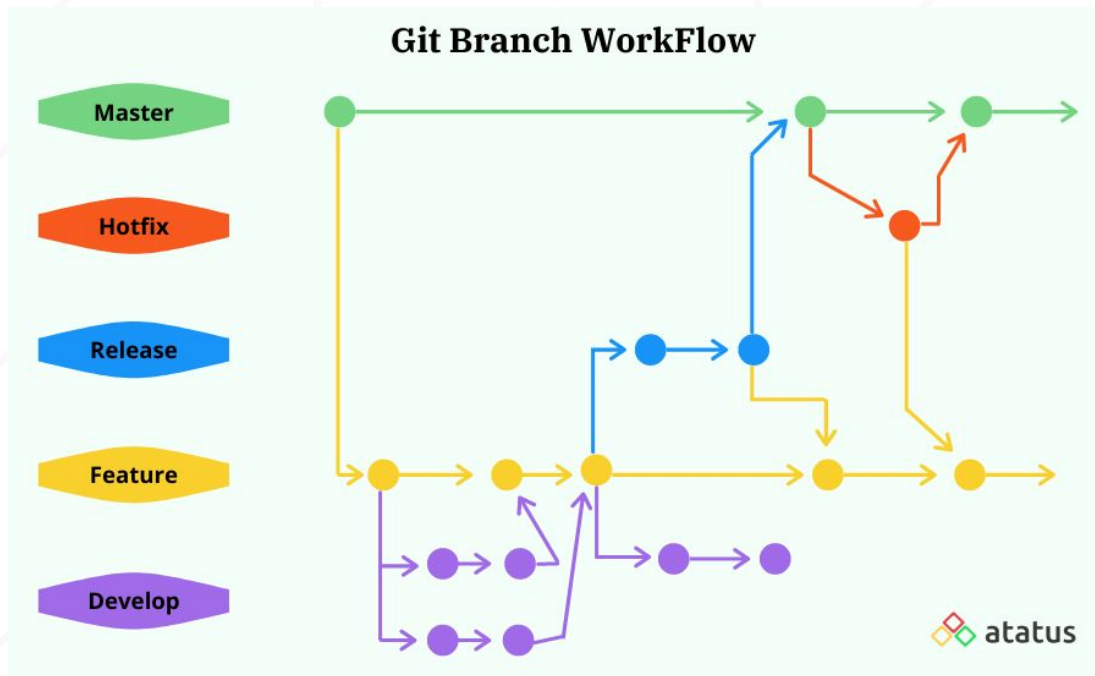
Ramas (Branch)

Las **branch** son bifurcaciones o variantes de un repositorio, estas pueden contener diferentes archivos y carpetas o tener todo igual excepto por algunas líneas de código.



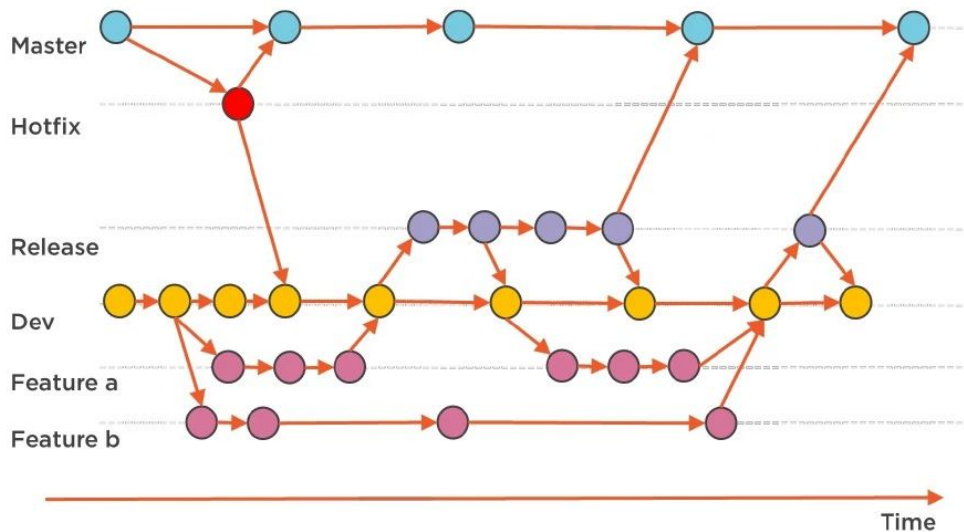
Nomenclatura de ramas

- **main (antes master):** Es la rama donde se debe colocar el código que va a producción
- **develop:** Es la rama de pruebas de desarrollo.
- **feature/my-feature:** Es una rama que se genera a partir de develop o main para agregar una nueva funcionalidad.
- **hotfix/my-hotfix:** Es una rama que se genera a partir de develop o main para agregar una corrección de algún bug.



¿Por qué usar ramas y commits?

Las ramas y los commits son las herramientas que nos van a permitir **controlar el flujo de cambios** e historial de los mismos, haciendo nuestra distribución del código mucho más controlada y administrable.



Comandos para administración de branches

Mostrar las ramas que tenemos

- `git branch`

Creamos una nueva rama

- `git branch newBranchName`

Nos cambiamos a la rama nombre

- `git checkout branchName`

Crear y cambiarse a una nueva rama

- `git checkout -b newBranchName`

Crear y cambiarse a una nueva rama

- `git switch -c branchName`

Comandos para administración de branches

Borrar una rama

- `git branch -D branchName`

Regresar en el tiempo



MASTER

DEV.F

Comando para viajar en el tiempo

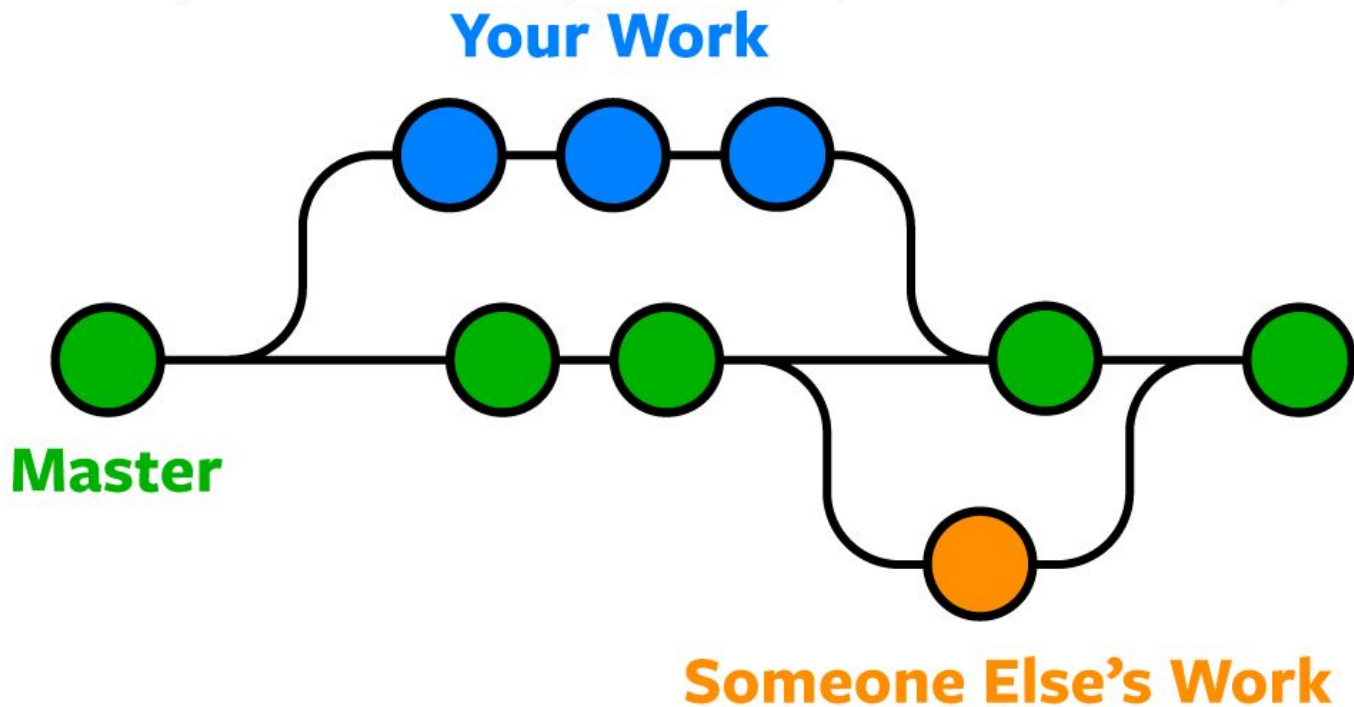
Cambiarnos a un commit en específico

`git checkout hash(id del commit)`

Regresar al commit más reciente de la rama actual

`git checkout .`

Fusión de cambios



Comandos para administración de branches

Unimos cambios desde una rama destino a la actual

- `git merge sourceBranchName`

NOTA: Antes de hacer un merge comprobar que estamos en la rama destino (git pull) y que no tenemos nada en staging area ni working directory.

Resolver conflictos

DEV.F
DESARROLLAMOS(PERSONAS);

dev

Resolver conflicto

- La unión de ramas puede ocasionar que tengamos **conflictos** si en una y otra rama se toca el mismo archivo en las mismas líneas.
- Para solucionar, simplemente debemos **revisar el archivo y elegir manualmente con que código queremos quedarnos**, después realizamos un nuevo commit para confirmar estas modificaciones.

```
You, 33 seconds ago | 1 author (You)
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Conflict Site</title>
7 </head>
8 <body>
9   <div style=" background-color: green; width: 100%; height: 40px;">
Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
10 <<<<<< HEAD (Current Change) You, 31 seconds ago • Uncommitted changes
11     ngxCoder Angular al siguiente nivel
12     =====
13     ngxCoder, Frontend al siguiente nivel
14 >>>>>> feature (Incoming Change)
15     </div>
16 </body>
17 </html>
```

Trabajando con el repositorio remoto

DEV.F
DESARROLLAMOS(PERSONAS);

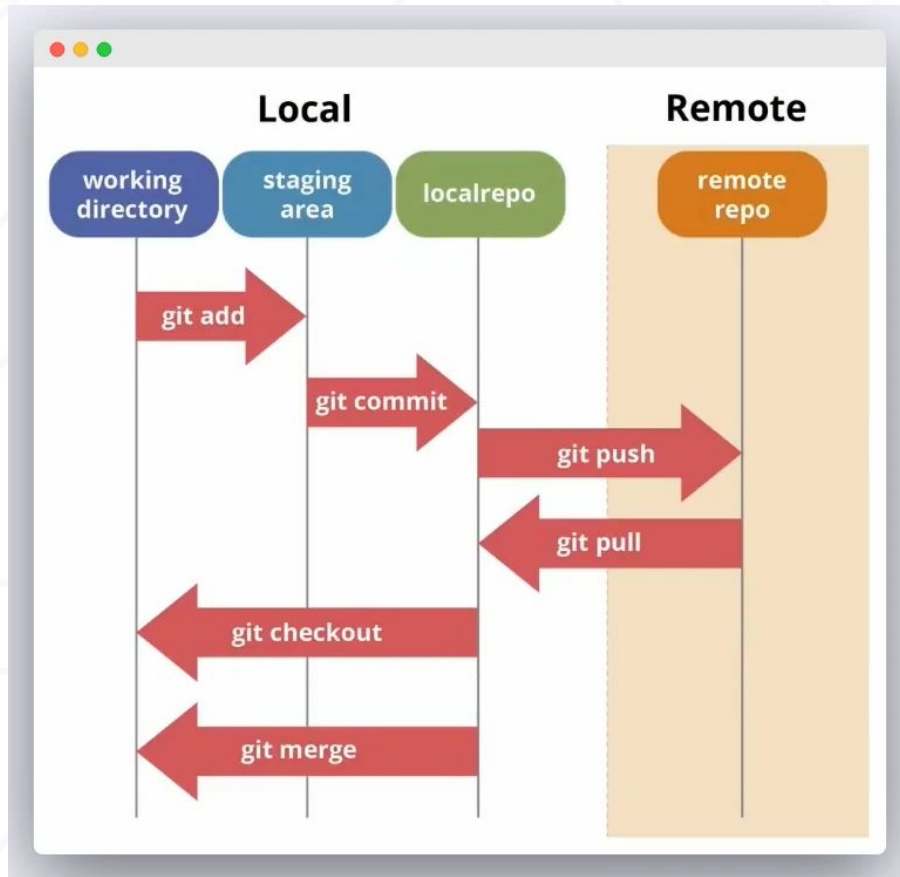
dev

Repositorios remotos

- Los repos remotos son el respaldo de nuestro local y medio de distribución de código.



Flujo con repositorio remoto



¿Cómo conectarse a un repo remoto?

DEV.F
DESARROLLAMOS(PERSONAS);

dev

DEV.F.

Práctica



Conectar un repo local a repo remoto

"Ser programador no es estresante."



-Harold, 27 años.

Existen **2 casos**:

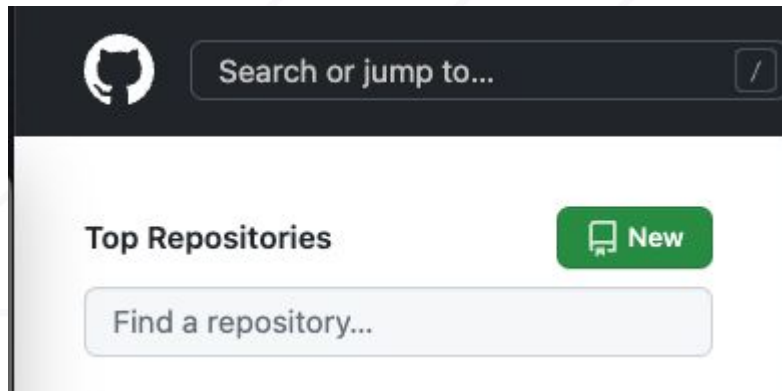
1. **Crear un nuevo repo en github** y clonarlo (ya es un repo) para comenzar a trabajar.
2. **Teniendo un repo local** que voy a inicializar con git y respaldar en remoto.

Caso 1

1. Crear un repositorio en remoto y clonarlo.

Nota: La nueva carpeta clonada estará vacía.

2. Comenzar a agregar archivos, commits y push.



Caso 2

1. Inicializar con git nuestra carpeta de trabajo.
2. Hacer un commit.
3. Crear un repositorio remoto.
4. Utilizar los comandos para repositorio preexistente.

```
montoyitag@monair devf % git init
```

Subir/bajar cambios al repo remoto

Clonar repositorio existente

- `git clone url`

NOTA: Clonar significa descargar una copia de un repositorio remoto a nuestra máquina local.

Conectarse con el repo remoto

Ver si nuestro repo local está conectado a algún repo remoto

```
git remote -v
```

Agregar la conexión de nuestro repo local al remoto

```
git remote add originName url
```

Agregar la conexión de nuestro repo local al remoto

```
git remote set-url originName myNewUrl
```

Subir/bajar cambios al repo remoto

Obtenemos cambios más recientes de la rama

git pull *alias branch*

Enviamos cambios a repositorio remoto

git push *alias branch*

Subir todas las ramas desde local a remoto

git push --all origin

Fusión de ramas

Merges

DEV.F
DESARROLLAMOS(PERSONAS);

dev

Conectar un repositorio local a un remoto

Existen 2 **formas** de hacer **merge**.

1. **Merge local.**
2. **Pull request.**

Merge local

1. Cambiarse a la rama de destino.
 - `git checkout main`
2. Ejecutar el comando merge en la rama destino
 - `git merge develop`

Flujo merge por pull request

1. Hacer un commit.
 - `git add .`
 - `git commit -m "Comentario"`
2. Enviar al repositorio remoto.
 - `git push originName branch`
3. Crear la pull request en el repositorio remoto (rama base y rama destino) y agregar revisores.
4. Los revisores aceptan la PR (Pull request) y se hace el merge.
5. Obtener los cambios en nuestra compu mediante.
 - `git pull originName branch.`

Otros comandos útiles

DEV.F
DESARROLLAMOS(PERSONAS);

dev

Otros comandos útiles

Descartar cambios y eliminarlos del stagin y working directory

- `git reset --hard HEAD^`

Descartar cambios y eliminarlos del stagin y working directory

- `git reset --soft HEAD~1`

Otros comandos útiles

Stashear

git stash

Unstashear

git stash pop

Ver la pila stash

git stash list

Otros comandos útiles

Limpia la cache de git

```
git rm --cached . r
```

Muestra todas las ramas con sus distintos commit de forma gráfica

```
git log --all --decorate --oneline --graph
```

Buenas prácticas

DEV.F
DESARROLLAMOS(PERSONAS);

dev

Buenas prácticas



Existen 3 principales que todo desarrollador debería conocer:

- Gitflow.
- Atomic commits.
- Conventional commits.

Gitflow

Gitflow es un **modelo alternativo de creación de ramas** en Git en el que se utilizan ramas de función y varias ramas principales.

Según este modelo, los desarrolladores crean una rama de función y retrasan su fusión con la rama principal base hasta que la función está completa.

En el caso de hotfix y feature, son borradas una vez integradas al flujo principal.

- **main (antes master).**
- **develop.**
- **release/appV1.0.0.**
- **feature/login.**
- **hotfix/logo-app.**

Atomic commits

Un **commit atómico** es un commit que está enfocado en **agregar al historial una sola cosa**, puede ser un feature, resolver un bug, un refactor, una actualización, una tarea, etc.

Ejemplos

`git commit -m "feat: agrega el menu"`

`git commit -m "fix: corrige la paleta de colores"`

`git commit -m "docs: agrega documentacion de la pagina principal"`

Conventional commits

Es una convención en el formato de los mensajes de los commits. Esta convención define una serie de reglas que hacen muy sencillo tanto la legibilidad del histórico del repositorio como el poder tener herramientas que automaticen procesos basándose en el historial de commits.

- [Docs](#)

Tipos de conventional commits

- **feat:** Nuevas características.
- **chore:** Cosas que no aportan un req funcional pero posiblemente si un req no funcional.
- **fix:** Corrección de errores.
- **docs:** Commits con documentación o comentarios.
- **style:** Cambios de legibilidad o formateo de código que no afecta a funcionalidad.
- **refactor:** cambio de código o arquitectura que no corrige errores ni añade funcionalidad, pero mejora el código.
- **test:** Para añadir o arreglar tests.

Documentación

DEV.F
DESARROLLAMOS(PERSONAS);

dev

Documentación

- Que es git y github.
- Cheat sheet.
- Curso de hola mundo.
- Curso free code camp.
- Git exercise w3Schools.
- VSC y sus herramientas para git.
- Git blame y git graph.