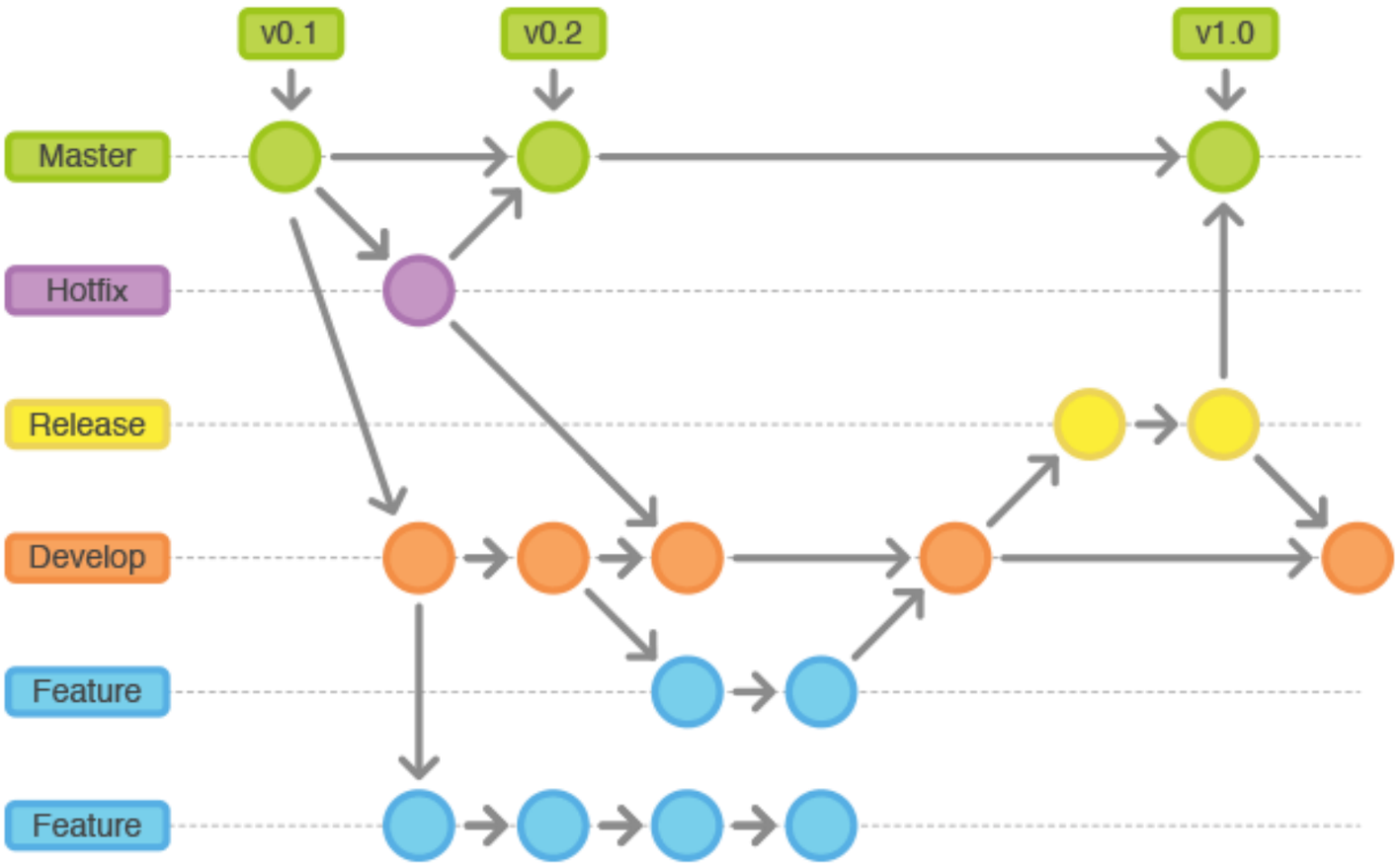
Repasemos algunos conceptos

Antes de comenzar

GIT



Git es un sistema de control de versiones distribuido gratuito y de código abierto diseñado para manejar todo, desde proyectos pequeños hasta proyectos muy grandes, con rapidez y eficiencia.

CONFIGURACIÓN LOCAL

Descarga: https://git-scm.com/downloads

git config --global user.name <username>
git config --global user.email <email>

Éstas son configuraciones por primera vez. Para realizar otras configuraciones adicionales puedes visitar: https://git-scm.com/book/es/v2/Inicio---Sobre-el-Control-de-Versiones-Configurando-Git-por-primera-vez

Nota: Sólo necesitas hacer esto una vez si especificas la opción --global, ya que Git siempre usará esta información para todo lo que hagas en ese sistema.

REPOSITORIOS REMOTOS

Bitbucket

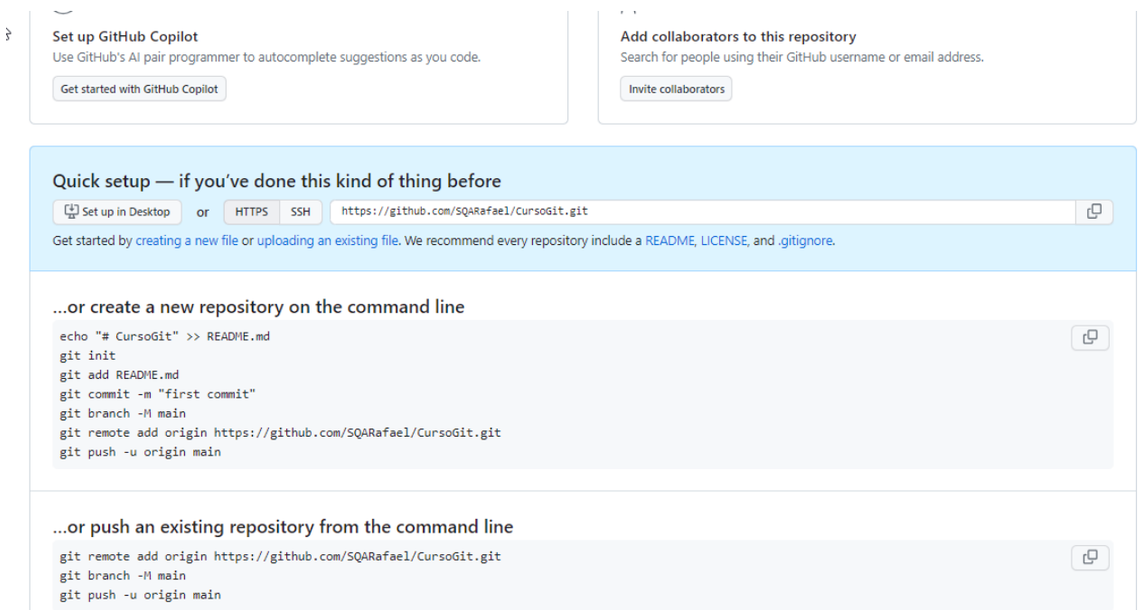
aws

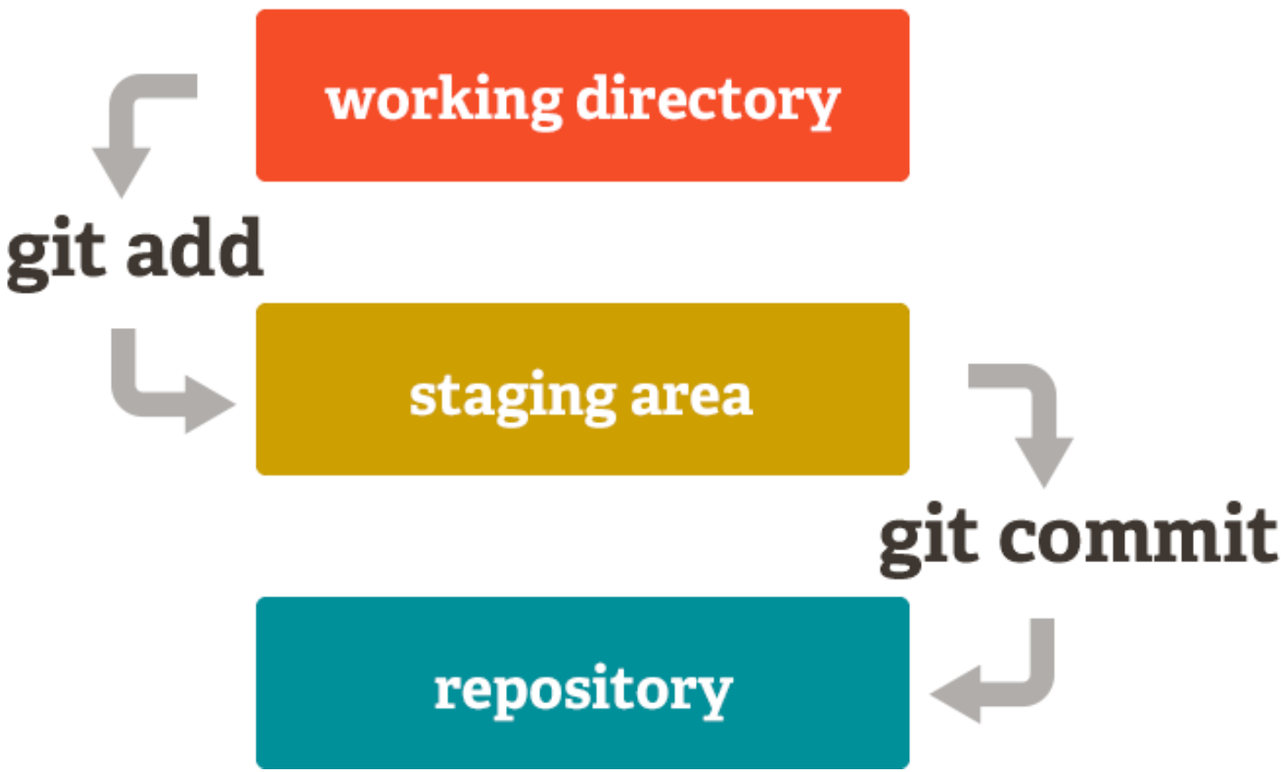
git

Crear cuenta en la web de GitHub:
https://github.com/signup

Cree el proyecto en github, siguiendo el paso a paso que le brinda la página, al finalizar, la página le brinda una serie de líneas de código las cuales sirven para enlazar su proyecto local al repositorio en la nube

Iniciar Git en el proyecto, esto en caso de que el IDE no lo inicie de forma automática:
git init




working directory
git add
staging area
git commit
repository

git

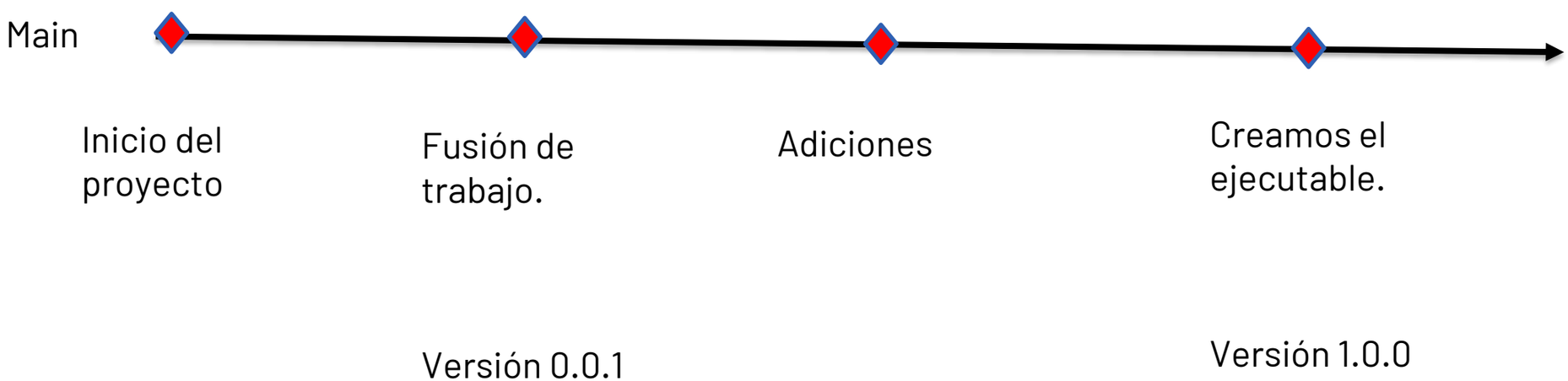
GITHUB: comandos iniciales.

- git init
- git add README.md
- git commit -m "first commit"
- git branch -M main
- git remote add origin https://github.com/SQARafael/CursoGit.git
- git push -u origin main
- git clone https://github.com/SQARafael/CursoGit.git

GIT: comandos necesarios

- **git branch:** conocer en qué rama estas.
- **git status:** conocer si tus archivos estan preparados para ser agregados al stage
- **git config** --global alias.lg "log --graph --abbrev-commit --decorate --format=format:'%C(bold blue)%h%C(reset) - %C(bold green)(%ar)%C(reset) %C(white)%s%C(reset) %C(dim white)- %an%C(reset)%C(bold yellow)%d%C(reset)' --all"
- **git reset** --soft HEAD^
- **git reset** --mixed " hash al cual te vas a mover"
- **git reset** –hard " hash que se quiere destruir "
- **git merge** "rama que se va a fusionar"

GIT: Tags


Main
Inicio del proyecto
Fusión de trabajo.
Adiciones
Creamos el ejecutable.
Versión 0.0.1
Versión 1.0.0

Cuando un equipo de automatizadores trabajan en simultáneo, es necesario tener un proceso establecido para implementar múltiples cambios a la vez, a esto se le conoce cómo estrategia de branching. La idea principal de una estrategia de branching es aislar el trabajo en diferentes ramas, lo que permite organizarlo y admitir de forma fácil y continua múltiples versiones de código en producción.

Tipos de estrategias de branching

Una estrategia de branching son un conjunto de reglas que se definen desde el inicio del proyecto/sprint y que pueden seguir los equipos para escribir, fusionar e implementar código compartido. Con ello, es posible trabajar en paralelo para lograr versiones más rápidas y menos conflictos al fusionar los cambios. A continuación, revisaremos algunas estrategias de control de versiones más usadas:

Tipos de estrategias de branching

	Git Flow Branch Strategy	GitHub Flow Branch Strategy	GitLab Flow Branch Strategy	Desarrollo basado en troncos
En que consiste	Se **crea** una **rama** de **características** desde la **rama maestra**. Cuando se completan los cambios, el automatizador vuelve a **fusionar** estos cambios en la rama maestra para su lanzamiento.	Comienza con la **rama maestra** y se crean desde la mismas **ramas** de **características.** Luego se **fusionan** de nuevo en la master y se **eliminan** las ramas de características.	Es similar a la estrategia de **GitHub Flow**, pero adiciona **ramas** de **entorno**, es decir, producción y preproducción, o ramas de **lanzamiento**, según la situación.	**No** requiere **ramas**, sino que los desarrolladores **integran** sus **cambios** en un **tronco compartido** al menos una vez al día. Este tronco compartido debería estar listo para su **lanzamiento** en cualquier momento.
Ramas	**Principales**: - Master - Develop **De apoyo**: - Feature - Release - Hotfix	**Principales**: - Master **De apoyo:** - Feature - HotFix	**Principales**: - Master **De apoyo:** - Release - Preproduction - Production - HotFix	**Principales**: - Master

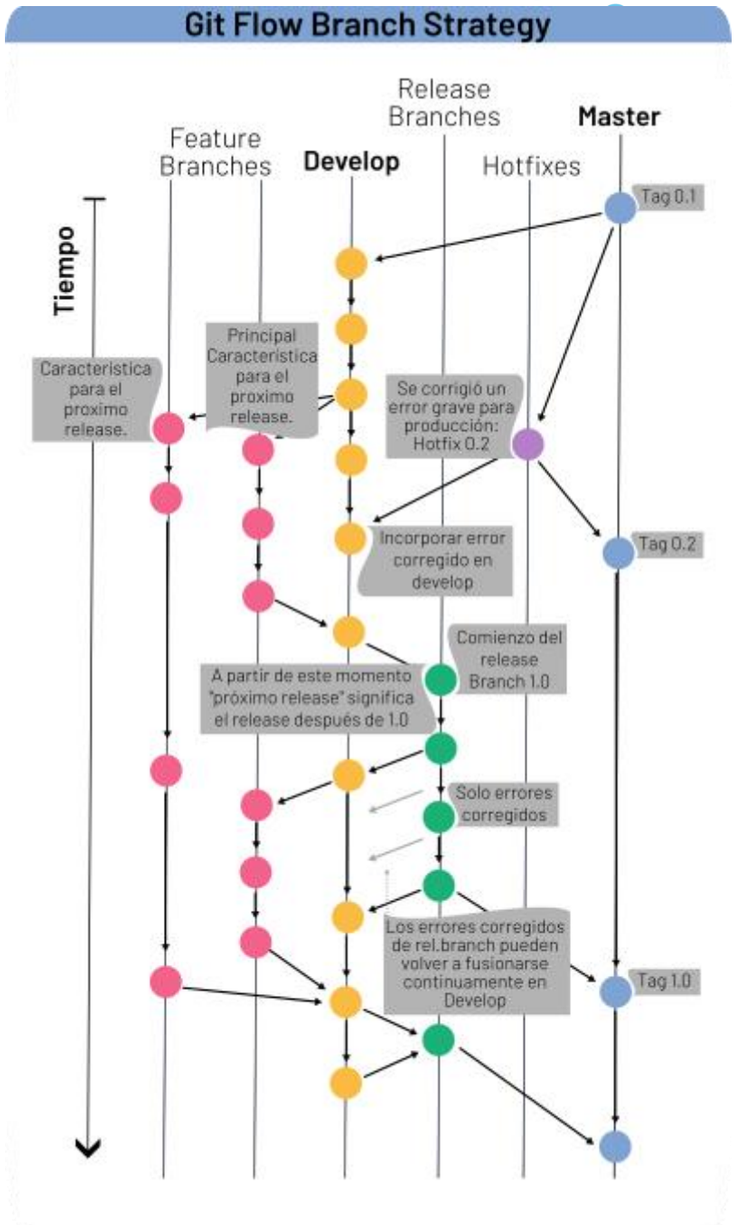
Tipos de estrategias de branching

	Git Flow Branch Strategy	GitHub Flow Branch Strategy	GitLab Flow Branch Strategy	Desarrollo basado en troncos
¿Cuándo es adecuado usarla?	Adecuada para cuando se requiere tener **múltiples versiones** de código de producción; y productos con un ciclo **largo** de **mantenimiento** de una versión.	Adecuada para **equipos pequeños** y **aplicaciones web** y es ideal cuando necesita mantener **una sola versión** de producción.	Excelente cuando desea **mantener** varios **entornos** y cuando prefiere tener un entorno de ensayo separado del entorno de **producción**.	Adecuada cuando los **cambios** se van a realizar con **mayor frecuencia** en el tronco, a menudo varias veces al día (**CI**), lo que permite que las características se publiquen mucho más rápido (**CD**).
Ventaja y desventajas	Las distintas ramas permiten **organizar** mejor el trabajo, pero pueden llegar a ser díficiles de administrar. **No admite CI/CD**.	**No** hay ninguna **rama** de **desarrollo**, lo que permite CI/CD, pero puede **no** ser **adecuado** para controlar **varias versiones** de código.	Permite que el código pase a través de **entornos internos** antes de que llegue a producción, aunque puede conducir a una colaboración desordenada.	Mejora la **colaboración** y **elimina** los **conflictos** de fusión, ya que los desarrolladores experimentados están impulsando **pequeños cambios** con mucha más frecuencia

Veamos un ejemplo

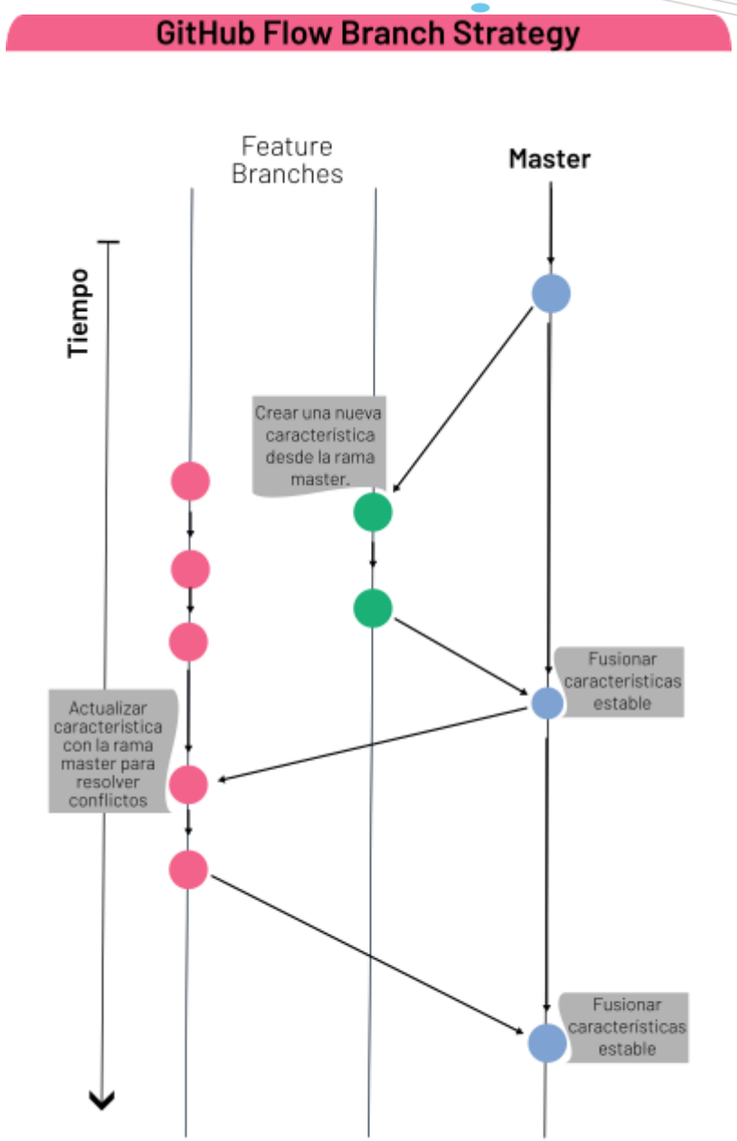
Un desarrollador crea una rama "feature/login" desde "develop" para agregar una nueva función de inicio de sesión. Después de la revisión, el desarrollador fusiona la rama "feature/login" de vuelta a "develop".

Cuando se alcanza un hito importante, como una nueva versión, se crea una rama de lanzamiento para realizar pruebas finales antes de fusionarla en la rama "master" para ser implementada.



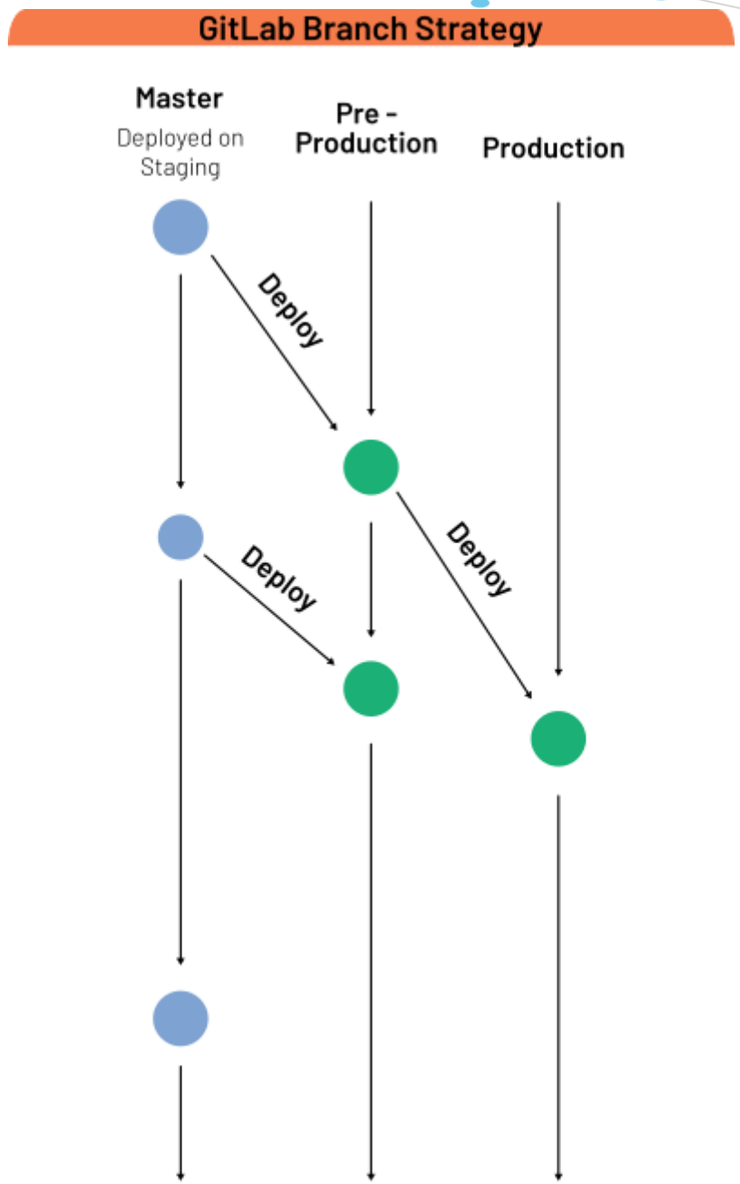
Veamos un ejemplo

El desarrollador crea una rama "feature/login" desde "master" para agregar una ueva funcionalidad de inicio de sesión. Después de la revisión, fusiona la rama "feature/login" de vuelta a "master". Cuando se alcanza un hito importante, como una nueva versión, se crea una rama de lanzamiento para realizar pruebas finales antes de fusionarla en la rama "master" para ser implementada.



Veamos un ejemplo

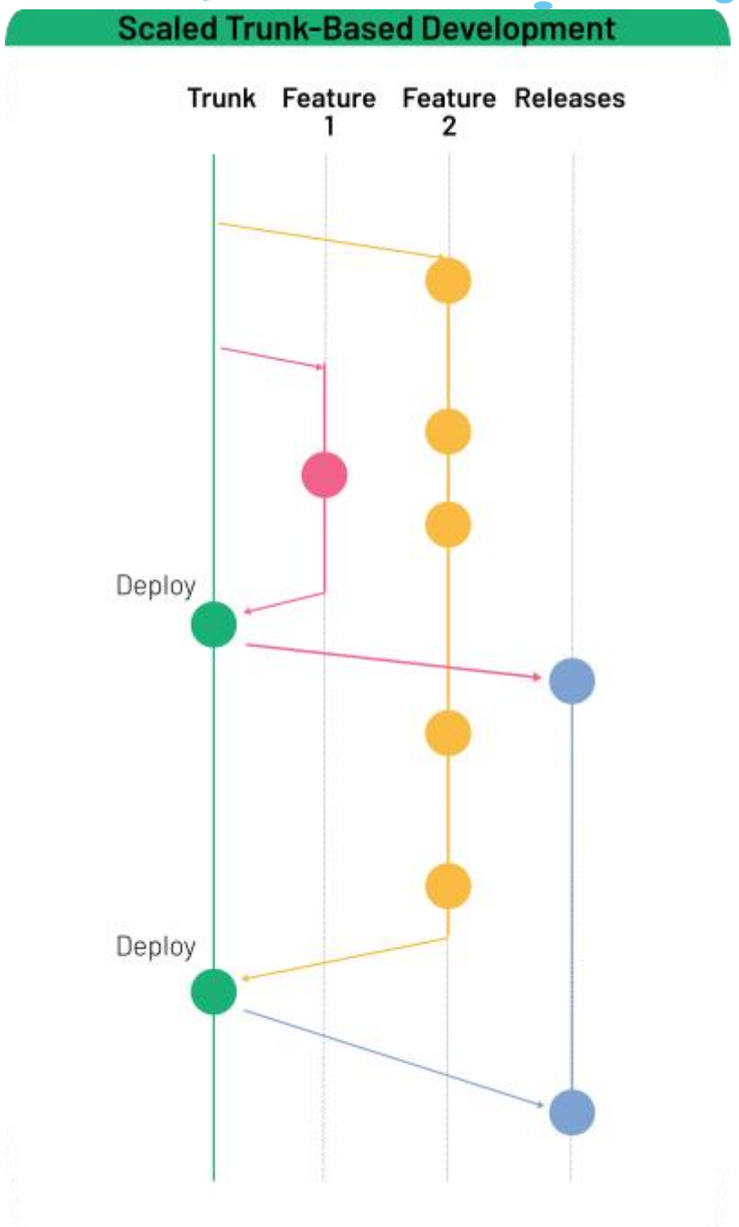
Funciona parecido a GitHub, pero cuando se alcanza un hito importante, como una nueva versión, se crea una rama de preproducción para diferentes entornos, como test, acceptance, etc, antes de fusionarla en la rama "master" para ser implementada.



Veamos un ejemplo

Se crea una nueva función de inicio de sesión directamente en "master" y envía la solicitud de revisión. Después de la revisión y de las pruebas automatizadas, puede crearse una rama de lanzamiento para realizar pruebas finales antes de fusionarla en la "master".

En resumen, cada herramienta de control de versiones puede sugerir cómo manejar la estrategia. Recuerde que es fundamental que la estrategia para el control de versiones de los scripts de automatización esté alineada con la estrategia usada para el código de desarrollo. Esta dependerá de los estándares definidos y podrá variar incluso de una aplicación a otra.



Tips para entender la estrategia de control de versiones

Cuando llegas a un nuevo equipo de desarrollo, es importante entender el flujo de control de versiones que utilizan. A continuación, se presentan algunas preguntas que puedes hacer para entender mejor el flujo de control de versiones:

- ✓ ¿Qué sistema de control de versiones utiliza el equipo?
- ✓ ¿Cómo es la estrategia de control de versiones? ¿Es un estándar o le hicieron algún ajuste?
- ✓ ¿Cómo se estructuran los repositorios? ¿Quién y cuando los crea?
- ✓ ¿Cómo se manejan las ramas? ¿Qué tipo de ramas se manejan? ¿Quién y cuándo las crea?
- ✓ ¿Hay algún proceso formal para crear y fusionar ramas?
- ✓ ¿Cómo se manejan las confirmaciones de código (commits)? ¿Quién las aprueba?
- ✓ ¿Hay algún proceso formal para escribir mensajes de confirmación y etiquetas (tags)?
- ✓ ¿Cómo se manejan los conflictos de fusión? ¿Hay algún proceso formal para ello?
- ✓ ¿Cómo se integra el sistema de control de versiones con herramientas de CI/CD?
- ✓ ¿Cómo se manejan los problemas de seguridad?

Finalmente, no existe la estrategia perfecta, esta dependerá del equipo y de la naturaleza y complejidad de su proyecto, por lo que esto debe evaluarse caso por caso. También está bien comenzar con una estrategia y adaptarla con el tiempo de acuerdo con sus necesidades.