Una guía práctica de:

GIT - ABC

jamescardona11.com

JAMES CARDONA

Software Engineer | 57Blocks
jamescardona11.com

Experiencia

Mobile developer, node.js, web3, beginner on Next.js & Clojure - enthusiastic of crypto & NFTs

Por qué Git hoy?

Git es la base para trabajo distribuido y se nos pasa por alto las bases y consideraciones básicas en muchos casos

DESARROLLADOR

Es mucho más que solo copiar código.

• • •

PREGUNTATE:

¿Qué tipo de profesional quiero ser?

¿QUÉ ES GIT?

¿Por qué se usa?

PASO 1: ENTENDER LO BÁSICO

add, commit, status, restore, checkout, log y .gitignore

El primer ciclo de GIT se basa en llevar nuestros archivos del workspace al área de staging para posteriormente llevarlos a nuestro historial definitivamente

¿Qué es un buen commit?

Compacto

La cantidad de cambios deben ser de algo específico

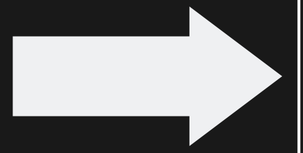
Descriptivo

Debe ser legible:
- el trabajo
- la solución
- pendientes

Agregar explicaciones para tú "yo futuro"

Pensar en el futuro

Un buen commit va a ayudar hacer un tracking de posibles errores en el futuro



<type>: <description>
[optional body]
[] List of changes
[optional scope]

fix: fix the animation for drawer

[X] Add new flag to control the state when is open
[X] Test on android and iOS
[ ] is pending interact with ....

TOK-1766 Jira

https://udacity.github.io/git-styleguide/

PASO 2: BRANCH

Las ramas pueden considerarse como la mejor característica de GIT
create, delete, switch, track, rename

Es la forma fácil de manejar equipos y diferentes propósitos dentro del proyecto.

Creando "sub-espacios" de trabajo

Se debe tener cuidado y a su vez unas buenas reglas para poder aprovechar esto y que no se convierta en un dolor de cabeza.

GIT-WORKFLOW

GIT WORKFLOW

Básicamente como se integran los diferentes personas de un equipo

1

¿Qué es?

Es un marco de trabajo de como se administra las branches y el trabajo en equipo

2

¿Cómo lo defino?

Puedes tomar reglas de definidas por un marco de trabajo previo o que el propio equipo defina.

Estas reglas son la forma como el equipo debe trabajar con GIT

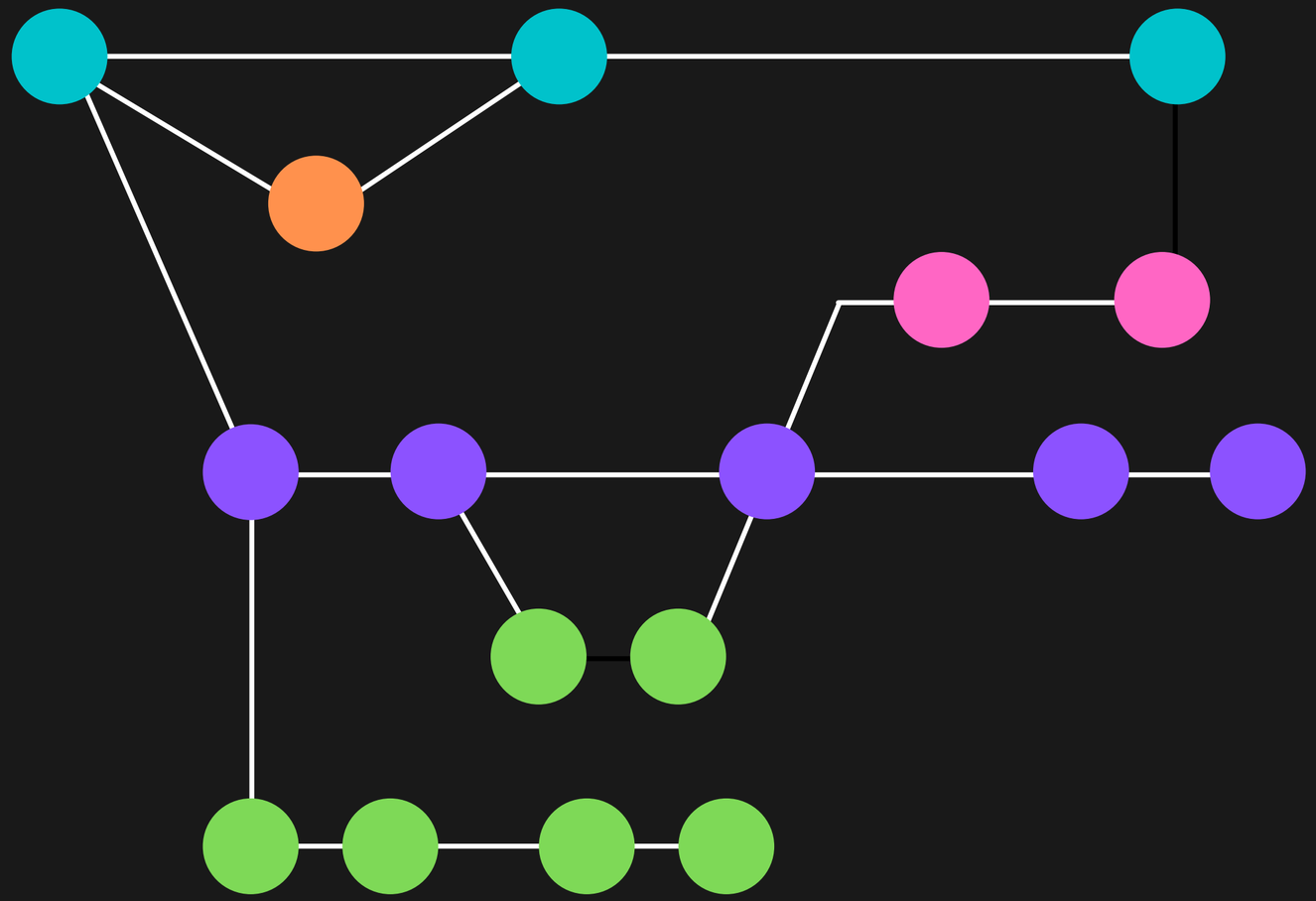
3

Marcos previos

Gitflow
Github flow
Trunk-Based-Development
Git release flow
Gitlab flow
Master only flow

GITFLOW

Main Hotfix Release Develop Feature



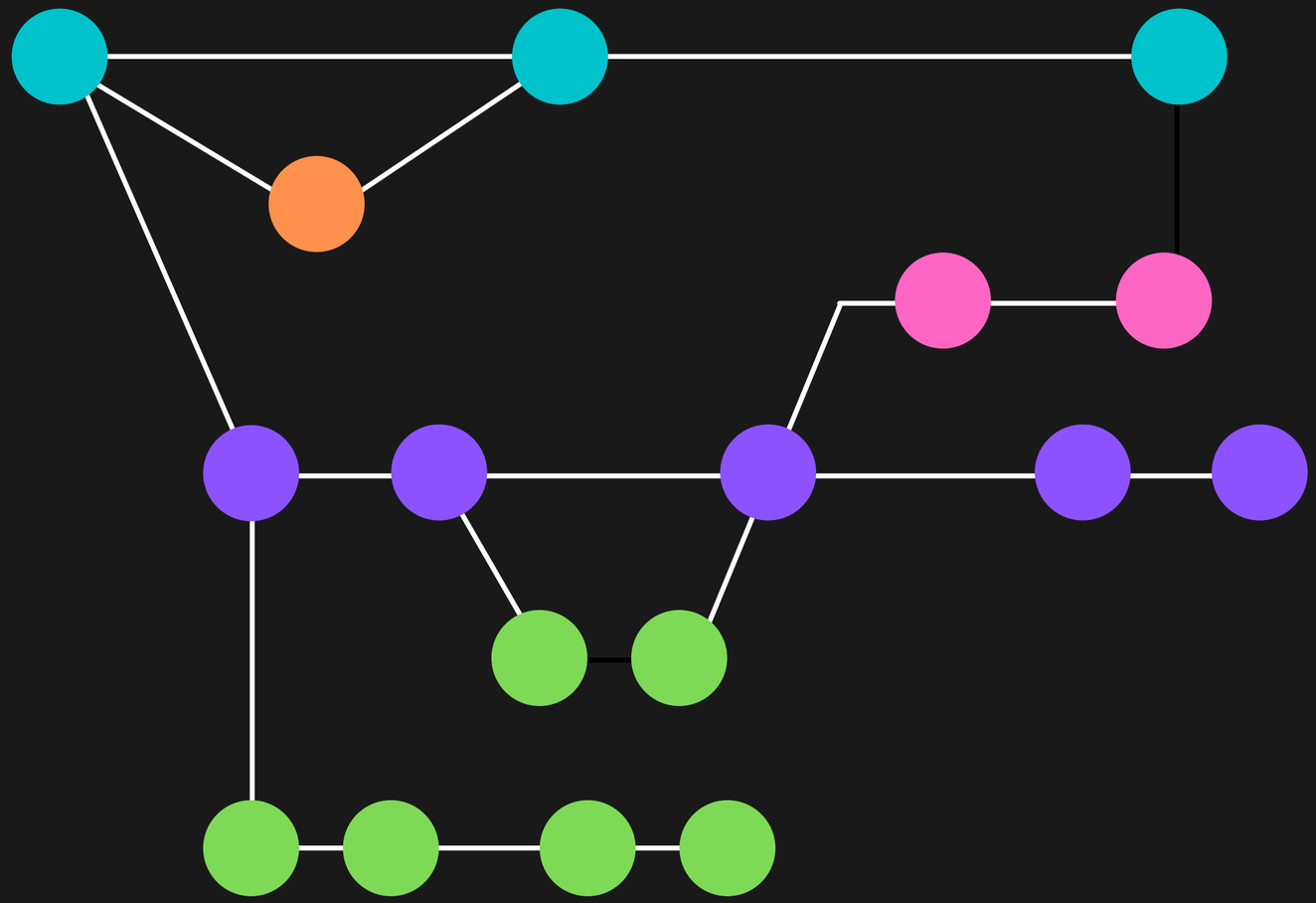
Pros

Riesgos

Conclusión

GITFLOW

Main Hotfix Release Develop Feature



Pros

- Alienta los pull request
- Control estricto de los cambios
- Es fácil de entender

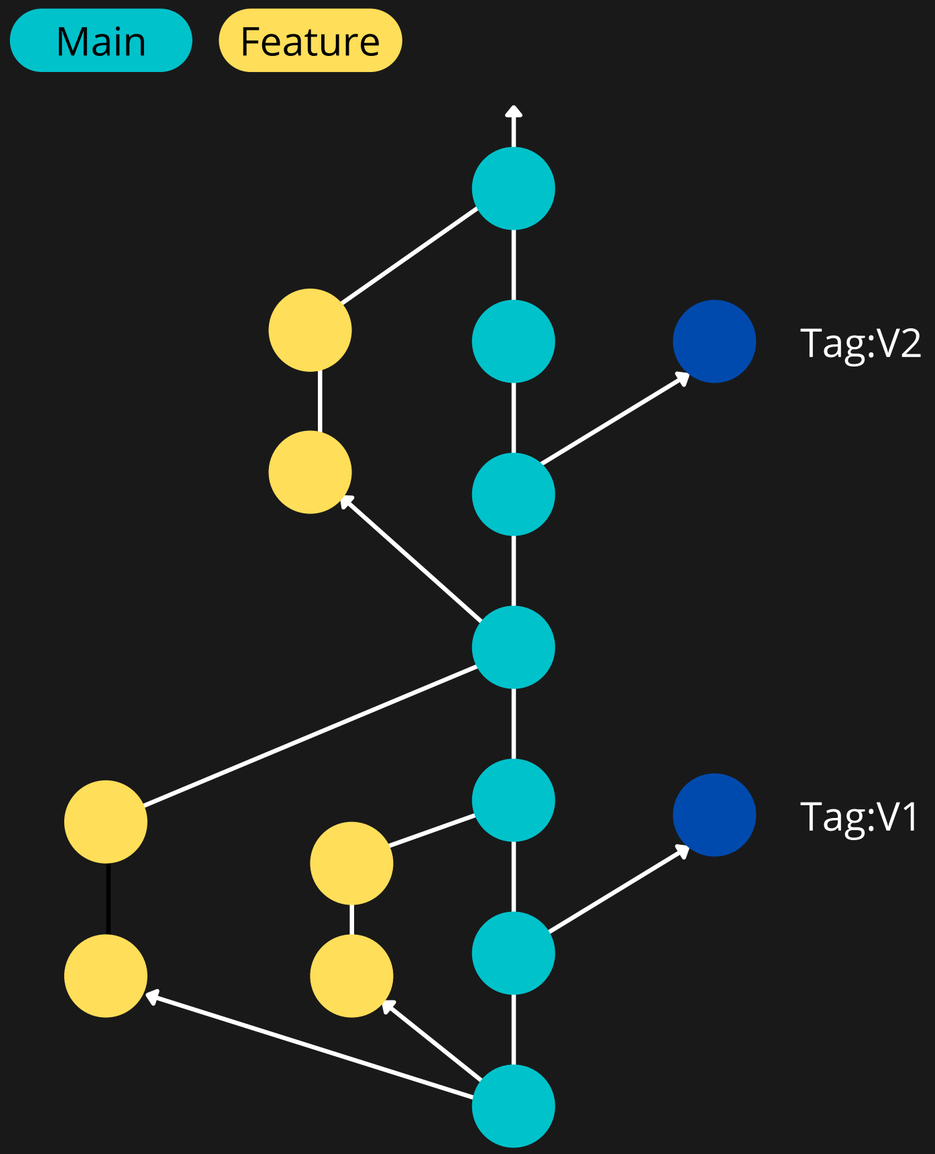
Riesgos

- Los dos primeros beneficios pueden ser contras
- Muchas ramas de larga duración
- Es lento para el release
- No todas las ramas son fiables

Conclusión

Es útil para iniciar o para equipos donde la madurez no es alta, pero puede convertirse un problema para desarrollo ágil

TBD

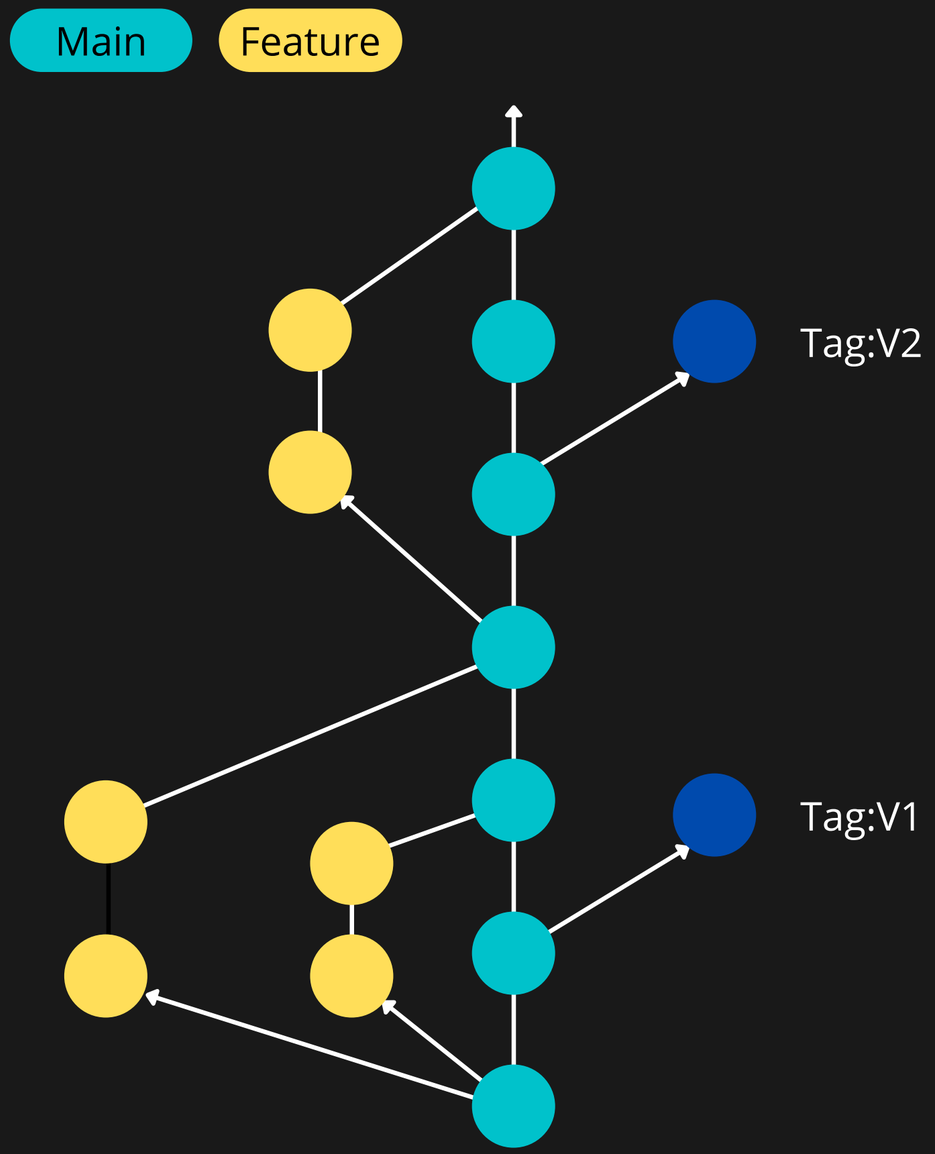


Pros

Riesgos

Conclusión

TBD



Pros

- No existen ramas de larga duración
- Promueve hacer más commits a main (*)
- El trunk siempre está en un estado óptimo
- Release por demanda
- Evitamos grandes conflictos
- Los desarrolladores tiene trabajo más reciente
- Promueve el crecimiento del equipo

Riesgos

- Los desarrolladores deben ser más responsables
- Hay que tener un alto grado de automatización
- No todo debe ser un release * features flags

Conclusión

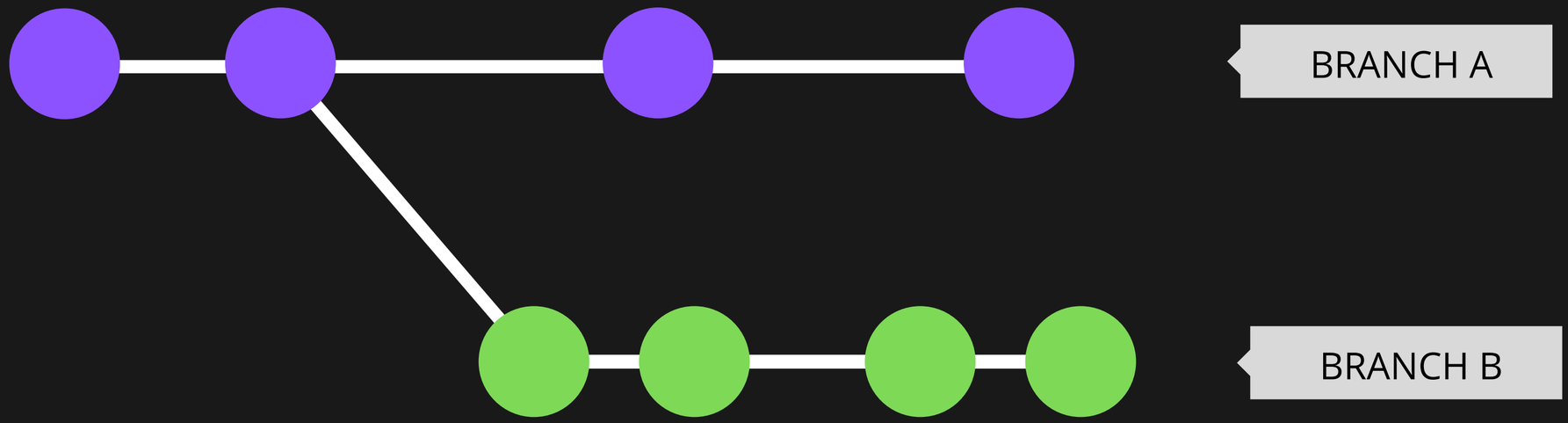
Se debe evaluar la madurez del equipo y la frecuencia de deploys. Es adecuado para cambiarse de gitflow

¿ESTÁS LISTO PARA PRACTICAR ?

Convertirse en un git master requiere tiempo y práctica como un lenguaje de desarrollo.

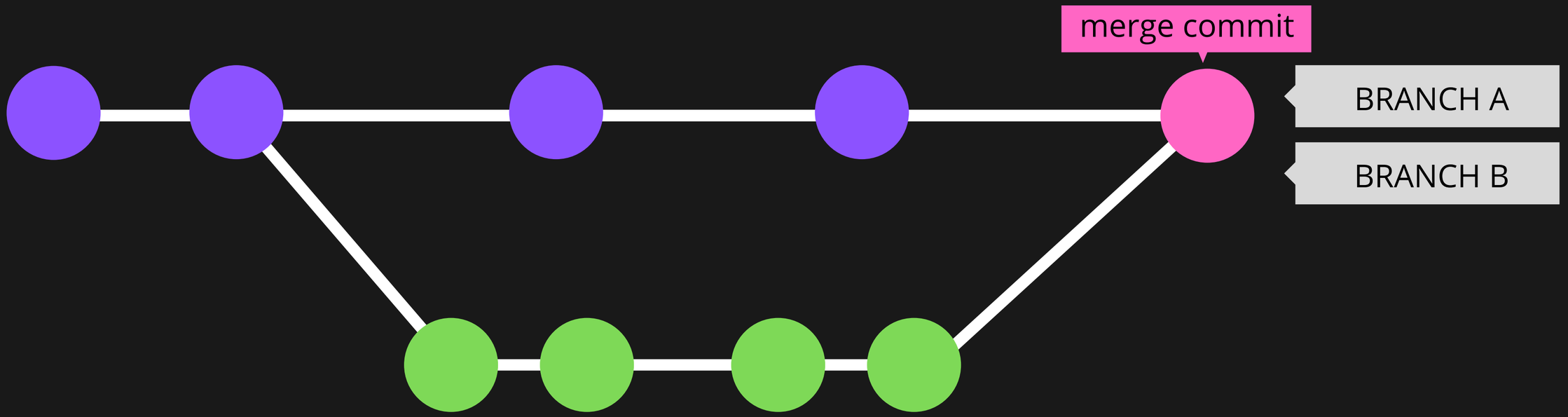
MERGE VS REBASE

Estrategias para "unir" ramas



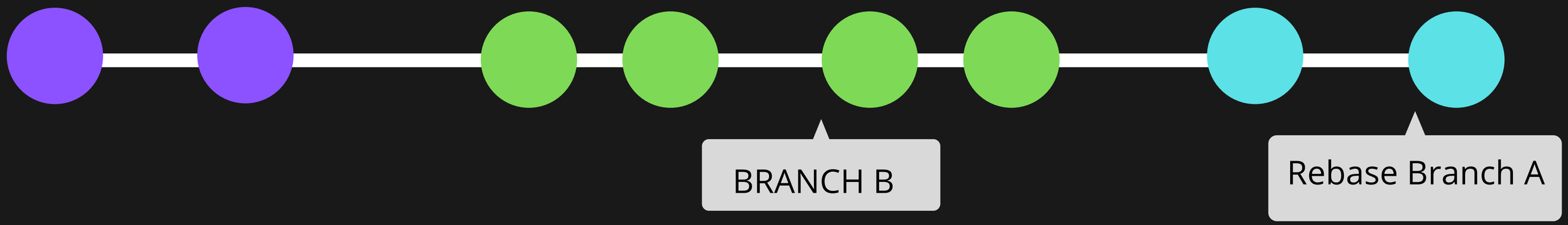
MERGE

git merge <branch>



REBASE

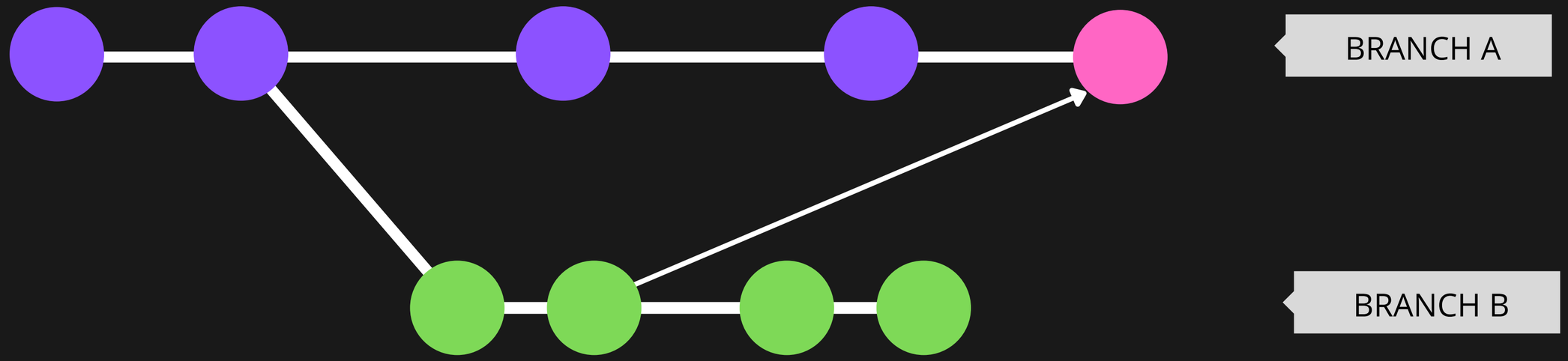
git rebase <branch>



Regla de oro: No usar rebase sobre commits que ya están compartidos en remote

CHERRY PICK

git cherry-pick <branch>



Esto reescribe la historia y no se debe usar
en remplazo del merge

DESHACER ERRORES

1 Descartar todos los cambios en un archivo

2 Restaurar un archivo eliminado (uncommitted)

3 Descartar algunas líneas de un archivo (uncommitted)

4 Descartar todos los cambios en local (uncommitted)

5 Arreglar el último commit

6 Revertir un commit en la mitad del proceso

7 Resetear a una versión anterior

8 Resetear un archivo a versión anterior

DESHACER ERRORES

9 Recuperar commits borrados

10 Recuperar una rama borrada

11 Mover un commit a una nueva rama

12 Mover un commit a una rama diferente

13 Agregar cambios a un commit viejo

14* Rebase interactivo

GRACIAS

Próximos Tech talks:

- Estructuras de datos
- ¿Qué es un desarrollador senior?
- El poder de la programación funcional
- ¿Cómo escalar con arquitectura de software?