

UNIVERSIDAD DE LA REPÚBLICA
CONSEJO DE EDUCACIÓN TÉCNICO PROFESIONAL

Programación de Aplicaciones GIT



Tecnólogo en Informática

¿Qué es GIT?

- ✓ Sistema de control de versiones creado por Linus Torvalds que registra los cambios realizado en un archivo o un conjunto de archivos.

Versión 1



Versión 2



¿Qué es GIT?

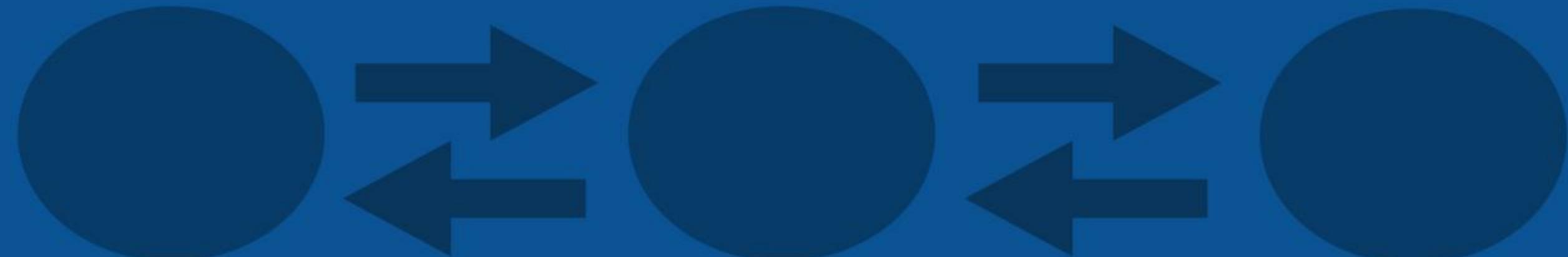


- Con GIT podremos "viajar en el tiempo" y detectaremos mejor los errores que aparecen en diferentes versiones de nuestro proyecto.
- Podremos tener nuestro proyecto organizado. No manejaremos un conjunto de carpetas, manejaremos algo llamado commits.

Versión 1

Versión 2

Versión 3



Los TRES estados

WORKING DIRECTORY, en donde estaremos editando y trabajando sobre el proyecto.

STAGING AREA, donde elegimos que archivos están para pasar al tercer estado y que archivos no lo están.

REPOSITORY, es el registro del trabajo realizado con algo llamado commit.

Flujo de trabajo

Working Area



Trabajamos, modificando, editando o eliminando archivos.

Staging Area



Elegimos que archivos están listos, y cuáles no.

Repository



Los registramos, confirmamos en nuestro proyecto.



Instalación y Configuraciones básicas

Instalación en Linux y Unix

✓ **Debian/Ubuntu**

apt-get install git

✓ **Fedora**

dnf install git

✓ **Arch Linux**

pacman -S git

✓ **openSUSE**

zypper install git

En caso de Windows
se tendrá un git bash
al instalar



Configuraciones iniciales

 **git config --global user.name "Yonapla"**

 **git config --global user.mail "yonapla@gmail.com"m**

 **git config --global color.ui true**

Para resaltar los diferentes resultados

que nos puede dar GIT

 **git config --global --list**





Comandos Básicos

> git init

- Ⓐ Es el comando para indicar el inicio de un nuevo proyecto.
- Ⓐ De esta forma se le indica a GIT que comience a monitorear todos los cambios realizados.
- Ⓐ Se utiliza solo una vez al iniciar git en nuestro proyecto.



> git status

- ① Nos indica el estado del proyecto.
- ② Archivos(modificados o nuevos) que NO fueron agregados.
- ③ Archivos(modificados o nuevos) que SI fueron agregados.
- ④ Archivos que fueron "commiteados" y no sufrieron modificaciones no aparecerán.



>git add y git rm --cached

- Ⓐ Con el primer comando indicamos que archivos están listos para el siguiente paso, para luego ser "commiteados".
- Ⓐ Se pueden agregar todos los archivos nuevos y/o modificados con el parámetro -A.
- Ⓐ Con el segundo comando retiramos un archivo de la staging area.



gitignore.

- Ⓐ Existen tipos de archivos que nunca queremos agregarlo al staging area, a los cuales no nos interesa que GIT haga seguimiento.
- Ⓐ En la carpeta del proyecto para ello creamos el archivo oculto .gitignore

touch .gitignore

- Ⓐ Por ejemplo no queremos agregar los archivos .o, lo indicamos:

***.o**



>git commit



¿Qué es estar agregando archivos?

¿Cuál es la función de agregarlos?

- ⑧ Con el comando git commit guardamos, confirmamos los cambios con un mensaje para poder identificarlos.

git commit –m "Mensaje"

- ⑧ Luego del commit al realizar un git status se notificará que no han habido cambios.



Flujo de trabajo



Flujo de trabajo



>git log

- Ⓐ Este comando lista ordenada de todos los commits con su respectiva información.
- Ⓑ La información que se puede apreciar de cada commit es:
 - Autor
 - Fecha
 - Mensaje
 - Código SHA (identifica los commits)



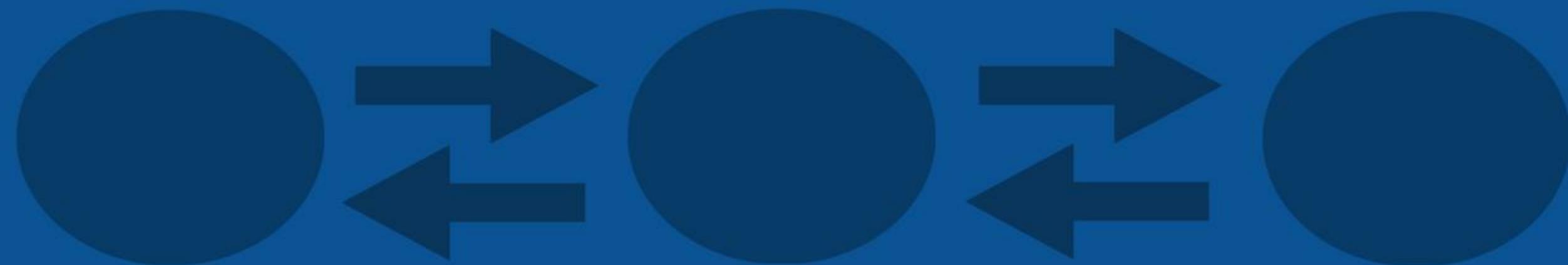
>git checkout

Este comando permite movernos en diferentes commits o ramas de nuestro proyecto.

Versión 1

Versión 2

Versión 3



Para ello utilizamos el código SHA del commit al que queremos pasar.

git checkout



3cfafadd3d2f9347225170e9d8bce8fa6

>git checkout

- Ⓐ Si ejecutamos un git log veremos los commits hasta el commit al cual me pase con el git checkout.
- Ⓑ No tenemos visibilidad a los commits posteriores al commit que nos movimos, ¿cómo lo solucionamos?

git checkout master

- Ⓐ De esta forma nos movemos al último commit que tenemos.



>git reset

Parecido a hacer checkout pero elimina los commits.

soft

Se elimina el commit pero NO los cambios realizados en el código, en la Working Area.

git reset --soft códigoSHA

hard

Se elimina el commit y TAMBIEN los cambios realizados en el código.

git reset --hard códigoSHA





Ramas y Fusiones

Head

- ⓐ Es el término que hace referencia al último commit de donde nos encontramos.



- ⓐ Si realizamos un git checkout a un commit cualquiera, ese commit será nuestro head.



Rama

- Ⓐ Es considerada como una línea del tiempo en el proyecto.



- Ⓐ Se utiliza para:
 - Arreglar errores.
 - Realizar cambios significativos.
 - No estar haciendo todos los commits solo sobre una rama.
 - Separar trabajo entre compañeros.



Rama Master



- Ⓐ Se genera por defecto al hacer git init.
- Ⓐ Es la rama principal y estable del proyecto.
- Ⓐ Es en donde hemos generado los commits hasta el momento.



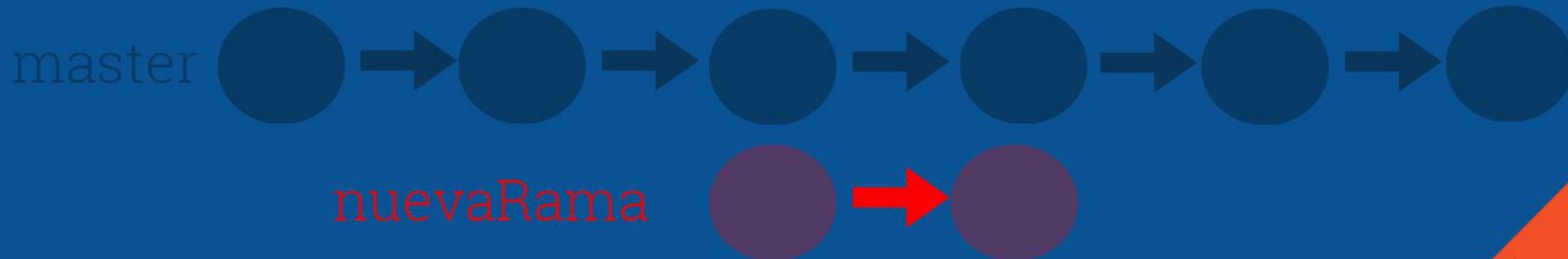
Ramas

⌚ ¿Cómo vemos las ramas?

`git branch`

⌚ ¿Cómo creamos una rama nueva?

`git branch nuevaRama`



⌚ ¿Cómo nos movemos de rama?

`git checkout nuevaRama`



Fusión

Cuando queremos tener los cambios que se fueron haciendo en paralelo sobre el proyecto.



1. Ir hacia la rama que absorverá.

git checkout master

2. Fusionar.

git merge nuevaRama



Tipos de Fusión

Cuando queremos tener todos los cambios que se fueron haciendo en paralelo sobre el proyecto.

Fast-Forward

El proceso es simple y automático. El proceso es largo y manual.

Sucede cuando se trabaja con archivos o líneas de código diferentes.

Manual Merge



Ramas

- Ⓐ ¿Cómo se elimina una rama?

git branch -D nombreRama

- Ⓐ Se suele borrar ramas una vez que ya se hizo el merge con la rama master.

- Ⓐ También podemos crear y movernos a una nueva rama de la siguiente forma:

git checkout -b nuevaRama





GitLab



GitLab

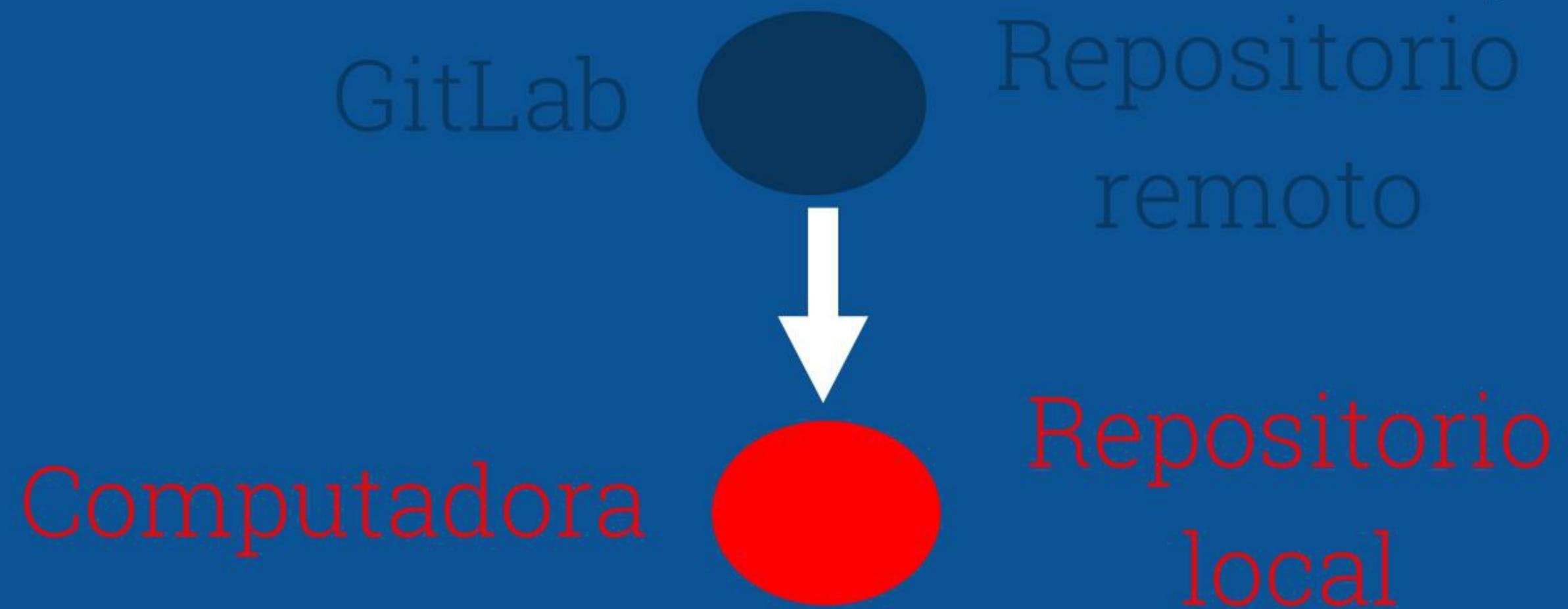
- Ⓐ GIT y GitLab no son lo mismo.
- Ⓐ GitLab es una plataforma donde se pueden guardar proyectos utilizando git para su gestión.
- Ⓐ Se necesita tener una cuenta, usaremos usuario@fing.edu.uy



>git clone

- ① Este comando nos sirve para clonar un proyecto (repositorio remoto) cuando queremos colaborar con el mismo.

git clone linkHTTP



SSH key

Ⓐ Sirve para poder conectarse a un servidor sin tener que indicar la password en cada intento de conexión.

ssh-keygen

Ⓐ Pedirá indicar un archivo para generar la clave, si no se indica ninguno lo hará en el home.

Ⓐ Luego solo hace falta indicar una password y la clave ssh será generada.



SSH key

↑Se puede ver la clave ssh generada ejecutando el siguiente comando:

```
cat .ssh/id_rsa.pub
```

↑Luego en GitLab en la configuración de usuario se podrá agregar la clave(el contenido del archivo) con un nombre asociado a la misma.

↑De esta forma GitLab ya conocerá la computadora.

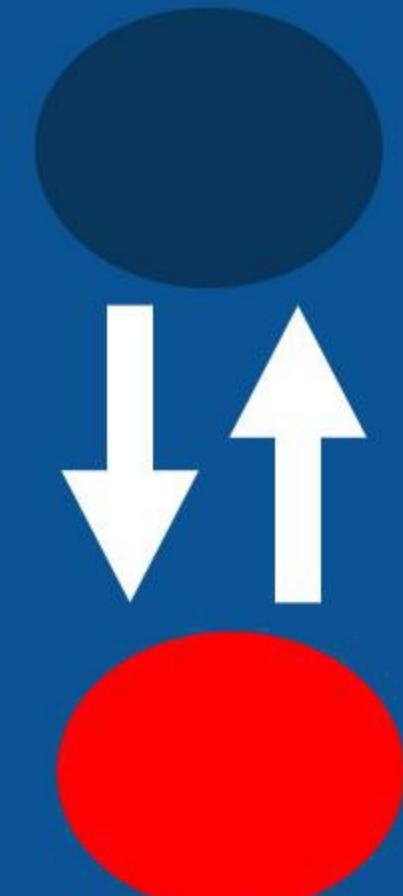


>git remote

- ①Este comando nos sirve para vincular un proyecto local con un proyecto remoto.

git remote add origin linkHTTP

GitLab



Repositorio
remoto

Computadora

Repositorio
local



>git remote

- Ⓐ Para establecer el vinculo hacemos lo siguiente:

git remote add origin linkHTTP

- Ⓐ Para ver el repositorio al que esta vinculado nuestro proyecto:

git remote -v

- Ⓐ Para eliminar el vínculo:

git remote remove origin



>git push

- Ⓐ Igualmente luego del git remote en GitLab no tenemos nada.
- Ⓐ Con este comando podremos enviar, subir todos nuestros cambios(commits) a GitLab, nuestro repositorio remoto:

git push origin master

- Ⓐ En este caso estamos enviando, subiendo la rama master.



>git tag

- Ⓐ TAGs serán puntos específicos en lo que sería la historia del proyecto.
- Ⓐ Son utilizados para indicar una versión del proyecto.
- Ⓐ Las tags son asignadas a los commits.
- Ⓐ Las utilizaremos para realizar las entregas.



>git tag

Ⓐ Para crear una tag en el último commit(head):

git tag -a v1.8 .m "Mensaje"

Ⓐ Para asignar una tag a un commit diferente al head:

git tag -a v1.8 .m "Mensaje" codigoSHA

Ⓐ GitLab aún no tiene dicha tag, para enviar una tag:

git push origin v1.8

Ⓐ Para enviar todas las tags:

git push origin --tags





Proyecto en
Equipo

Dinámica en equipo

- Ⓐ La dinámica es muy similar a como veníamos viendo.
- Ⓐ Habrá commits de más de una persona, con lo cuál habrá que trabajar un poco diferente.
- Ⓐ Haremos uso de dos comandos en esta nueva dinámica.

git fetch



git merge

git pull



Dinámica en equipo

- ⓐ Cuando se realiza el clone de un proyecto o conectamos un proyecto a un repositorio remoto se crea una rama oculta origin/master.
- ⓐ Es como un espejo de la rama master del repositorio remoto.
- ⓐ Para ver las ramas ocultas hacemos:

git branch -a



Dinámica en equipo

Repositorio remoto (GitLab)

master



origin/
master



master

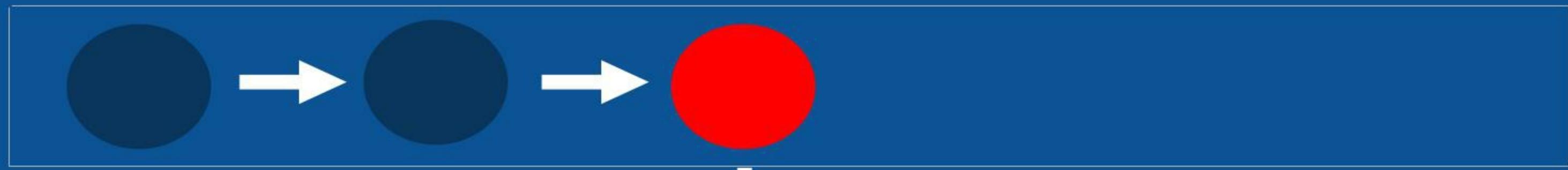


Repositorio local

Dinámica en equipo

Repositorio remoto (GitLab)

master



git fetch origin

origin/
master



master



(ahora se tienen los
cambios en la rama
origin/master oculta)

Repositorio local

Dinámica en equipo

Repositorio remoto (GitLab)

master



(ahora si tenemos los cambios en nuestra rama master)

origin/
master



master

git fetch origin

git merge origin/master

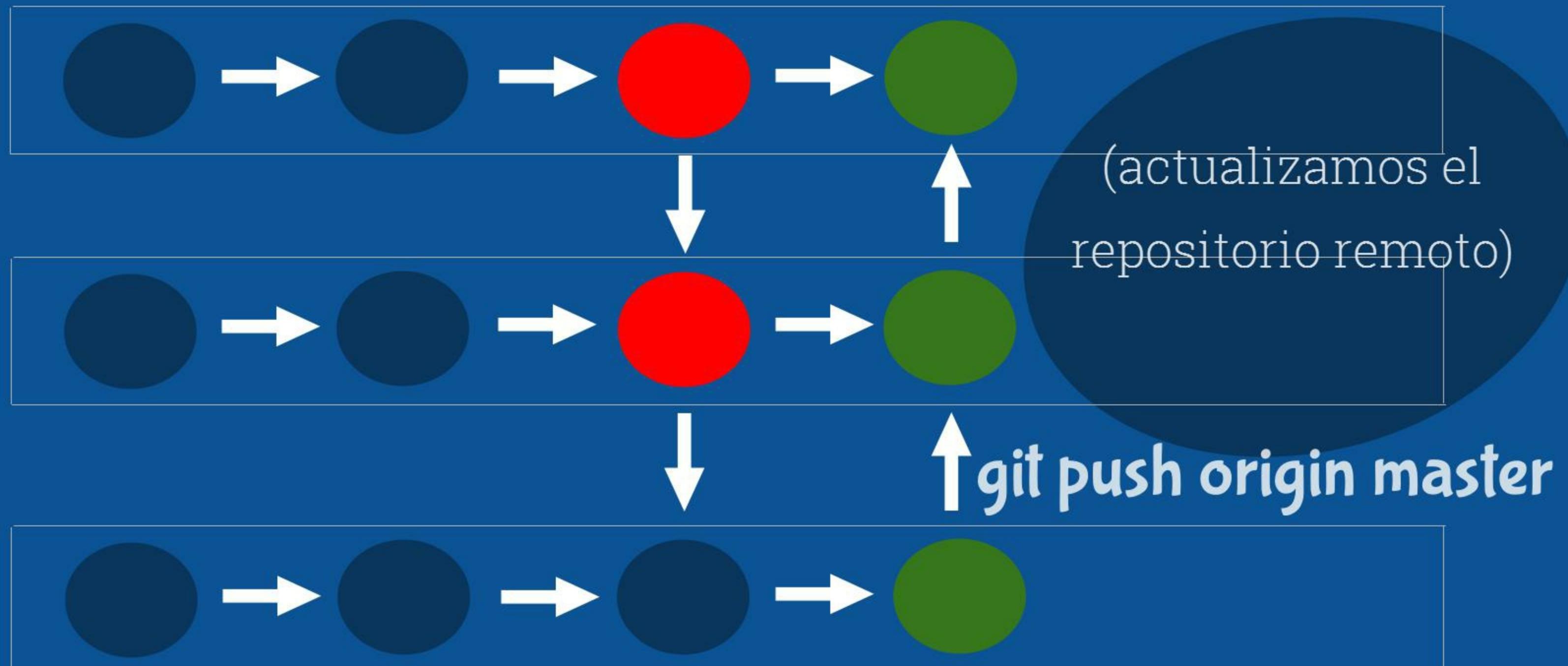


Repositorio local

Dinámica en equipo

Repositorio remoto (GitLab)

master



origin/
master

master

Repositorio local

Manual merge

- Ⓐ Esto sucede cuando existen conflictos al hacer un merge ya que los archivos son diferentes y GIT no sabe que versión del archivo mantener.
- Ⓐ Esto implica comunicar con el desarrollador que cuaso el conflicto y decidir manualmente que versión mantener..
- Ⓐ Con el comentario head se indicará el commit en el cual se intentó hacer el merge, y con el comentario origin/master la versión del master remoto.