



El futuro digital
es de todos

MinTIC



Ciclo 3: Desarrollo de Software

02

Sistemas de Control de Versiones

```
element* item = el->FirstChildElement(); item != 0; item = item->NextChildElement()
{
    boost::string el_name = item->Attribute( "name" );
    boost::string type = item->Attribute( "type" );
    ...
    std::string float_x = boost::lexical_cast<float>( item->Attribute( "x" ) );
    float y = boost::lexical_cast<float>( item->Attribute( "y" ) );
    float offset = boost::lexical_cast<float>( item->Attribute( "offset" ) );
    ...
    spriteDescList::iterator sp = sprite_descs.begin();
    while( sp != sprite_descs.end() && sp->name_ != spritename )
        ++sp;
}
```



El futuro digital
es de todos

MinTIC

Objetivo de Aprendizaje

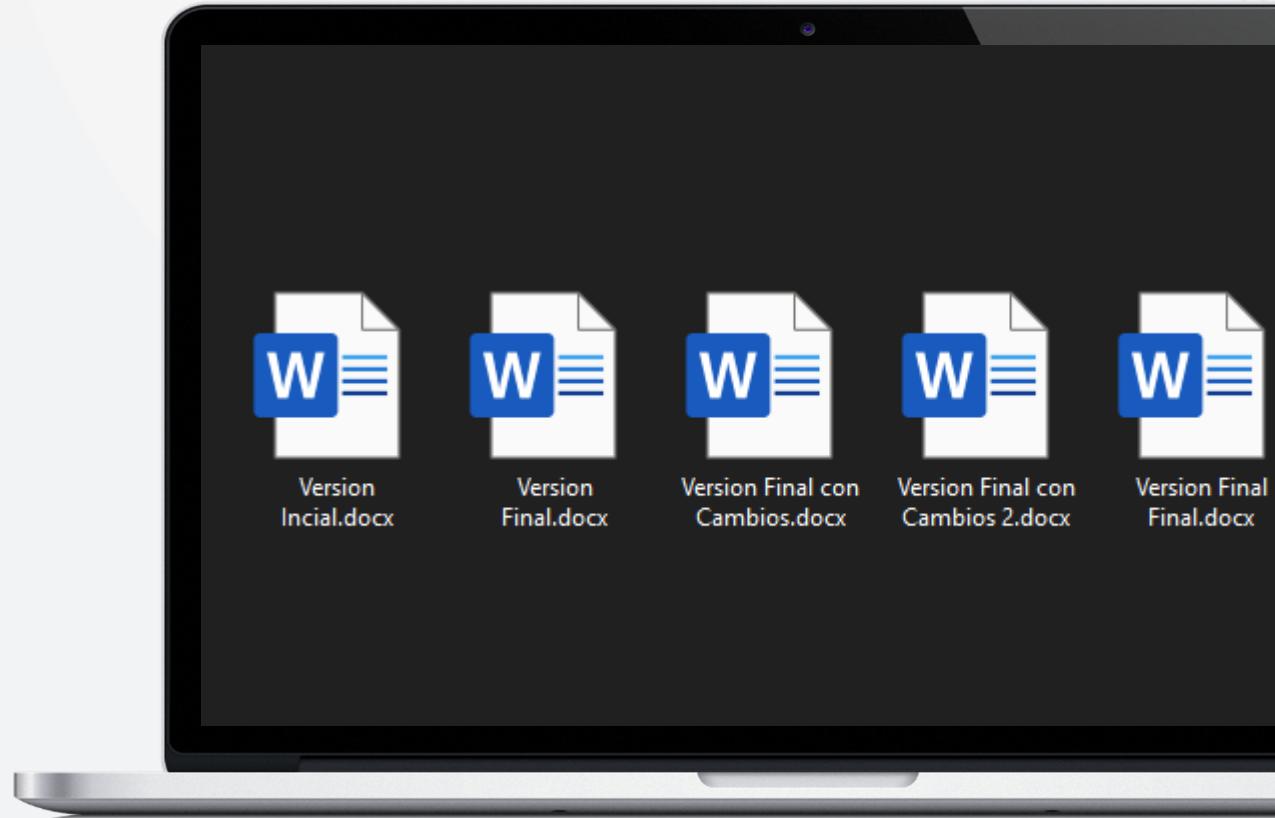
Identificar las principales características de los **sistemas de gestión de versiones**, en el contexto del **desarrollo de software**.



Control de Versiones

Los **proyectos de software** avanzan **agregando** cambios gradualmente y cada **cambio** significativo representa una nueva **versión** del proyecto. El **control** de estas versiones es un **problema** tedioso.

Un **ejemplo** clásico es el **manejo de documentos**.





Sistema de Control de Versiones

Un sistema de control de versiones (VCS - Version Control System) provee mecanismos que permiten manejar de forma ordenada los cambios que ocurren entre las distintas versiones de un proyecto, documento, código, entre otros.

El sistema de control de versiones más simple es el Ctrl + z y el Ctrl + y, el cual permite deshacer y redo cambios en un documento.

ctrl + z

ctrl + y



El futuro digital
es de todos

MinTIC

GIT

GIT es un sistema de control de versiones «open source», ideado por **Linus Torvalds**, que busca **manejar** de manera **eficiente** y simple el **código fuente** de proyectos de **software**.

GIT esta **enfocado** a **código fuente**, el cual es texto **plano**. Es por esto que GIT **no** suele **funcionar** bien con **archivos binarios**, como **imágenes**.



git



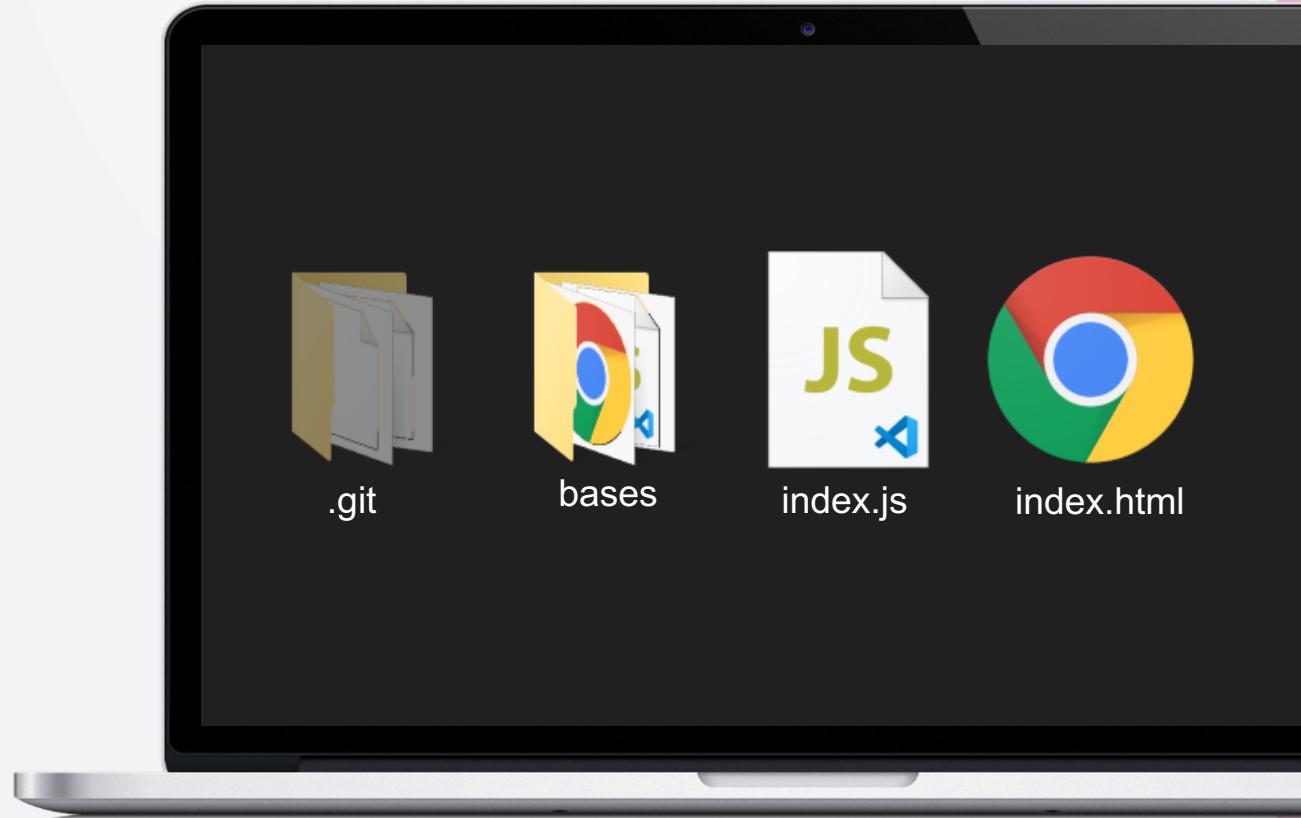
El futuro digital
es de todos

MinTIC

GIT: Repositorio

La **unidad** con la cual trabaja **GIT** se conoce como **repositorio**, el cual es una **directorio (carpeta)**, con el **código** fuente del proyecto, pero con unos **archivos adicionales** que le permiten a GIT realizar el control de versiones.

Estos **archivos** adicionales están en un **sub-directorio** llamado **.git**.



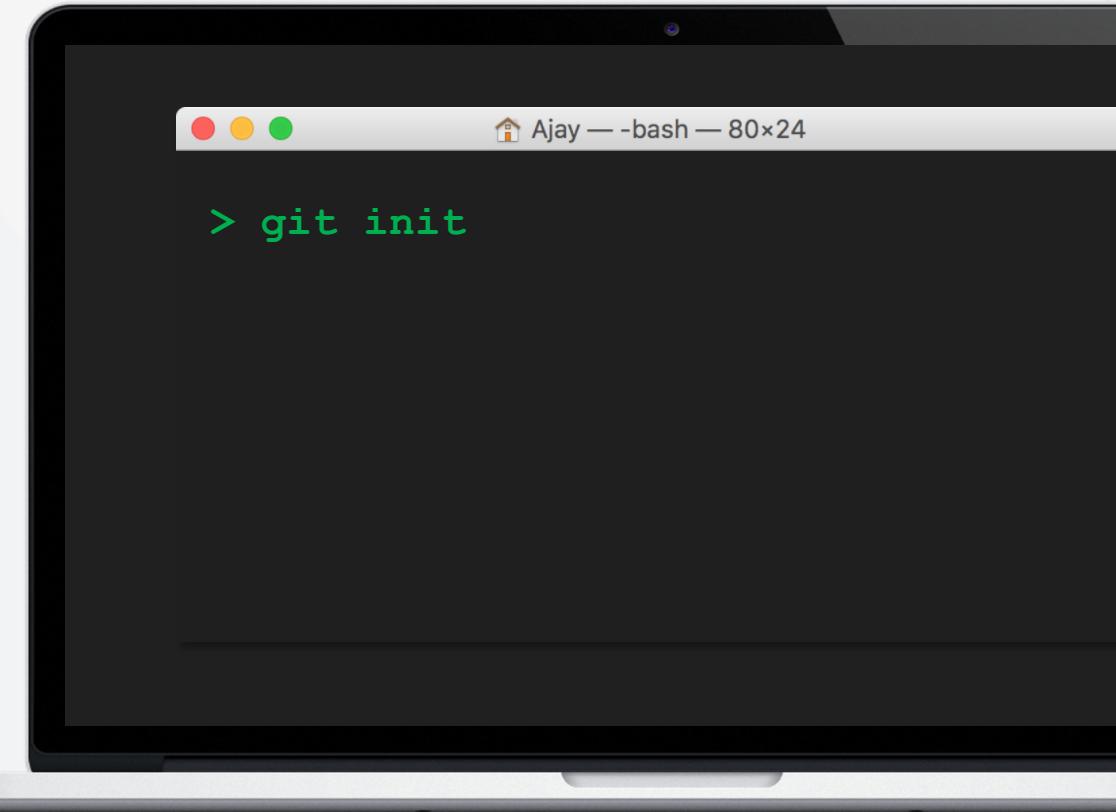


El futuro digital
es de todos

MinTIC

GIT: Repositorio

GIT provee el comando **init** para **inicializar** un **repositorio**, es decir, para pasar de una **carpeta** con el **código fuente**, a un **repositorio** sobre el cual **GIT** puede **trabajar**.

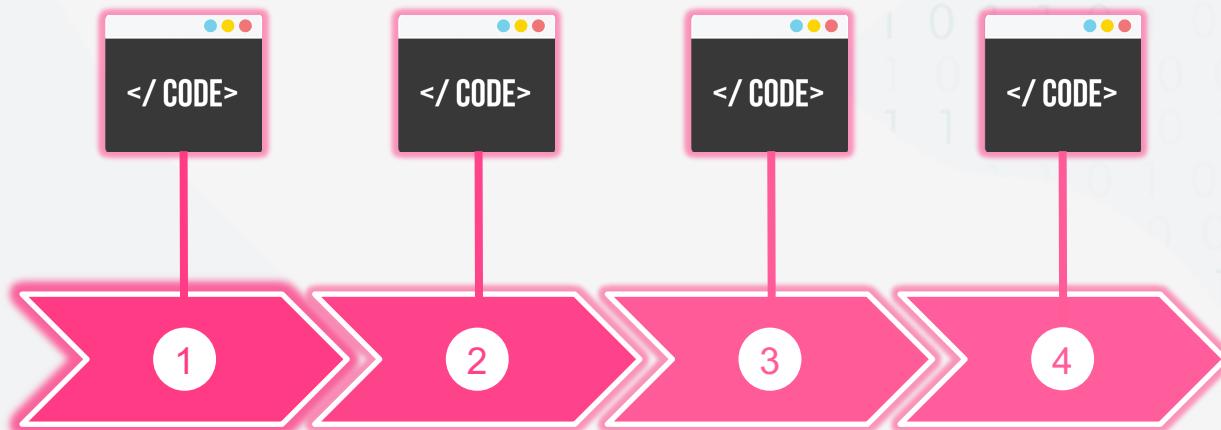




GIT: Commit

Cada **versión** de un **proyecto** se puede considerar un **punto** específico en el **tiempo**, de tal manera que la **historia** del **repositorio** es una **línea de tiempo**, sobre la cual se puede **viajar**.

GIT **maneja** de esta manera el **repositorio** y a cada uno de estos **puntos** se le conoce como **commit**.





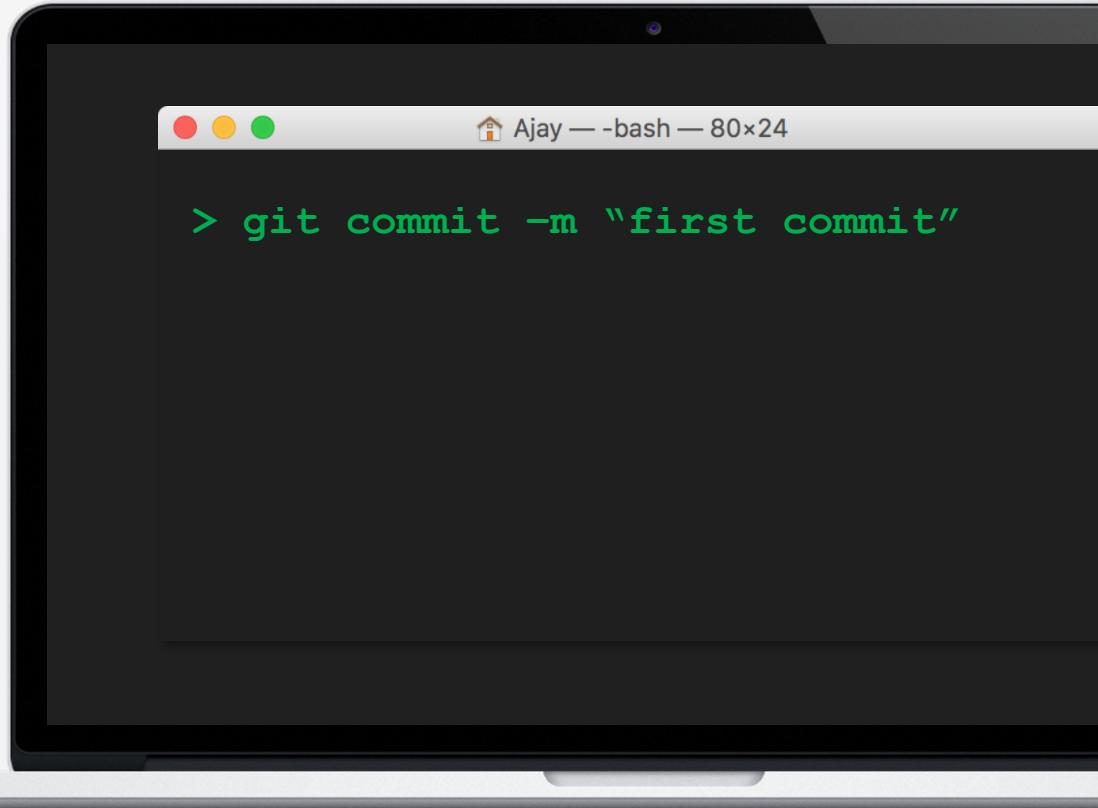
El futuro digital
es de todos

MinTIC

GIT: Commit

GIT provee el comando **commit** para crear un punto en la historia del repositorio.

Commit se usa cuando se tiene un grupo de cambios y se desea guardar ese estado del proyecto.

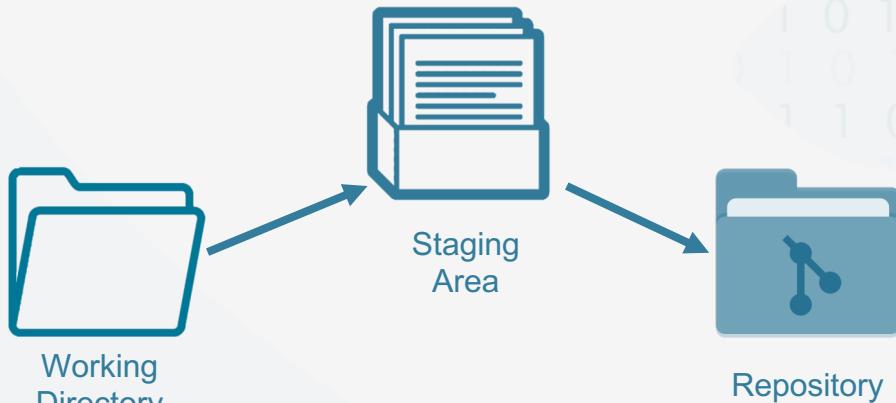




GIT: Staging Area

Commit permite registrar cambios, pero no se especifica que cambios registra.

Para esto, GIT provee el concepto de «Staging Area», el cual se puede ver como un escenario en el cual se tienen los cambios que se registrarán en el próximo commit.





GIT: Staging Area

Para ver el **estado** del «Staging Area», GIT provee el comando **status**, este muestra cuales **cambios** están **agregados y cuales no**.

Para **agregar** un cambio se usa el comando **add** y para **retirar** un **cambio** se utiliza el comando **rm**.

```
> git status  
> git add <file>  
> git rm <file>
```



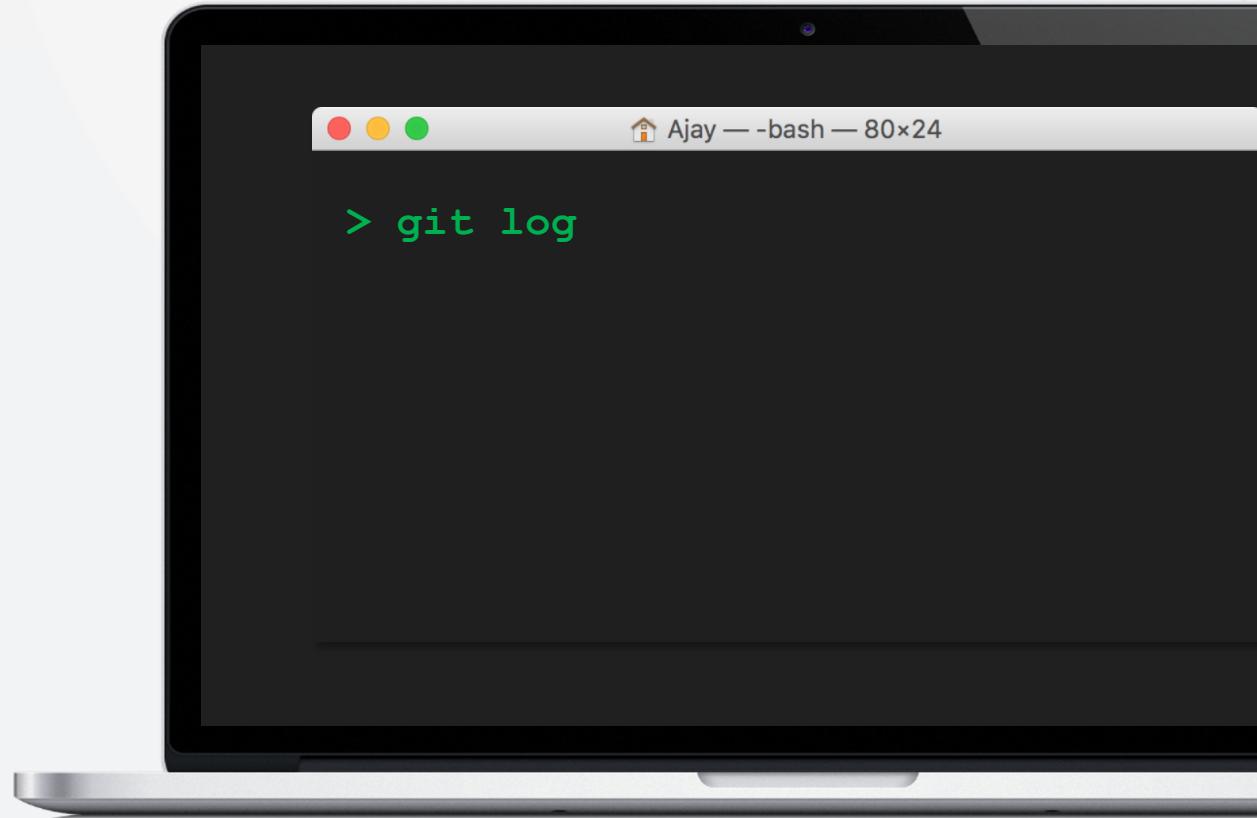
El futuro digital
es de todos

MinTIC

GIT: Log

GIT provee el comando **log** para visualizar la **historia** del **repositorio**, esta historia esta **dada** por los **commits** que se han **realizado**.

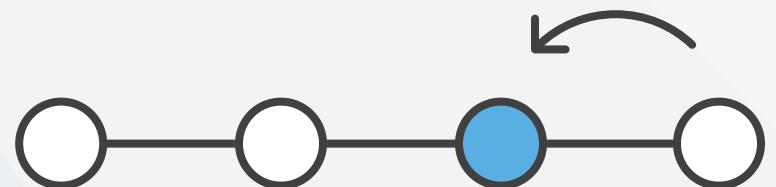
Cada **commit** se **identifica** con un **SHA**, el cual no es más que una **combinación** de **números** y **letras**.





GIT: Checkout

Una de las **características** más sorprendentes de **GIT**, es tener la capacidad de **viajar** entre **versiones**, es decir **visitar** un **commit** y ver el **estado** del código en ese punto.

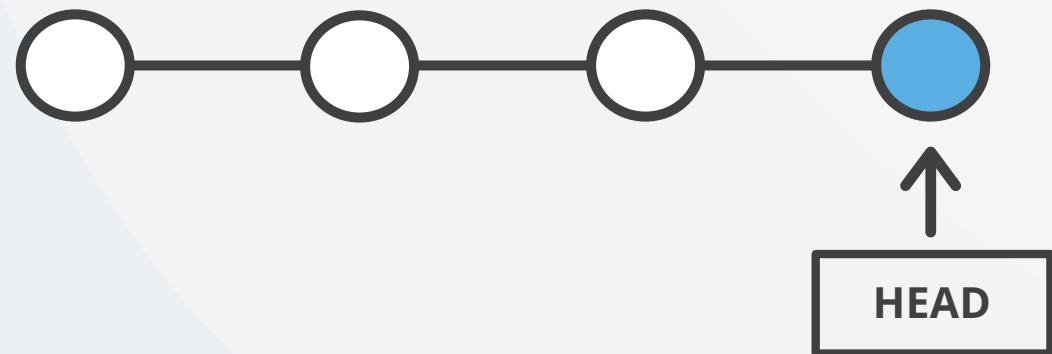


Para hacer esto posible, **GIT** provee el **comando checkout**. Para **viajar** a un **commit solo** hace falta conocer su **SHA**.



GIT: Head

En un **proyecto** siempre se **tiene** un **estado actual**, es decir el punto en el que se encuentra el proyecto.

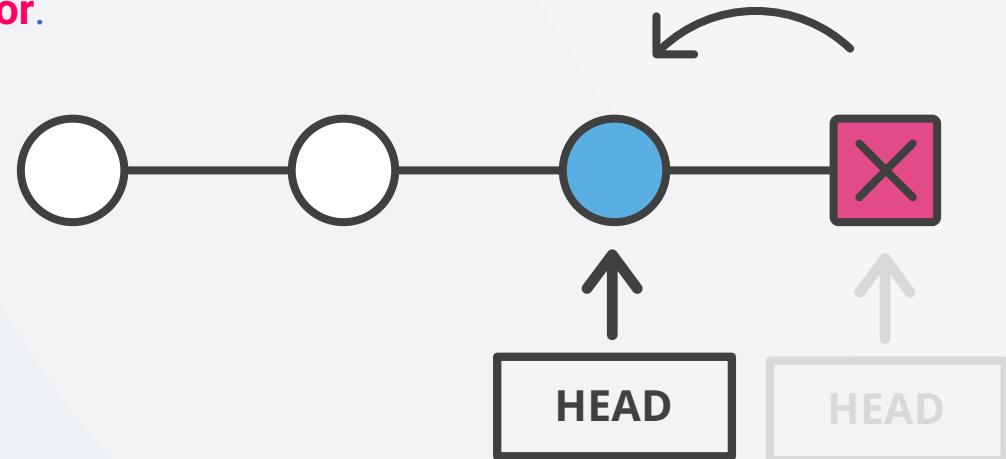


GIT utiliza un **identificador** llamado **HEAD** para **indicar** el **estado actual**, HEAD **apunta** a un **commit** específico.



GIT: Reset

Muchas veces en el **desarrollo** de un proyecto se requiere **regresar** a una versión **anterior**.



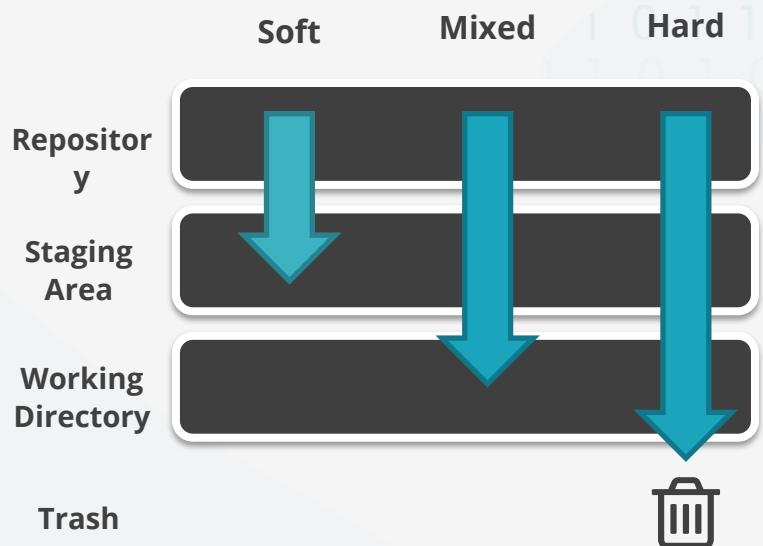
Para esto **GIT** provee el comando **reset**, el cual básicamente hace que el **HEAD** apunte a un **commit previo** al actual.



GIT: Reset

GIT reset tiene 3 variantes, que se diferencian en el tratamiento de los cambios sucedidos entre el commit previo y el actual:

- **Reset Soft:** los cambios permanecen agregados al «staging area».
- **Reset Hard:** los cambios son descartados.
- **Reset Mixed:** se conservan los cambios pero no están agregados al «staging area».



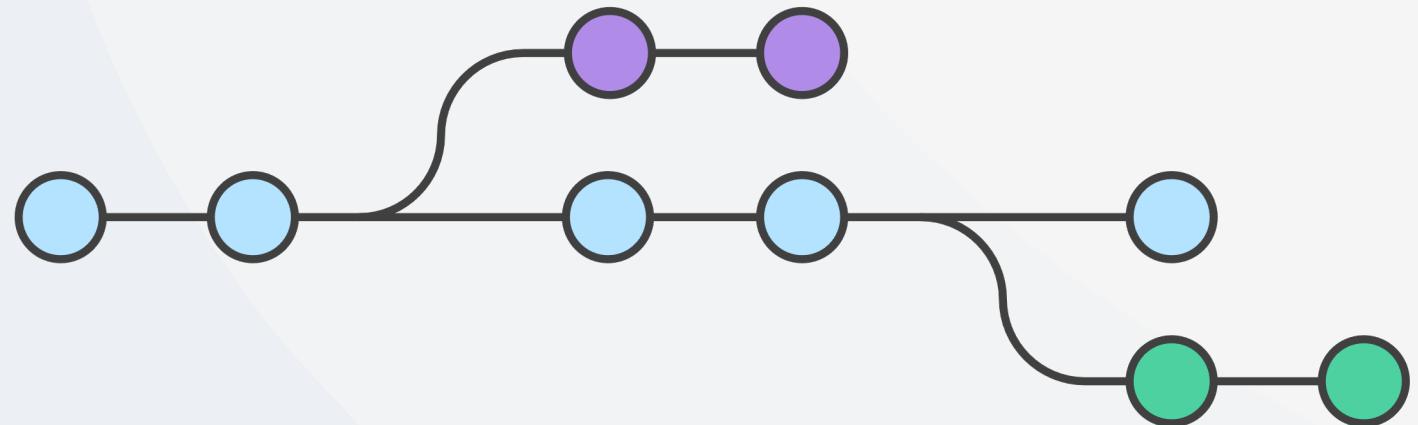


El futuro digital es de todos

MinTIC

GIT: Branches

GIT maneja la **historia** del repositorio como una **línea** de tiempo, pero **también permite** tener **múltiples líneas** que se **separan** en un **commit** específico y se pueden **unir** o **no** en un **commit** futuro.



A cada una de estas **líneas** se le **conoce** como **rama (branch)**. La rama por **defecto** se conoce como **master**.



El futuro digital
es de todos

MinTIC

GIT: Branches

Las **ramas** o branches son **muy útiles** al momento de **trabajar en equipo**, o trabajar en **varias características** del proyecto a la vez.

Las **ramas** permiten **dividir el trabajo** de manera organizada y **unirlo** de manera **fácil y rápida**.



[Imagen] Vector Free. (s. f.). Teamwork Illustration [Ilustración]. <https://vectorforfree.com/product/teamwork-illustration/>



El futuro digital
es de todos

MinTIC

GIT: Branches

Para **administrar** las **ramas** GIT provee el comando **branch**, este permite **crear, eliminar y cambiar** de rama.

Para **unir** dos **ramas** se usa el comando **merge**.

```
Ajay — -bash — 80x24
> git branch <name>
> git merge master
```



GIT y Distribución

Hasta el momento se ha **trabajado** con un **único repositorio**, pero en la vida **real no** ocurre esto, ya que trabajan **varias** personas en el **mismo proyecto** y **cada** uno tiene su **copia** del repositorio.

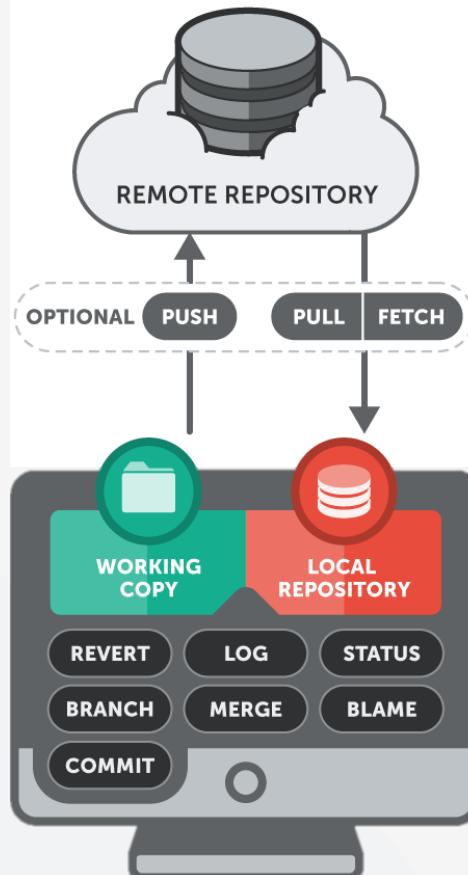
Por ello **GIT no** es un sistema **central**, sino uno **distribuido** que permite **sincronizar** los distintos **repositorios**.





GIT: Remote

En la **práctica** se trabaja con un repositorio **remoto** (almacenado en la nube), que es **clonado**, y sobre la **copia** resultante se **realizan cambios**, los cuales son **enviados** al repositorio **remoto** para **sincronizar** el trabajo, también se puede **actualizar** la **copia** del repositorio para **obtener** los **nuevos cambios**.





GIT: Remote

GIT ofrece el comando **clone** para **clonar** un **repositorio** de manera local.

Para **enviar** los **cambios** al remoto se utiliza la instrucción **push** y para **obtener** nuevos **cambios** se usa la instrucción **pull**.

Para **visualizar** y controlar el repositorio **remoto**, GIT ofrece el comando **remote**.

```
> git clone <url>
> git push origin
> git pull origin
> git remote -v
```



Plataformas

Los repositorios **remotos** se suelen **almacenar** en **plataformas de hosting** de repositorios. Estas plataformas son **basadas** en **GIT**, pero usualmente **ofrecen** características **adicionales** que no son definidas por GIT.

Algunas de las **plataformas** más conocidas **son**: GitHub, Bitbucket, GitLab, AWS CodeCommit, entre otras.

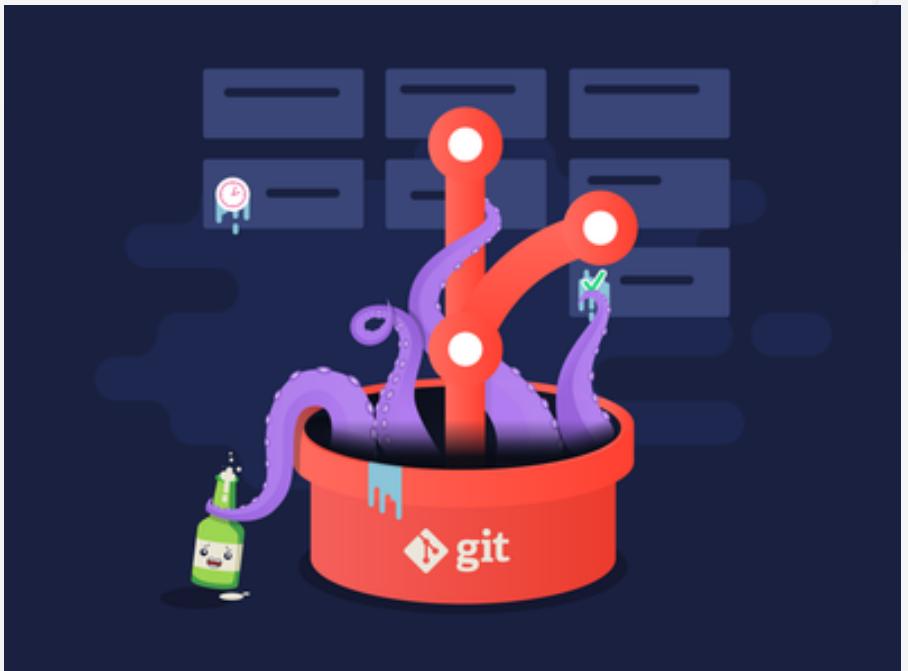




Si bien las **ramas** se usan para **trabajo en equipo**, o trabajar en **varias características** a la vez, **GIT** no define ningún **estándar** sobre como **usarlas**.

Pero existe una **modelo** de **ramas** llamado **Git Flow**, que define **reglas** sobre el **trabajo con ramas**, lo cual facilita su uso.

GIT Flow



[image] *Git branches*. (s. f.). [Ilustración]. <https://cdn.dribbble.com/users/36400/screenshots/4037272/attachments/925201/git-monster-wallpaper-21-9.png?compress=1&resize=400x300>



El futuro digital es de todos

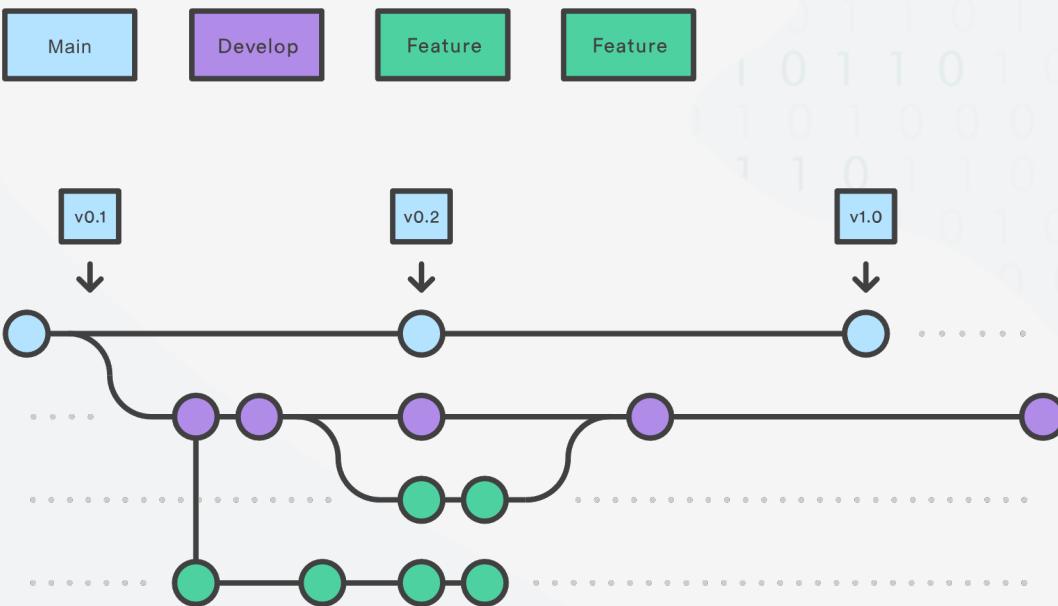
MinTIC

Se tiene una **rama principal** llamada **master** o **main**, la cual solo tiene **versiones estables** del proyecto.

Se tiene una **rama secundaria** llamada **develop** que contiene **todos los cambios** que ya han sido **terminados**.

Y se usa una **rama auxiliar** para **cada uno** de los **cambios** que actualmente se están **realizando**.

GIT Flow



[imagen] Atlassian. (s. f.-c). Gitflow Workflow | Atlassian Git Tutorial. Recuperado 22 de junio de 2021, de <https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow#:~:text=The%20overall%20flow%20of%20Gitflow,merged%20into%20the%20develop%20branch>



GIT Flow





Pull Request

Algunas **plataformas** como **GitHub** y **Bitbucket** tienen una característica llamada **Pull Request**.

Un **pull request** básicamente realiza el **mismo trabajo de merge** (unir dos ramas), pero lo hace de manera **formal y ordenada**.



[image] Pull request. (s. f.). [Ilustración]. [https://res.cloudinary.com/practicaldev/image/fetch/s--Bs2A9ezM--c_imagg_a_scale,f_auto,fl_progressive,h_900,q_auto,w_1600/https://3kllhk1ibq34qk6sp3bhoto1-wpengine.netdna-ssl.com/wp-content/uploads/bitbucket411-blog-1200x-branches2.png](https://res.cloudinary.com/practicaldev/image/fetch/s--Bs2A9ezM--c_imagg_a_scale,f_auto,fl_progressive,h_900,q_auto,w_1600/)



Pull Request

Un **pull request** inicia con una **solicitud** para **mezclar** dos ramas, esta solicitud pasa por ciertas **revisiones** que pueden sugerir **cambios** y finalmente es **aceptada o rechazada**.



[image] Pull request. (s. f.). [Ilustracion]. [https://res.cloudinary.com/practicaldev/image/fetch/s--Bs2A9ezM--c_imagg_a_scale,f_auto,fl_progressive,h_900,q_auto,w_1600/https://3kllhk1ibq34qk6sp3bhox1-wpengine.netdna-ssl.com/wp-content/uploads/bitbucket411-blog-1200x-branches2.png](https://res.cloudinary.com/practicaldev/image/fetch/s--Bs2A9ezM--c_imagg_a_scale,f_auto,fl_progressive,h_900,q_auto,w_1600/)