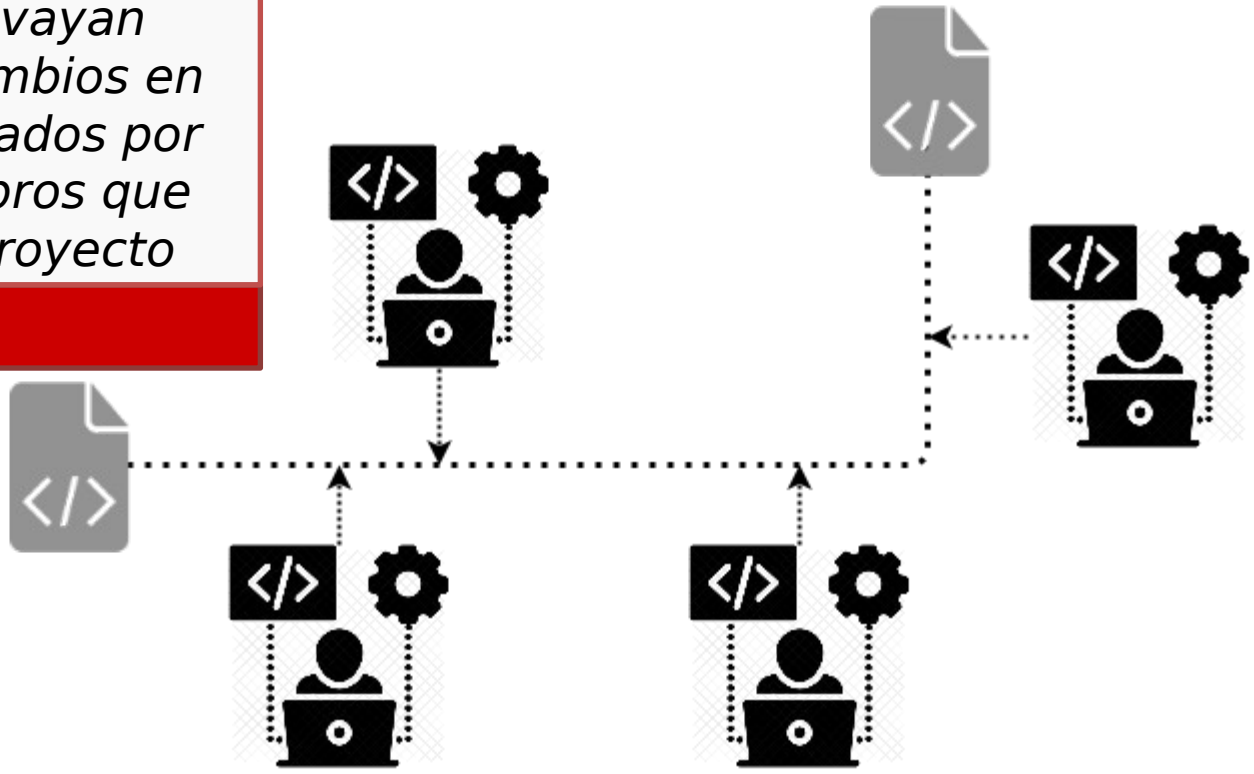

Universidad Industrial de Santander

Universidad Industrial de Santander

*Proyectos de desarrollo de un software es muy habitual que se vayan produciendo cambios en el código, realizados por todos los miembros que trabajan en el proyecto*

Necesidad

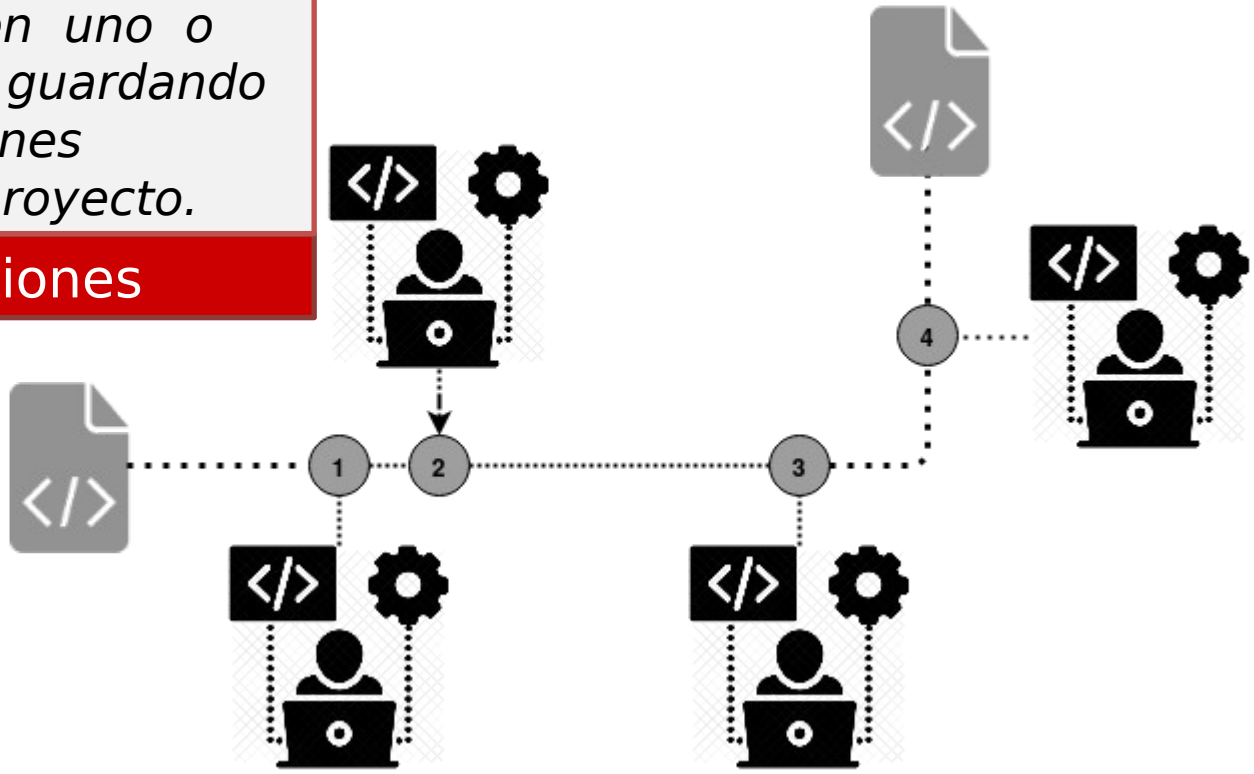


git

Universidad Industrial de Santander

*Un sistema de control de versiones es una herramienta capaz de registrar todos los cambios que se realizan en uno o más proyectos, guardando a su vez versiones anteriores del proyecto.*

## Control de Versiones




Universidad Industrial de Santander

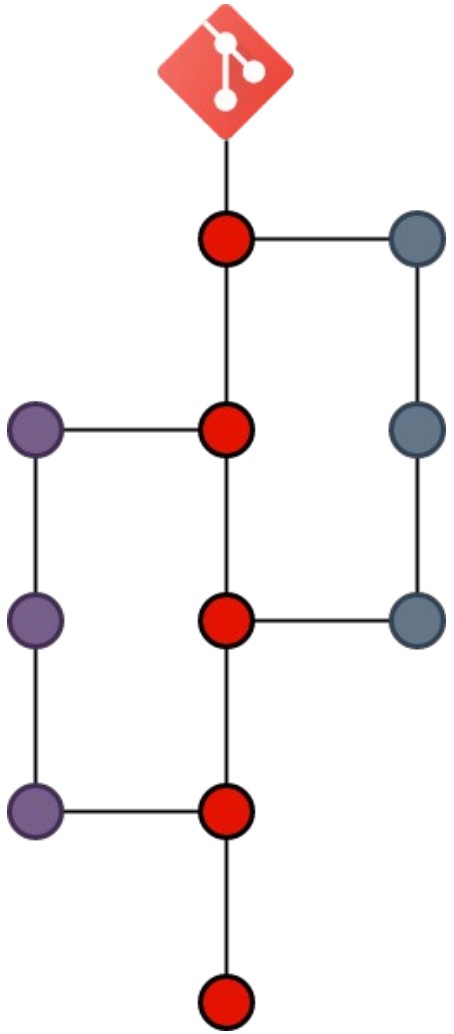
| | | |
|---|---|---|
| ***Sistemas de control de versiones locales*** *Sistema locales versionados por fecha de cambios* | ***Sistemas de control de versiones centralizados*** *Encontramos un único servidor que contiene todos los archivos versionados, y los usuarios que forman parte del proyecto se los pueden descargar desde ese servidor centralizado.* | ***Sistemas de control de versiones distribuidas*** *en estos sistemas no tenemos un único servidor que mantenga la información del proyecto, sino que cada usuario contiene una copia completa del proyecto de forma local.* |

Como se Clasifican ?

## Que es

*Git es un sistema de control de versiones de código abierto diseñado para gestionar todo proyecto con rapidez y eficiencia.*

*Git es un sistema de control de versiones distribuido, como habíamos indicado en el punto anterior, cuyo principal objetivo es ayudar en el desarrollo de cualquier tipo de aplicación manteniendo.*




Universidad Industrial de Santander

## Quien lo diseño

Linus Torvalds

## Donde lo descargo

http://git-scm.com/

# Que nos aporta?

Auditoría completa del código, sabiendo en todo momento quién ha tocado algo, cuándo y qué

Control sobre cómo ha ido cambiando nuestro proyecto con el paso del tiempo.

Volver uno o más pasos hacia atrás de forma rápida.

Control de versiones del proyecto por medio de etiquetas.

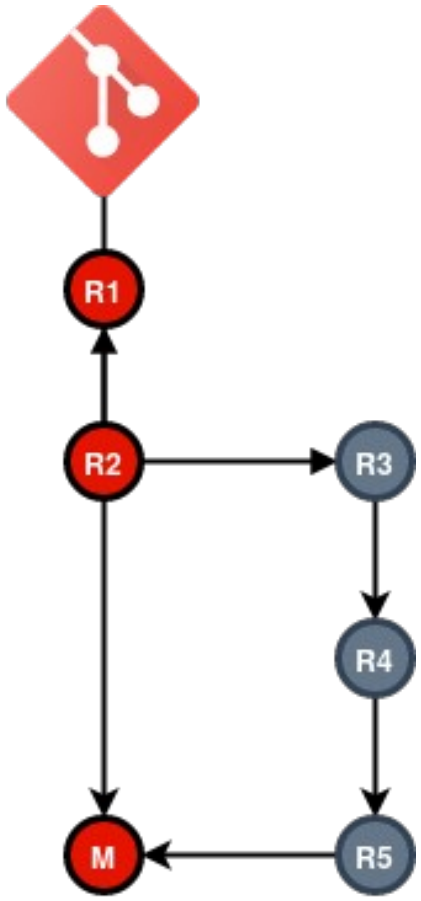
Seguridad, ya que todas las estructuras internas de datos irán cifradas con el algoritmo

## Workflow

Creas una rama de Master (Maestro),

haces tu trabajo, y cuando has finalizado

haces el merge (mezcla) en el Master.




Merge en la Master

Universidad Industrial de Santander

## Estado principales

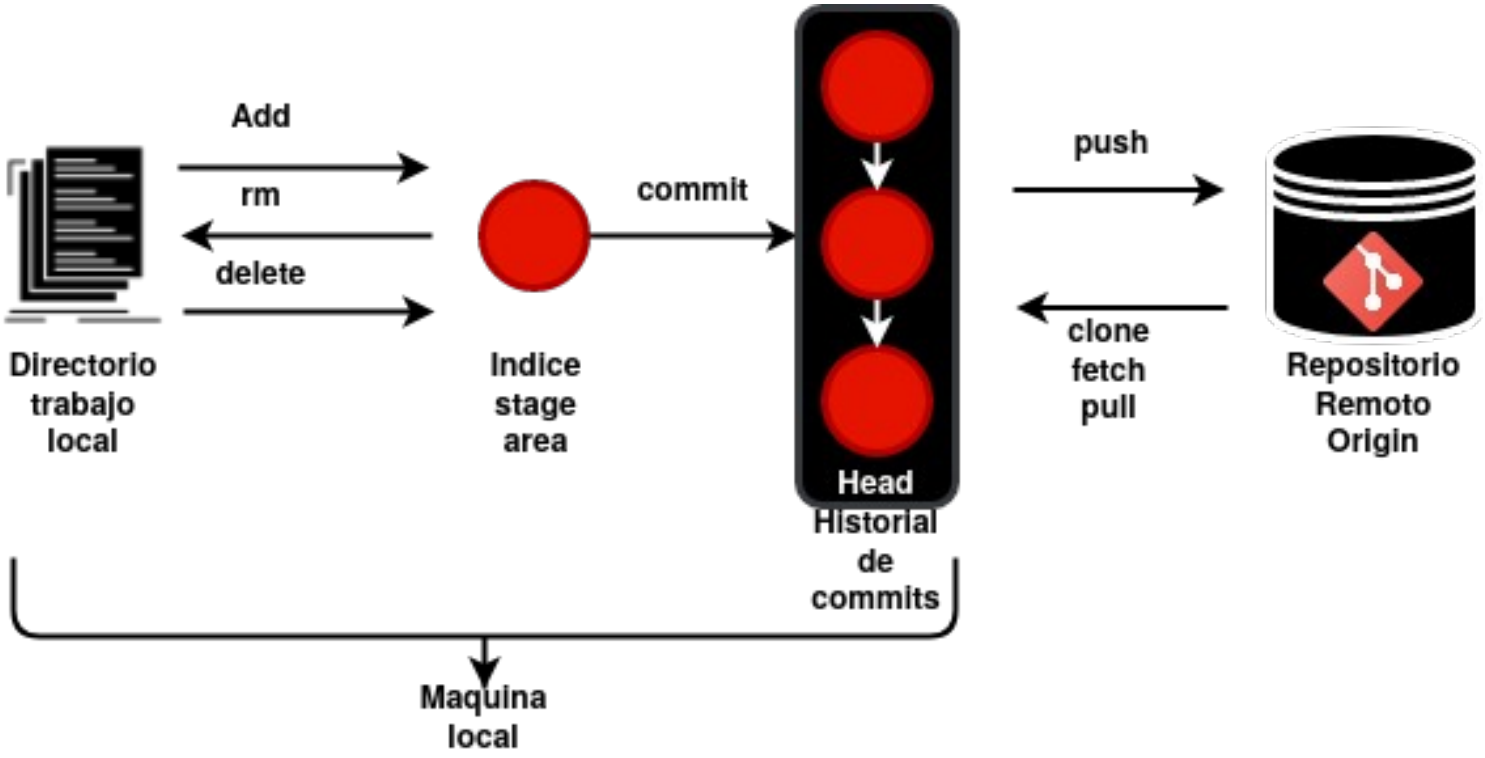
Git tiene tres estados principales:

- confirmado (committed)
- modificado (modified)
- preparado (staged)

## Secciones principales

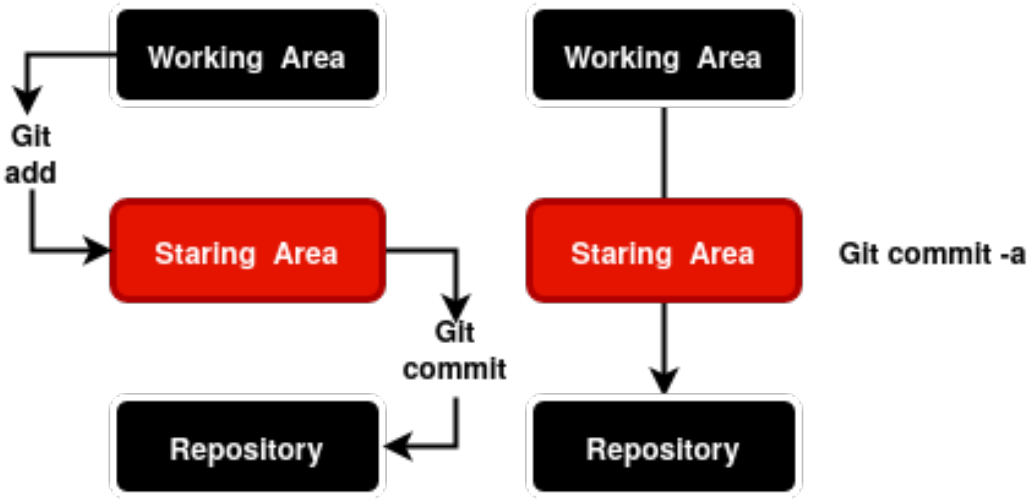
Git tiene tres Secciones principales:

- Directorio de Git (Git directory)
- Directorio de trabajo (working directory)
- Área de preparación (staging area)


Add
rm
delete
commit
push
clone
fetch
pull
Directorio
trabajo
local
Indice
stage
area
Head
Historial
de
commits
Repositorio
Remoto
Origin
Maquina
local

## Staging area

área que guardará temporalmente nuestros archivos (cuando ejecutemos un comando especial para eso) y nos permitirá, más adelante, guardar estos cambios en el repositorio

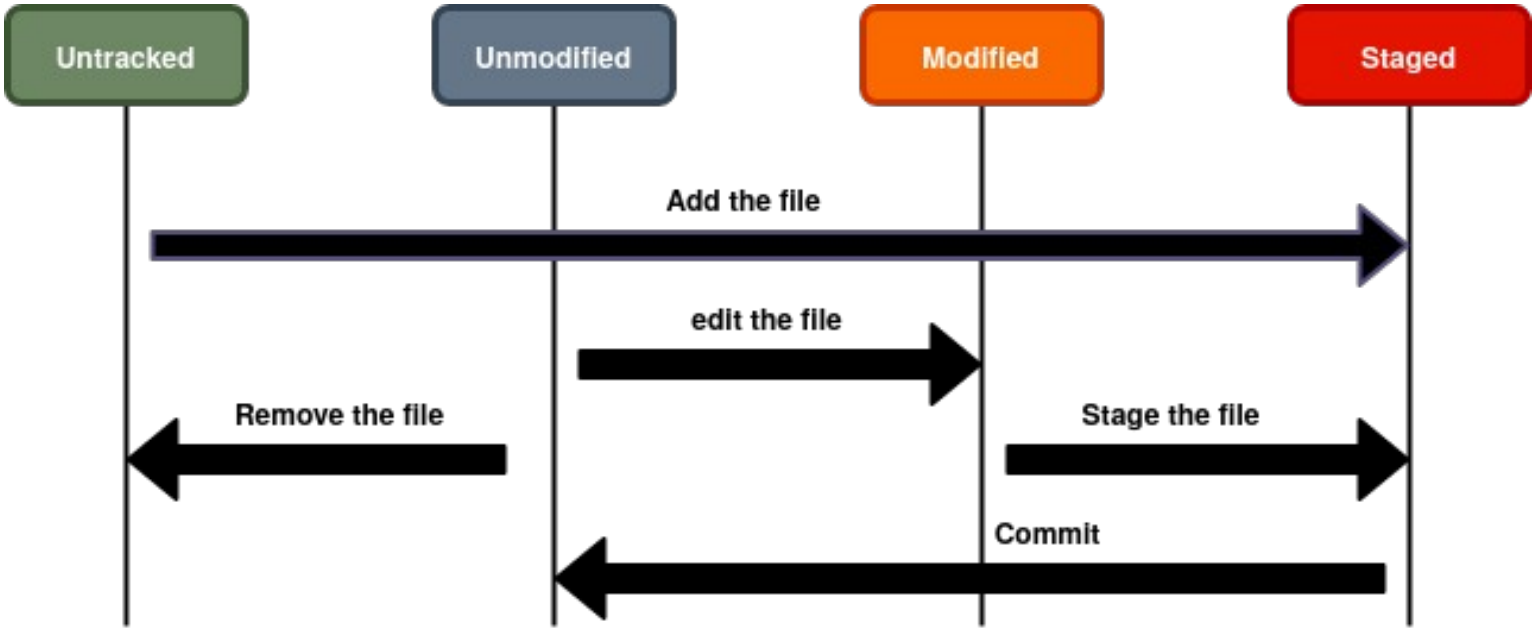


Universidad Industrial de Santander

## estados de los archivos


| | |
|---|---|
| **Tracked** : archivos que no tienen cambios pendientes y sus últimas actualizaciones han sido guardadas en el repositorio com los comandos git add y git commit. | **Staging**: archivos que han sido afectados por el comando git add, aunque no sus últimos cambios. Git ya sabe de la existencia de estos últimos cambios, pero todavía no han sido guardados en el repositorio porque falta ejecutar el comando git commit. |
| **Unstaged**: Arquivosque no han sido afectados por el comando git add ni mucho menos por git commit. | **Untracked**: son archivos que NO viven dentro de Git, solo en el disco duro. Nunca han sido afectados por git add, así que Git no tiene registros de su existencia. |

# Ciclo de vida de los archivos



Universidad Industrial de Santander

## Guia On line

https://learngitbranching.js.org/


Universidad Industrial de Santander

## Crear  Reposito

Nuevo
git init

checkout a un repositorio
Local
git clone /path/to/repository

Remoto
git clone
username@host:/path/to/repository

## Add y commit

git add <filename>
git add .

git commit -m "Commit message"

## Subir cambios

git push origin master

## Branch

Crea una nueva
git checkout -b feature_x

Volver a  la master
git checkout master

borra la branch
git branch -d feature_x

Branch a tu repositorio remoto
git push origin <branch>

Universidad Industrial de Santander

## Comandos

**Ayuda:**
**Git help**
*comando* ó **git** *comando* --help
Muestra la ayuda para ese comando

**Creación de un repositorio:**
**git init**
Crea un repositorio en el directorio actual
**git clone** *url*
Clona un repositorio remoto dentro de un directorio

## Comandos

**Operaciones :**
**git add** *path*
Adiciona un archivo o un directorio de manera recursiva
**git rm** *ruta*
Remueve un archivo o directorio del árbol de trabajo
-f : Fuerza la eliminación de un archivo del repositorio
**git mv** *origen destino*
Mueve el archivo o directorio a una nueva ruta
-f : Sobre-escribe los archivos existentes en la ruta destino
**git checkout** *[rev] archivo*
Recupera un archivo desde la rama o revisión actual
-f : Sobre-escribe los cambios locales no guardados

Universidad Industrial de Santander

## Comandos

**Trabajando:**
**git status**
Imprime un reporte del estado actual del árbol de trabajo local
**git diff** *[ruta]*
Muestra la diferencia entre los cambios en el árbol de trabajo local
**git diff** HEAD *ruta*
Muestra las diferencias entre los cambios registrados y los no registrados
**git add** *path*
Selecciona el archivo para que sea incluido en el próximo commit
**git reset** HEAD *ruta*
Marca el archivo para que no sea incluido en el próximo commit

## Comandos

**Trabajando:**
**git commit**
Realiza el commit de los archivos que han sido registrados (con git-add)
-a : Automáticamente registra todos los archivos modificados
**git reset** --soft HEAD^
Deshace commit & conserva los cambios en el árbol de trabajo local
**git reset** --hard HEAD^
Restablece el árbol de trabajo local a la versión del ultimo commit
**git clean**
Elimina archivos desconocidos del árbol de trabajo local

Universidad Industrial de Santander

## Comandos

**Repositorios remotos:**
**git fetch** *[remote]*
Trae los cambios desde un repositorio remoto
**git pull** *[remote]*
Descarga y guarda los cambios realizados desde un repositorio remoto
**git push** *[remote]*
Guarda los cambios en un repositorio remoto
**git remote**
Lista los repositorios remotos
**git remote add** *remote url*
Añade un repositorio remoto a la lista de repositorios registrados

## Comandos

**Ramas:**
**git checkout** *rama*
Cambia el árbol de trabajo local a la rama indicada
-b *rama* : Crea la rama antes de cambiar el árbol de trabajo local a dicha rama
**git branch**
Lista las ramas locales
**git branch** -f *rama rev*
Sobre-escribe la rama existente y comienza desde la revisión
**git merge** *rama*
Guarda los cambios desde la rama

Universidad Industrial de Santander

## Comandos

**Exportando e importando:**

**git apply** - < *archivo*
Aplica el parche desde consola (stdin)

**git format-patch** *desde [..hasta]*
Formatea un parche con un mensaje de log y un reporte de diferencias (diffstat)

**git archive** *rev > archivo*
Exporta resumen de la revisión (snapshot) a un archivo

--prefix=*dir/* : Anida todos los archivos del snapshot en el directorio

--format=*[tar|zip]* : Especifica el formato de archivo a utilizar: *tar* or *zip*

Universidad Industrial de Santander

## Herramienta graficas

| | |
|---|---|
| **GitHub Desktop:**<br>extensión de su flujo de trabajo de GitHub. La herramienta le proporciona una interfaz de usuario que le permite administrar su código. Puede iniciar sesión usando sus credenciales de GitHub y comenzar a trabajar en sus repositorios. | **GitKraken:**<br>es un cliente Git desarrollado independientemente es compatible con GitHub , Bitbucket y Gitlab.<br>La interfaz gráfica es intuitiva para la administración de su proyecto.<br>Se Debe registrarte en GitKraken antes de usar esta herramienta. |
| **SmartGit:**<br>SmartGit es un gran cliente profesional de Git que es gratuito para organizaciones no comerciales | **SourceTree:**<br>es un cliente Git desarrollado independientemente es compatible con GitHub , Bitbucket y Gitlab.<br>La interfaz gráfica es intuitiva para la administración de su proyecto.<br>Se Debe registrarte en GitKraken antes de usar esta herramienta. |

Universidad Industrial de Santander

## Taller

- Crear una cuenta en github.com
- (El ejercicio se va a hacer en parejas, uno de los estudiantes debe crear un nuevo repositorio usando la interface web de github)
- Enseguida deben descargar e instalar el software cliente de git (bash y GUI) https://git-scm.com/downloads
- Con el software instalado deben realizar un git clone (preferiblemente usando la línea de comandos bash) del repositorio creado.

Universidad Industrial de Santander

## Taller

- Cada uno de los estudiantes debe asumir un rol A y B. El estudiante A debe crear un archivo de texto (Hola1.java, un Hola Mundo por ejemplo). Enseguida el estudiante A hace un git commit con un mensaje de “commit primer archivo”

- Ambos estudiantes deben realizar un git log y ver que sus repositorios difieren, el estudiante A tiene dos commits mientras que el estudiante B un solo commit.

- Enseguida el estudiante A hace un git push en ese momento le va a pedir que configure email y usuario, en el resultado del comando dice como se hace, una vez lo configure lo deja hacer el push. Nuevamente ambos estudiantes hacen git log


Universidad Industrial de Santander

## Taller

- Luego el estudiante B hace un git pull y trae los cambios y ambos hacen git log y en ese momento sus repositorios están idénticos

- Luego ambos estudiantes crean dos archivos nuevos, A crea Hola2.java y B crea Hola3.java, ambos hacen git commit y luego ambos hacen git log y ven las diferencias

- El estudiante B hace push primero y el estudiante A hace push que le debe fallar, en ese momento el estudiante A debe hacer un pull y se realizara un merge. Una vez logrado B también hace pull y ambos miran su historia.

Universidad Industrial de Santander

## Taller

- Finalmente ambos modifican el mismo archivo Hola3.java hacen commit y push, uno de los dos tendrá problemas y tendrá que hacer un merge manual

- La última parte de la clase es hacer las mismas operaciones con el cliente gráfico y luego les puede decir que descarguen otro cliente. También que vean lo que ha pasado en la interface de github.com, ahí se pueden ver los commits, quien los hizo, los branches y demás.

Universidad Industrial de Santander

## Herramientas

Tortoisegit
https://tortoisegit.org/
SourceTree
https://www.sourcetreeapp.com/
Egit
https://www.eclipse.org/egit/
Smartgit
http://www.syntevo.com/smartgit/