INFORMATORIO

UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL RESISTENCIA

UNCAUS
UNIVERSIDAD NACIONAL DEL CHACO AUSTRAL

Globant

POLO IT /CHACO

Subsecretaría de Empleo
Chaco Gobierno de todos

Ministerio de Producción, Industria y Empleo
Chaco Gobierno de todos

CHACO
Gobierno de todos

INFORMATORIO

Versionado

Git y GitHub

UNIVERSIDAD TECNOLÓGICA NACIONAL FACULTAD REGIONAL RESISTENCIA
UNCAUS UNIVERSIDAD NACIONAL DEL CHACO AUSTRAL
Globant
POLO IT /CHACO
Subsecretaría de Empleo Chaco Gobierno de todos
Ministerio de Producción, Industria y Empleo Chaco Gobierno de todos
CHACO Gobierno de todos

INFORMATORIO

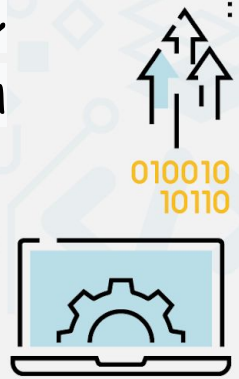
¿Qué es el control de versiones?

Se llama control de versiones a los métodos y herramientas disponibles para controlar todo lo referente a los cambios en el tiempo de un archivo.

INFORMATORIO

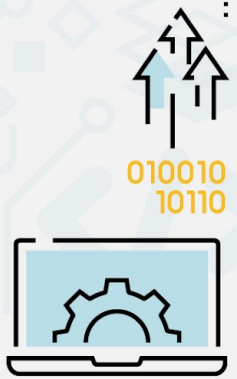
¿Qué es un sistema de control de versiones?

un sistema de control de versiones (VCS) 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, versiones a las que podemos acudir en caso de que no funcionen de la forma correcta.



Ventajas del uso de un VCS

Aunque se puede desarrollar software sin utilizar ningún control de versiones, hacerlo somete al proyecto a un gran riesgo que ningún equipo profesional debería aceptar. Así que la pregunta no es si utilizar el control de versiones, sino qué sistema de control de versiones usar.



Sistemas de Control de Versiones

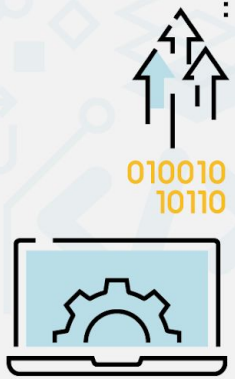
Algunos de los más conocidos son:

- Git
- CVS
- Subversion
- Mercurial
- Apache Subversion (SVN):
- Monotone
- Baazar

¿Qué es Git?

Git es un sistema de control de versiones de código abierto, actualmente es el sistema de control de versiones más extendido.

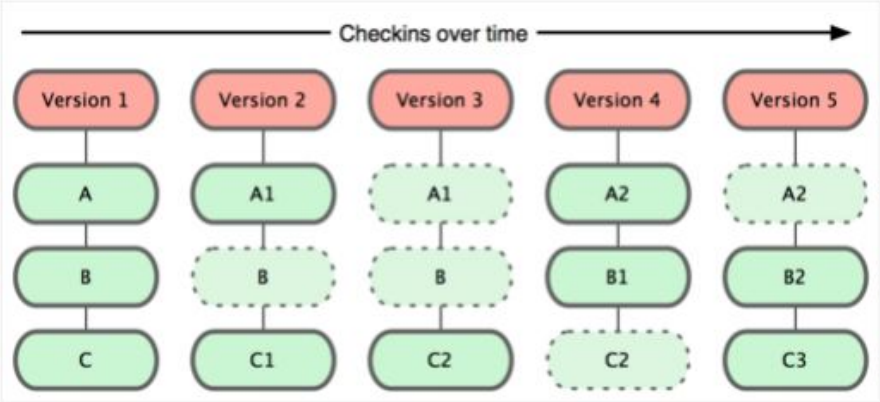
Git es el sistema de control de versiones más utilizado del mundo. Según la encuesta entre los desarrolladores de Stack Overflow, más de el 87% de los desarrolladores usan Git. el historial de cambios completo del proyecto. Esto aumenta significativamente su rendimiento.

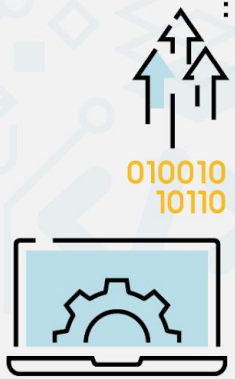


¿Qué es Git?

Se diferencia del resto en el modo en que modela sus datos.

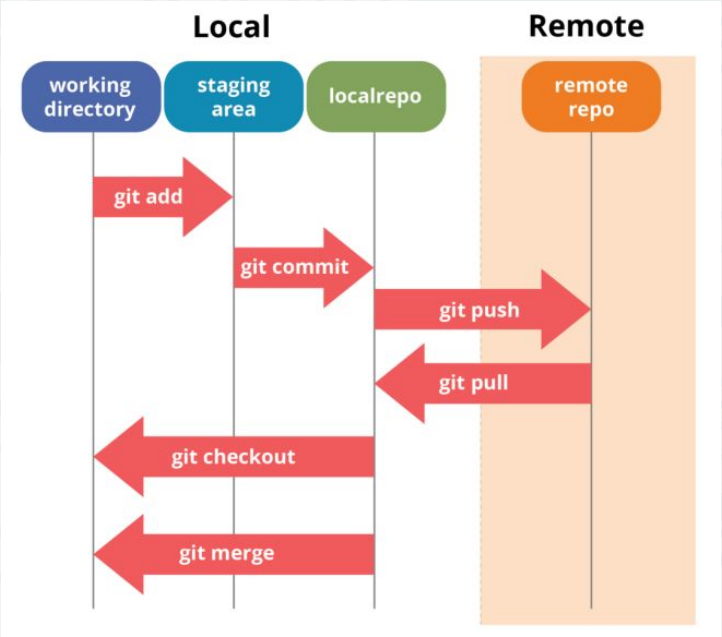
La mayoría de los demás sistemas almacenan la información como una lista de cambios en los archivos, mientras que Git modela sus datos más como un conjunto de instantáneas de un mini sistema de archivos.





INFORMATORIO

Los estados y las áreas de trabajo de GIT



Configurando Git

Descargar Git:

https://git-scm.com/downloads

Configurando Git

Configurar el usuario de git y la email

```
git config --global user.email "tu email"

git config --global user.name "tu username"
```

Ver la información que se encuentra configurada

```
git config --list
```

Comenzamos a trabajar con Git

- Crea un directorio nuevo.
- Para crear un nuevo repositorio en se usa la orden:

```
git init
```

Este comando crea una carpeta oculta llamada .git que contiene la base de datos donde se registran los cambios en el repositorio.

INFORMATORIO

Comprobar el estado del repositorio

- Crear un archivo
- Agregar una o dos líneas de código para imprimir en pantalla un "Hola Mundo"
- Con la orden git status podemos ver en qué estado se encuentran los archivos de nuestro repositorio.

```
git status
```

Añadir y confirmar cambios

- Con la orden git add indicamos a git que prepare los cambios para que sean almacenados.

```
git add <nombre del archivo>
```

- Con la orden git commit confirmamos los cambios definitivamente, lo que hace que se guarden permanentemente en nuestro repositorio.

```
git commit -m <mensaje del commit>
```

Ver el historial de cambios

- Con la orden git log podemos ver todos los cambios que hemos hecho.

```
git log
```

- Ver el historial en forma de gráficos

```
git log --graph
```

- También es posible ver versiones abreviadas o limitadas, dependiendo de los parámetros:

```
git log --oneline --since='5 minutes ago' --max-count=2
```

INFORMATORIO

Recuperando versiones anteriores

- Cada cambio es etiquetado por un hash, para poder regresar a ese estado del proyecto se usa la orden git checkout.

```
git checkout <hash del commit>
```

- Para volver a la última versión de la rama master usamos git checkout indicando el nombre de la rama.

```
git checkout master
```

Visualizar cambios

- Para ver los cambios que se han realizado en el código usamos la orden:

```
git diff
```

- Se puede indicar un parámetro y dará los cambios entre la versión indicada y el estado actual. O para comparar dos versiones entre sí, se indica la más antigua y la más nueva.

```
git diff <versión más  antigua> <versión más nueva>
```

INFORMATORIO

Deshaciendo cambios antes de la fase de staging

- Si tenemos un archivo con modificaciones, que aún no han sido agregados al stag, y queremos revertirlas, debemos hacer un checkout a ese archivo.

```
git checkout <nombre del archivo>
```

Deshaciendo cambios antes del commit

- Si tenem… que ya se encuentran … agregaron a un commit, y que… almacenados en el stag debemos hacer …:

Se utiliza la palabra HEAD para referirse siempre al último commit. Para referirse al penúltimo commit se utiliza HEAD~1, al antepenúltimo HEAD~2, etc.

```
git reset HEAD <nombre del archivo>
```

Deshaciendo commits no deseados

- Si hemos hecho un commit y nos hemos equivocado, podemos deshacerlo con la orden git revert.

```
git revert HEAD --no-edit
```

--no-edit hace que la reversión no abra el editor del sistema usado para editar el mensaje de confirmación antes de realizar la confirmación

Borrar commits de una rama

Si además de querer revertir un commit, queremos eliminar la huella en el historial de cambios hay que usar la orden git reset.

- Reset soft si queremos borrar el commit pero mantener los cambios en los archivos hechos en el commit y los posteriores.

```
git reset --soft <hash del commit>
```

Borrar commits de una rama

Si además de querer revertir un commit, queremos eliminar la huella en el historial de cambios hay que usar la orden git reset.

- Reset hard si queremos borrar el commit y además deshacer los cambios en los archivos hechos en el commit y los posteriores.

```
git reset --hard <hash del commit>
```

Borrar commits de una rama

De todas maneras, reset es una operación delicada, que debe evitarse, sobre todo cuando se trabaja en repositorios compartidos.

Modificar un commit

- Uno de las acciones más comunes a deshacer es cuando confirmas un cambio antes de tiempo y olvidas agregar algún archivo, o te equivocas en el mensaje de confirmación.
- Para incluir archivos a un commit, sin editar el mensaje, usamos luego de agregar los archivos:

```
git commit --amend --no-edit
```

- Para editar el mensaje del commit usamos:

```
git commit --amend -m "<nuevo mensaje>"
```

Trabajando con ramas

Cuando vamos a trabajar en una nueva funcionalidad, es conveniente hacerlo en una nueva rama, para no modificar la rama principal y correr el riesgo de dejarla inestable. Aunque la orden para manejar ramas es git branch podemos usar también git checkout.

Trabajando con ramas

- Ver las ramas existentes

```
git branch
```

- Crea una nueva rama

```
git branch <nombre de la rama>
```

- Cambiarse a la rama nueva

```
git checkout <nombre de la rama>
```

Trabajando con ramas

- O de la forma mas rapida:

```
git checkout -b <nombre de la rama>
```

Fusionar una rama

una vez que tengas el trabajo aislado en una rama, es muy probable que desees incorporarlo en la rama principal. Podés combinar cualquiera de las ramas en su rama actual con el comando git merge.

Resolver conflictos

Al realizar una fusión ocurrirá que la operación se realizó con éxito (Fast-Forward) o que la operación presentó unos conflictos (Manual Merge).

- Manual Merge: Es lo que ocurre cuando en ambas ramas se realizan modificaciones afectando las mismas líneas de código. En este caso Git te pedirá que elijas con cual fragmento de código (cambio) te quedarás, rega la nueva y se reemplaza la vieja.

Resolver conflictos

Git insertará marcas de conflicto estándares, en los archivos cuando hay un conflicto de merge. Ahora nos toca a nosotros resolverlos.

Debemos seleccionar qué cambios mantener y una vez hecho estos cambios podremos agregar a un nuevo commit quien tendra la fusion de ambas ramas.

INFORMATORIO

Fusionar una rama - Merge

1. Cambiarse a la rama master

```
git checkout master
```

2. Agregar los cambios hechos en otra rama

```
git merge <nombre de la rama>
```

Fusionar una rama - Rebase

- También se puede fusionar ramas con el comando

```
git rebase <nombre de la rama>
```

- En el caso de que en la rama se hayan hecho varios commits el rebase irá uno a uno recorriendo los commits para que puedas seleccionar qué cambios dejar. Para ir recorriendo esos commit ejecutar:

```
git rebase --continue
```

Este comando se tiene que ejecutar luego de haber solucionado los conflictos del anterior.

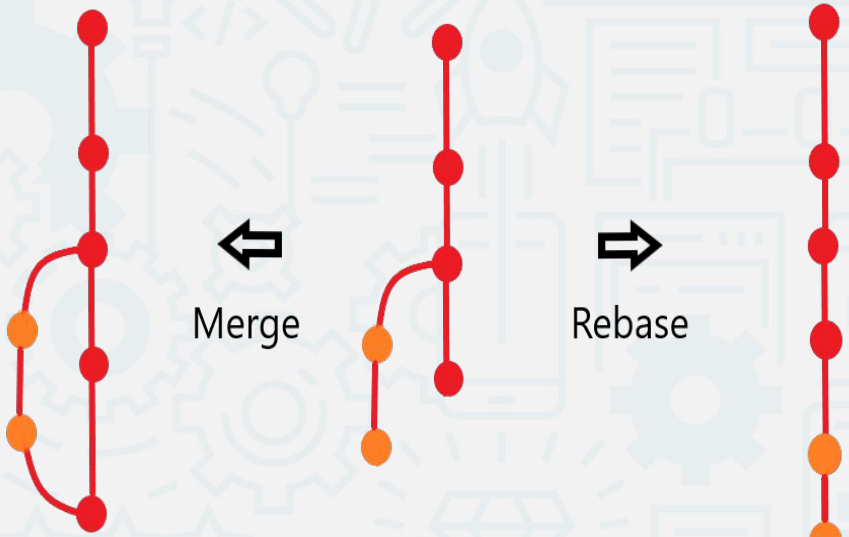
Diferencias entre Merge y Rebase

Merge hace explícita la integración de ramas y además mantiene contiguos todos los commits de una rama, pero dificulta la legibilidad del repositorio cuando se está trabajando con varias ramas.

Por otro lado, rebase mantiene una historia lineal del proyecto, pero es más propenso a cometer errores al reescribir la historia. La regla de oro es no hacer rebase sobre commits públicos, o al menos no hacerlo sobre commits sobre los que cualquier miembro del equipo haya basado su trabajo. Además, con rebase perdemos la trazabilidad de cuando se integró una rama.

INFORMATORIO

Diferencias entre Merge y Rebase



Git y el trabajo remoto

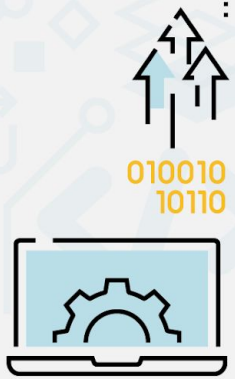
La otra característica de Git, que unida a las ramas, facilita la colaboración entre distintos usuarios en un proyecto son los repositorios remotos.

Git permite la creación de una copia del repositorio en un servidor git en internet. La principal ventaja de tener una copia remota del repositorio, a parte de servir como copia de seguridad, es que otros usuarios pueden acceder a ella y hacer también cambios. Existen muchos proveedores de alojamiento para repositorios Git pero el más usado es GitHub.

¿Qué es GitHub?

GitHub es el mayor proveedor de alojamiento de repositorios Git. Aunque no sea parte directa del proyecto de código abierto de Git, es muy probable que durante tu uso profesional de Git necesites interactuar con GitHub en algún momento.

La principal ventaja de GitHub es que permite albergar un número ilimitado de repositorios tanto públicos como privados.



Configurando GitHub

- Crearse una cuenta en GitHub

```
https://github.com/
```

- Crear un repositorio remoto
- Enlazarlo con el repositorio local

```
git remote add origin <url del repositorio>
```

Git y el trabajo remoto

Cuando se añade un repositorio remoto a un repositorio, Git seguirá también los cambios del repositorio remoto de manera que se pueden descargar los cambios del repositorio remoto al local y se pueden subir los cambios del repositorio local al remoto.

INFORMATORIO

Trabajar con repositorios remotos

- Modificar archivos del repositorio y ejecutar

```
git add .
```

- Agregarlos a un nuevo commit

```
git commit -m "<mensaje del commit>"
```

- Actualizar la rama con los cambios hechos en el repositorio local

```
git push origin master
```

INFORMATORIO

Trabajar con repositorios remotos

- Iniciar sesión en GitHub.

INFORMATORIO

Conflictos en repositorios remotos

- Hacer cambios de forma remota desde la web de GitHub.
- Hacer cambios en alguno de los archivos, agregarlos a stage y pushearlos a rama remota.

```
git add .

git commit -m <mensaje del commit>

git push origin master
```

INFORMATORIO

Trabajar con ramas en el repositorio remoto

- Crear una rama.

```
git branch <nombre de la rama>
```

- Moverse a la rama recién creada.

```
git checkout <nombre de la rama>
```

- Hacer cambios en los archivos y agregarlos a la zona de stage.

```
git add .
```

Trabajar con repositorios remotos

- Agregar los cambios a un nuevo commit.

```
git commit -m <mensaje del commit>
```

- Hacer un push de tus cambios al repositorio remoto.

```
git push origin <nombre de la rama>
```

Trabajar con repositorios remotos

- Hacer cambios en alguno de los archivos, agregarlos a stage y pushearlos a rama remota.

```
git add .

git commit -m <mensaje del commit>

git push origin <nombre de la rama>
```

- Crear un pull request

Clonando repositorios remotos

- Clonar un repositorio remoto.

```
git clone <URL del repositorio>
```

- Hacer cambios en los archivos y ejecutar.

```
git diff
```

INFORMATORIO

Trabajar con repositorios remotos

1. Crear una rama con tu nombre

```
git branch <nombre>
```

2. Cambiarse a esa rama

```
git checkout <nombre>
```

INFORMATORIO

Trabajar con repositorios remotos

1. Hacer cambios locales, agregarlos al stage

```
git add .

git commit -m <mensaje del commit>
```

2. Modificar el comentario del comit

```
git commit --amend -m <Nuevo mensaje>
```

Eliminar ramas locales

1. Crear una nueva rama con el comando

```
git checkout -b <nombre de la rama>
```

2. Volver a la rama principal

```
git checkout master
```

3. Ejecutar el comando

```
git branch -d <nombre de la rama>
```

Eliminar ramas remotas

- En el caso de querer eliminar la rama del repositorio remoto, la sintaxis será la siguiente:

```
git push origin :<nombre de la rama>
```

De esta forma, la rama también desaparecerá del servidor.

Trabajar con repositorios remotos

1. Por otro lado, si quieres deshacer todos los cambios locales y commits, puedes traer la última versión del servidor y apuntar a tu copia local principal de esta forma

```
git fetch origin

git reset --hard origin/master
```

1. Crear un archivo .gitignore

```
* ignora los archivos terminados en .a

*.a

* pero no lib.a, aun cuando había ignorado los archivos terminados en .a en la línea anterior

!lib.a

* ignora unicamente el archivo TODO de la raiz, no subdir/TODO

/TODO

* ignora todos los archivos del directorio build/

build/

* ignora doc/notes.txt, pero no este: doc/server/arch.txt

doc/*.txt

* ignora todos los archivos .txt del directorio doc/

doc/**/*.txt
```

INFORMATORIO

Enlaces útiles

https://help.github.com/articles/creating-a-pull-request/

https://codigofacilito.com/articulos/buenas-practicas-en-commits-de-git

https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet

https://git-scm.com/book/en/v1/ o https://git-scm.com/book/es/v1/ (en español)