Curso profesional de Git y GItHub

Contenido

[¿Qué es Git? 3](#__RefHeading___Toc277_1488830239)

[Características importantes de Git 3](#__RefHeading___Toc279_1488830239)

[Directorios de Git 4](#__RefHeading___Toc281_1488830239)

[¿Qué es GitHub? 5](#__RefHeading___Toc283_1488830239)

[Herramientas propias de GitHub 6](#__RefHeading___Toc285_1488830239)

[¿Qué es un sistema de control de versiones? 7](#__RefHeading___Toc287_1488830239)

[Comandos Git 8](#__RefHeading___Toc289_1488830239)

[Git init 8](#__RefHeading___Toc291_1488830239)

[Git status 8](#__RefHeading___Toc293_1488830239)

[Git add 8](#__RefHeading___Toc295_1488830239)

[Git rm –cached 8](#__RefHeading___Toc297_1488830239)

[Git Commit 8](#__RefHeading___Toc299_1488830239)

[Git Tag 8](#__RefHeading___Toc301_1488830239)

[Git diff 9](#__RefHeading___Toc303_1488830239)

[Git reset 9](#__RefHeading___Toc305_1488830239)

[Git reset –soft 9](#__RefHeading___Toc307_1488830239)

[Git reset –mixed 9](#__RefHeading___Toc309_1488830239)

[Git reset –hard 10](#__RefHeading___Toc311_1488830239)

[Las ramas en git 10](#__RefHeading___Toc313_1488830239)

# ¿Qué es Git?

Git es un sistema de control de versiones que originalmente fue diseñado para operar en un entorno Linux. Actualmente Git es multiplataforma, es decir, que ahora ya no solamente es compatible con Linux, sino también con MacOS y Windows.

Un sistema de control de versiones es el conjunto de herramientas que tenemos disponibles para poder gestionar de una forma eficiente las distintas versiones de nuestros proyectos.

## Características importantes de Git

* Git se diferencia de otros sistemas de control de versiones en la forma en la que modela sus datos. Usualmente otros sistemas almacenan la información en una lista de cambios en archivos, mientras Git lo hace como un conjunto de archivos.
* La integridad con la que cuenta es bastante seria. No existen cambios, corrupción en archivos o cualquier alteración sin que Git lo sepa. Esto funciona gracias a una verificación con la que cuenta mediante un checksum, que es básicamente una suma de comprobación que se hace previo al almacenamiento de información.
* Casi todo en Git es local. Es difícil que se necesiten recursos o información externos, basta con los recursos locales con los que cuenta.

Git cuenta con 3 estados en los que podemos localizar nuestros archivos:

* Committed. Es la parte en la que nuestra información está segura alojada en nuestras bases de datos.
* Modified. En esta parte hemos realizado cambios en nuestros archivos, pero aún no se ven reflejados en nuestra base de datos.
* Staged. En esta parte marcamos nuestros archivos modificados dejándolos listos para confirmarlos.

## Directorios de Git

Esta es quizá la parte más importante de Git, ya que es el lugar donde se almacenan los metadatos y las bases de datos para nuestros proyectos, y es justamente lo que se copia cuando clonamos de un ordenador a otro los archivos.

Dentro de estos directorios es posible que no solamente tengamos nuestra información útil, sino también ciertos archivos de caché innecesarios, para ello es posible ignorar directorios sin afectar lo necesario de nuestros proyectos.

Una buena práctica para tener todo en orden y eliminar lo innecesario es crear un archivo .gitignore donde podamos incluir esos directorios que no dan valor a nuestros proyectos y así mantener solamente lo necesario.

¿Habías utilizado o escuchado de Git antes de este curso?

Cuéntanos en el sistema de discusiones cómo lo conociste y qué otras características te animaron a preferirlo.

# ¿Qué es GitHub?

GItHub es un servicio de alojamiento que ofrece a los desarrolladores repositorios de software usando el sistema de control de versiones, Git.

Existen varios aspectos que hacen que GitHub sea una opción eficiente para el control y gestión de tus proyectos.

Estos son algunos de ellos:

* GitHub permite que alojemos proyectos en repositorios de forma gratuita.
* Los repositorios son públicos, sin embargo Github tiene una forma de pago que te permite alojar tus proyectos de forma privada.
* Permite que puedas compartir tus proyectos de una forma mucho más fácil.
* Te permite colaborar para mejorar los proyectos de otros y a otros mejorar o aportar a los tuyos.
* Ayuda reducir significativamente los errores humanos, a tener un mejor mantenimiento de distintos entornos y a detectar fallos de una forma más rápida y eficiente.
* Es la opción perfecta para poder trabajar en equipo en un mismo proyecto.
* Ofrece todas las ventajas del sistema de control de versiones, Git, pero también tiene otras herramientas que ayudan a tener un mejor control de nuestros proyectos.

## Herramientas propias de GitHub

* Wiki. Esta herramienta ayuda en el mantenimiento necesario para las distintas versiones que tenga la página.
* Sistema de seguimiento de problemas. Esta herramienta es la que permite que otras personas, ya sean de tu equipo o no, puedan hacer mejoras, sugerencias y optimizaciones en nuestros proyectos. Así como también nos permite a nosotros hacerlo en los proyectos de otros.
* Herramienta de revisión de código. Esta herramienta nos permite poder hacer pequeñas anotaciones respecto a ciertas partes de un proyecto y con ello tener la colaboración de otros, creando una discusión de las mejoras y optimizaciones.
* Visor de ramas o branch. Nos permite conocer el progreso que llevamos en nuestros proyectos, mostrándonos cuántos updates hemos realizado a partir de nuestro master (primera versión del proyecto), así como también nos permite mirar los cambios que hemos realizado.

En este curso vas a poder aprender mucho más de cómo funciona GitHub y cómo usarlo.

Recuerda que todas tus dudas y/o aportes puedes dejarlos en el sistema de discusiones que encontrarás en cada clase.

# ¿Qué es un sistema de control de versiones?

Un sistema que registra los cambios realizados sobre un archivo o conjunto de archivos a lo largo del tiempo. Este tipo de sistemas nos permiten volver en el tiempo y salvar nuestro trabajo.

La idea con este curso es ser capaces de crear un proyecto del cual tendremos siempre las versiones que modificamos a lo largo del desarrollo.

Los tipos de sistemas de control son:

* **Local Computer:** Solo vive en nuestro computador.
* **Centralizado:** No depende únicamente de un computador en el que se trabaja, sino que depende del súper servidor en donde se almacena la información. El servidor provee las copias a sus hijos, pero solo guarda los cambios en un solo lugar.
* **Sistema de control distribuido:** Cada uno de los que participan en el proyecto, tienen copia del proyecto que se realiza, por eso no dependemos de un solo computador que almacene toda la información.

**NOTA:** Git es un Sistema de Control de Versiones Distribuido.

# Comandos Git

## Git init

Con git init, iniciamos un nuevo directorio de git, es como decir la carpeta contenedora de mi repositorio. Para borrar el repositorio borramos la carpeta .git

## Git status

Nos muestra el status de nuestro repositorio.

## Git add

Con este comando añadimos todos los archivos que creemos en nuestro repositorio.

## Git rm –cached

Recibe un parámetro que se refiere al nombre del archivo que deseamos eliminar de nuestro repositorio.

## Git Commit

Con este comando confirmamos o comprometemos nuestros archivos al repositorio, es decir, con este comando agregamos nuestros archivos del directorio de trabajo al repositorio.

## Git Tag

Con este comando podremos versionar nuestro proyecto

Creando un tag

***Git tag –a <número de versión> -m ‘comentarios de la versión’***

Borrando un tag

***Git tag –d <número del tag>***

Para renombrar un tag

***Git tag –f –a <número de versión> -m ‘comentarios’***

Luego eliminamos el tag al que realizamos el renombramiento

## Git diff

Con este comando podemos ver las diferencias entre los Commit que hemos realizado, algo así como una comparación entre las versiones, esto nos servirá al momento de consultar los cambios que hemos realizado en nuestro proyecto.

Comparando Commit por los tag

***Git diff <versión 1> <versión 2>***

También podemos comparar los Commit con el número de SHA

## Git reset

Con el git reset podemos volver a una versión anterior de nuestro proyecto, existen varios comando que mostraremos a continuación.

Debemos tomar en cuenta que reset recibe como parámetro el SHA de nuestro Commit desde el cual queremos quitar los cambios, es decir, debemos colocar el SHA que queremos que se mantenga en nuestro master

### Git reset –soft

Este comando nos va a quitar los cambios de mi Commit seleccionado y dejara todo listo para hacer un git add a nuestro proyecto, cabe destacar que no elimina los archivos de nuestro directorio

***Git reset –soft <número SHA del Commit que queremos mantener>***

### Git reset –mixed

Este comando a diferencia del git –soft nos elimina todos los cambios que hemos añadido al repositorio, es decir, los add que realizamos en el Commit actual no estarán disponibles, debemos volver a agregarlos.

***Git reset –mixed <número SHA del Commit que queremos mantener>***

### Git reset –hard

Con este comando eliminaremos todo, tanto del staging como del working directory, no nos quedara nada de lo que habríamos realizado después del Commit especificado.

Aun podemos recuperar la información si tenemos el SHA del Commit posterior.

# Las ramas en git

Las ramas en git son de mucha importancia porque nos permiten trabajar en nuestro proyecto de forma ordenada, y definir los privilegios que cada colaborador tendrá en nuestro proyecto.

Crear ramas <nombre de la nueva rama>

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

para crear una rama y moverse a ella de una vez hacemos lo siguiente.

**Git checkout -b <nombre de la rama>**

listar ramas.

**Git branch -l**

para eliminar las ramas hacemos.

**Git branch -d <nombre de la rama>** con esto borramos y para forzar el borrado reemplazamos la -d por una -D

para renombrar las ramas ya creadas.

**Git branch -m <nombre actual> <nombre nuevo>**