# **MAKE IT REAL PREPARACIÓN:** Git y Github

# **Bienvenido al módulo de Git y Github**

En este módulo vas a aprender lo siguiente:

* Qué es git.
* Cómo crear un repositorio de git.
* Qué son y cómo crear commits.
* Cómo consultar el estado de un repositorio de git.
* Cómo consultar el historial de commits.
* Qué son y cómo trabajar con las ramas de git.
* Para qué sirve y cómo usar el archivo .gitignore
* Al final te recomendamos volver a esta sección y verificar que tengas claros estos conceptos.

Aún si ya has trabajado con git, revisar el contenido de esta sección puede ayudarte a cubrir vacíos de conocimiento o a apropiar mejores prácticas. Empecemos!

# **Primeros pasos**

Git es un sistema de control de versiones distribuido creado por Linus Torvalds.

Un sistema de control de versiones nos permite guardar el historial de cambios de nuestros proyectos y trabajar con otros colaboradores en un mismo proyecto.

Git es una aplicación para la línea de comandos. Para instalarla y configurarla sigue las instrucciones que se encuentran en este enlace.

El concepto más importante de git es el repositorio. Un repositorio es una carpeta que va a almacenar todo el historial de cambios de un proyecto.

Para crear un repositorio ejecuta el comando git init sobre la carpeta de tu proyecto. Eso va a crear una carpeta oculta llamada .git.

El historial de cambios se crea a partir de commits. Un commit es una fotografía de tu proyecto en un momento determinado.

Creando un commit

Para crear un commit debes ejecutar dos comandos:

git add .

git commit -m 'Acá va el mensaje describiendo los cambios'

l primer comando, git add, le dice a git que incluya todos los cambios en el siguiente comit.

El segundo comando, git commit, es el que crea el commit. Cada commit tiene la siguiente información:

* Un identificador (una cadena larga de caracteres).
* Un autor (nombre y correo electrónico).
* Una fecha
* Un mensaje que describe el commit (lo escribe uno cuando crea el commit).
* Los cambios desde el último commit.

El estado del repositorio

Para ver el estado del repositorio utiliza el comando git status.

Cuando no hay nuevos cambios vas a ver algo así:

$ git status

On branch master

nothing to commit, working tree clean

Si hay cambios git status muestra tres secciones:

1. Archivos que se van a incluir en el siguiente commit.
2. Archivos modificados o eliminados que no serían incluidos.
3. Nuevos archivos.

El historial de commits

Para ver el historial de commits utiliza el comando git log:

$ git log

commit 7844a6552c2f838db7b7bed81f7be61e4b51ac84

Author: Pedro Perez <pedro@gmail.com>

Date: Sun Jan 19 15:33:49 2020 -0500

Modifica el landing principal

commit fc3b5612f0e23158263ad1a47d91d6c8f84f1a1f

Author: Pedro Perez <pedro@gmail.com>

Date: Sun Jan 19 14:43:21 2020 -0500

Crea el landing principal

commit a4fcb6a1c87958a64e305c302757b591822b7075

Author: Pedro Perez <pedro@gmail.com>

Date: Sun Jan 19 14:23:20 2020 -0500

Primer commit

Este comando muestra los commits en orden descendente (primero los más recientes)

Puedes utilizar la opción --oneline para ver una versión resumida que muestra los primeros 7 caracteres del identificador y el mensaje de cada commit:

$ git log --oneline

7844a65 Modifica el landing principal

fc3b561 Crea el landing principal

# a4fcb6a Primer commit

# **El comando git status**

El comando git status nos permite ver el estado de nuestro espacio de trabajo y del repositorio: los nuevos archivos que hemos creado, modificado o eliminado, los cambios que se van a incluir en el siguiente commit, etc.

Cuando no hay nuevos cambios en el espacio de trabajo vas a ver algo similar a lo siguiente al ejecutar git status:

$ git status

On branch master

nothing to comit, working tree clean

Cuando hay cambios, el comando git status puede mostrar hasta 3 secciones:

1. Cambios que se van a incluir en el siguiente commit (Changes to be commited). A esto también se le conoce como el index.
2. Cambios que no están en el índice (Changes not staged for commit). Estos cambios no serán incluidos en el siguiente commit.
3. Nuevos archivos (Untracked files).

Agregando cambios al índice

Para agregar los cambios de uno o varios archivos al índice utiliza el comando git add seguido de la lista de archivos o de punto (.) para agregar todos los cambios pendientes. Por ejemplo, el siguiente comando agregaría al índice los cambios en index.html

git add index.html

Para agregar todos los cambios pendientes utiliza el siguiente comando:

git add .

Recuerda: Sólo los archivos que estén en el índice serán incluidos en el siguiente commit.

Removiendo cambios del índice

Para remover archivos del índice utiliza el comando git reset HEAD seguido de uno o más archivos. Por ejemplo, asumiendo que tienes un archivo index en el índice y lo quieres remover, utiliza el siguiente comando:

git reset HEAD index.html

Explorando los cambios

Para ver los cambios que aún no están en el índice utiliza el siguiente comando:

git diff

Para ver los cambios que están en el índice utiliza el siguiente comando:

git diff –staged

Descartando cambios

Si quieres descartar los cambios de un archivo completamente, es decir, volver a como estaba en el último commit, utiliza el comando git checkout -- seguido del nombre del archivo:

git checkout -- index.html

Para que este comando funcione, el archivo se debe encontrar fuera del índice. Si el archivo está en el índice debes removerlo como vimos previamente antes ejecutar este comando.

Ver un commit específico

Para ver un commit específico utiliza git show seguido del identificador del commit (puedes utilizar los primeros 7 caracteres). Este comando, además de la información del commit, muestra la lista de cambios.

$ git show fc3b561

commit fc3b5612f0e23158263ad1a47d91d6c8f84f1a1f

Author: Pedro Perez <pedro@gmail.com>

Date: Sun Jan 19 14:43:21 2020 -0500

Crea el landing principal

diff --git a/index.html b/index.html

index 1196288..fe8ec0c 100644

--- a/index.html

+++ b/index.html

@@ -1,2 +1,2 @@

Esta línea ya estaba

+Esta línea se agregó

-Esta línea se eliminó

Cada cambio empieza con la línea diff --git. En este ejemplo sólo tenemos un cambio (el archivo index.html). En las últimas líneas vemos qué cambio: una línea se agregó (la que empieza con +) y otra se eliminó (la que empieza con -). Generalmente en la consola aparecen las nuevas líneas en verde y las que se eliminaron en rojo.

Recursos adicionales: <https://guias.makeitreal.camp/git/conceptos-comandos-esenciales>

# **Trabajando con ramas**

Las ramas (branches) nos permiten desviarnos de la línea principal de desarrollo.

Por defecto git crea una rama por defecto llamada master.

En qué rama estamos?

Para ver en qué rama nos encontramos utiliza el comando git status. La primera línea nos dice en qué rama estamos:

$ git status

On branch master

...

Listar ramas

Para listar todas las ramas utiliza el comando git branch. Por ejemplo:

$ git branch

\* master

rama-1

rama-2

...

La rama que tiene el asterísco (\*) es en la que nos encontramos actualmente.

Crear una rama

Para crear una rama utiliza el comando git branch seguido del nombre que le quieras dar a la rama. Por ejemplo, para crear una nueva rama llamada mi-rama ejecutaríamos el siguiente comando:

git branch mi-rama

Este comando crea la rama pero no nos ubica sobre ella. Para cambiar de rama debes ejecutar el siguiente comando:

git checkout mi-rama

Sin embargo, la mayoría de veces, cuando uno quiere crear una rama también se quiere ubicar sobre ella, así que existe un atajo:

git checkout -b mi-rama

La diferencia con el comando anterior es la opción -b.

Cambiar el nombre de una rama

Para cambiar el nombre de la rama debes estar ubicado sobre ella y ejecutar el comando git branch -m seguido del nuevo nombre.

Por ejemplo, asumiendo que estamos sobre la rama mi-rama, y que queremos cambiarle el nombre a otra-rama, ejecutaríamos el siguiente comando:

git branch -m otra-rama

Integrar una rama

Para integrar los commits de una rama (p.e. mi-rama) a otra (p.e. master) primero debemos ubicarnos sobre la rama principal (p.e. master) y ejecutar el comando git merge seguido del nombre de la rama:

git checkout master

git merge mi-rama

Si la rama master tiene commits que no están en mi-rama se va a crear un commit adicional llamado merge commit.

Eliminar una rama

Para eliminar una rama que ya ha sido integrada en otra utilizamos el comando git branch -d seguido del nombre de la rama que quieres eliminar.

Por ejemplo, asumiendo que la rama mi-rama ya fue integrada a master ejecutaríamos el siguiente comando:

git branch -d mi-rama

Si la rama no ha sido integrada aparecería un error. Sin embargo, si quisieramos eliminar una rama que aún no ha sido integrada podríamos cambiar la opción -d por -D:

git branch -D mi-rama

Recursos adicionales

Para configurar el editor (VSCode o Atom) sigue las instrucciones que se encuentran en el siguiente enlace:

<https://guias.makeitreal.camp/git/instalacion-configuracion#configurando-el-editor-de-texto>

# **Estado de archivos y .gitignore**

Nota: En el video empezamos haciendo un resumen del estado de los archivos, pero desde el minuto 3:48 hablamos del archivo .gitignore.

En git es posible ignorar archivos o carpetas que no queremos incluir en el sistema de control de versiones, ya sea porque tienen información sensible o son autogeneradas por alguna herramienta: logs, archivos temporales, etc.

El archivo donde se define la lista de archivos y carpetas a ignorar se llama .gitignore.

Los archivos y carpetas definidos en .gitignore no van a aparecer como archivos nuevos o modificados al ejecutar git status.

Si aún no existe el archivo .gitignore podemos crearlo con nuestro editor de texto favorito e incluirlo en el siguiente commit.

En el archivo .gitignore escribimos los nombres de los archivo o carpetas que queremos ignorar (uno por línea). También podemos tener comentarios (utilizando numeral #). Por ejemplo:

# esto es un comentario

development.log

build

En este caso estamos ignorando un archivo llamado development.log (que esté en cualquier carpeta) y la carpeta build (incluyendo todos los archivos que estén dentro de la carpeta).

Si queremos ignorar un archivo o carpeta que esté en la raíz del proyecto (no en una subcarpeta), utilizaríamos un / antes del archivo o la carpeta. Por ejemplo, para ignorar el archivo development.log que se encuentra en la raíz utilizaríamos la siguiente línea:

/development.log

También puedes utilizar patrones en el archivo .gitignore:

# ignorar todos los archivos que terminen en .log

\*.log

# excepto production.log

!production.log

# ignorar los archivos terminados en .txt dentro de la carpeta doc (pero no sus subdirectorios)

doc/\*.txt

# ignorar todos los archivos terminados en .pdf dentro de la carpeta doc y sus subdirectorios

doc/\*\*/\*.pdf

En este enlace vas a encontrar algunas plantillas del archivo .gitignore que Github ha creado para algunos lenguajes de programación y frameworks populares. Puede ser un buen punto de partida para tu proyecto o también una buena forma de ver ejemplos de este archivo.

<https://github.com/github/gitignore>

# **Github**

# **Resumen Git**