Git y GitHub

Información del curso

- Organización: freeCodeCamp
- Profesora: Estafanía
- Fecha: 9/02/2023

1 Introducción a Git y GitHub

Git	Es un control de versiones que nos permite rastrear los cambios que hemos hecho en un conjunto de archivos.
GitHub	Es una plataforma pensada para colaboración entre personas en un proyecto o para almacenar el repositorio de un proyecto en la nube. Formalmente: es un servicio de hosting que nos permite almacenar proyectos de desarrollo de software y control de versiones usando git.

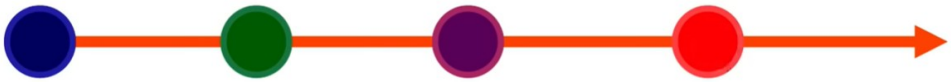
2 Conceptos Básicos

Control de versiones	*Version* Control – Sistema encargado de administrar los cambios realizados en programas de computadora o conjuntos de archivos.
Repositorio	*Repository* – Colección de archivos de distintas versiones de un proyecto.

Un repositorio puede ser local o remoto. Se pueden enviar una copia del proyecto a un repositorio remoto o se puede descargar el repositorio remoto a la PC.

Commit	Componentes básicos de la línea del tiempo de un proyecto de git. Son como un registro o *foto* del estado de un proyecto en un momento específico.

Cada punto de colores es un *commit* diferente.



Git Bash	Herramienta que te permite ejecutar comandos de Git. Es una línea de comandos.

3 Instalar Git

- Web: git-scm.com
- Descargar la versión de git que sea compatible con el sistema en donde se utilizará.
- En el momento de la instalación:
  - Se elige el editor por defecto (VSCode).
  - Nombre de la rama inicial.
  - Se ajusta e PATH (recomendable, Git desde la línea de comandos y también desde software de terceros).
  - Todas las demás opciones se mantienen por defecto.

4 Comandos en Git Bash

Cuando se abre Git Bash, este se encuentra en del directorio raíz o principal del sistema. A continuación una lista con los principales comandos:

Comando	Información

`cd`	*Change Directory* – Me permite moverme al directorio deseado:
	`cd Documents`
	* Si el nombre de la carpeta tiene espacios, se puede introducir con comillas, por ejemplo
	`cd “Esto es una carpeta”`
	* Si le pasamos “..” retornamos un directorio atrás.
	* Si le pasamos no se le pasa nada te enviá al directorio raíz.

`mkdir`	*Make Directory* – Crea una carpeta en el directorio actual.
`ls`	Nos permite ver todos los archivos que se encuentran en el directorio actual.
`rmdir`	*Remove Directory* – Elimina una carpeta del directorio actual.
`clear`	Limpia la consola.

5 Configurar usuario y correo

Cada vez que se crea un *commit*, se registra tu información personal también, así se sabe que usuario realizó el cambió. Esta configuración se puede hacer para un repositorio en particular o para todos los repositorios.

```
git config --global user.name “Lorenzo”
```

Para saber cual es el **`user.name`** se escribe

```
git config user.name
```

De igual manera para configurar el correo (`user.email`).

6 Crear un repositorio

Se trata que los nombres de las carpetas dentro del repositorio sean todos en minúscula (entro de lo que se pueda hacer) y separados con “-”.

El comando que se utiliza para crear un repositorio (*initialize*) es el siguiente

```
git init
```

Una vez hecho esto, se inicializa el repositorio y te crea una carpeta (oculta) “.git” en la cual se almacenarán todos las versiones de nuestro proyecto, por lo que si esta carpeta es eliminada se borra en control sobre el proyecto (directorio actual) al cual estaba asociado.

Cuando en la dirección del directorio en el que te encuentras sale (main) o (master) o …, esto significa que nos encontramos en un repositorio que está siendo monitorizado por git. En caso de querer cambiar el nombre de la rama inicial se hace con la siguiente linea

```
git config –global init.defaultBranch main
```

En este caso el nombre que se seleccionó es *main*, pero esto puede ser el nombre que se desee (sin comillas).

7 Las tres áreas de Git

Cunado se trabaja con Git se tiene tres áreas de trabajo:

1. Directorio de Trabajo.
2. Área de Preparación.
3. Repositorio (directorio .git).

Directorio de Trabajo	*Working Directory* – La carpeta del proyecto que contiene los archivos y el directorio *.git* del repositorio.
Área de Preparación	*Staging Area* – El conjunto de archivos y cambios que serán incluidos en el próximo *commit*.
Repositorio	Directorio que contiene los metadatos y las versiones de tu proyecto. Es la parte del repositorio que se copia cuando clonas un repositorio a tu computadora. Es la parte más importante de Git.

Una vez que se decide pasar de la segunda área a la tercera, se realiza a trabes de un *commit*.

Estas áreas hacen que los archivos tengan tres estados, en dependencia del área en la que se encuentran. Las diferentes versiones del archivo se llaman:

1. Modificada - *Modified*
2. Preparada - *Staged*
3. Confirmada - *Committed*

Modificada	Si la versión del archivo contiene cambios que no son parte del repositorio y no se ha añadido al área de preparación.
Preparada	Si la versión del archivo contiene cambios que no son parte del repositorio pero fue añadida al área de preparación.
Confirmada	Si la versión del archivo ya se encuentra en el directorio de Git.

8 Git status

El estado se verifica con

```
git status
```

Una vez hecho esto, seguramente sale “Untracked files” y “nothing added to commit …”; y esto sale porque no estamos monitorizando ningún archivo y nos muestra que existen archivos que se pueden monitorizar. Para hace que git monitorice a los archivos dentro del directorio se hace con la siguiente instrucción

```
git add <archivo1> <archivo2> … <archivo3>
```

En caso de querer que git no monitorice a dichos archivos se hace así

```
git rm --cached <archivo1> <archivo2> … <archivo3>
```

NOTA: Cuando se va a agregar los archivos con add y todos los archivos que se están monitorizando se quieren agregar, vasta con poner un punto en vez de la lista de nombres.

9 Commit

Commit	Componente básico de la línea del tiempo de un proyecto de Git. Es como un registro o un *foto* del estado de un proyecto en un momento específico. Registra los cambios que se realizaron en los archivos en comparación con la versión anterior.

Cuando se realiza un *commit* viene con la siguiente información:

- Identificador único llamado SHA (Secure Hash Algorithm) o hash (los cambios realizados, dónde se realizaron los cambios y quien realizó los cambios).
- Autor y correo.
- Fecha y hora
- Mensaje asociado (normalmente se escribe algo conciso en la primera linea, y una descripción más detallada en las demás).

Para crear un commit, se puede hacer de diferntes maneras, la más sencilla es la siguiente ya que se hace desde la misma linea de comandos

```
git commit -m “Mensaje descriptivo de los cambios”
```

La otra manera es desde el editor de texto, por ejemplo VSCode. Para hacerlo así solo se escribe en la consola **`git commit`** y el solo abrirá el editor por defecto, para que se escriba el mensaje directamente en el editor.

Una parte muy importante en Git es escribir mensajes concisos y que describan muy bien los cambios realizados. Existen diferentes convenciones para escribir estos mensajes, por ejemplo, el repositorio de freeCodeCamp en GitHub escribe sus mensajes con una sola linea diciendo el cambio.

10 Asociar editor de texto

Digamos que se quiere cambiar el editor que está por defecto en Git, pues se realiza con la siguiente linea

```
git config –global core.editor “code --wait”
```

La opción **`--wait`** hace que Git tenga que esperar a que terminemos de escribir el mensaje en el editor y una vez cerrado el realiza el commit. Si se quiere asociar otro editor de texto se puede consultar en la documentación oficial de GitHub (*associating text editors with Git*).

11 Modificar un commit

Para cambiar el texto del último commit que se realizó, se escribe

```
git commit --amend
```

Luego se abrirá el editor por defecto con el mensaje viejo y una vez que se cambie lo que se deseaba cambiar se cierra la pestaña en el editor y los cambios en el mensaje se guardarán.

NOTA: (miá) en caso de querer cambiar un commit antiguo, creo que lo aconsejable sería crear un commit nuevo con los cambios referentes y haciendo referencia al antiguo.

12 Deshacer el ultimo commit

Supongamos que se quieren eliminar uno o varios commit del seguimiento. Para hacer esto se escribe la siguiente sentencia

```
git reset –soft HEAD~1
```

Esta linea en particular elimina la cabeza de la rama en la que estemos trabajando, o sea, el ultimo commit que realizamos. Esto lo indica el 1 del final de la linea, si en caso de que se quiera borrar el último y el penúltimo se podría un 2 en el lugar del uno (se sigue la misma lógica para los demás casos). La opción **`soft`** indica que solo se eliminará el commit en el seguimiento, no los cambios en el proyecto.

NOTA: En caso de querer que los archivos en el proyecto también sean modificados se cambiaría **`soft`**, por **`hard`**, así en el proyecto se deshacen los cambios de los commit eliminados (reseteados).

13 Ramas en Git y la rama main

Rama	Brach – Una rama en Git es una línea independiente de desarrollo en el repositorio.

(mia) La idea es crear una rama (o una copia de ese commit) para realizar pruebas o fraccionar el trabajo de diferentes desarrolladores, resultado del cual no se vería afectada la rama principal del proyecto. Luego esta rama puede introducirse al proyecto principal (pero esto se ve más adelante).

14 Crea una rama en Git

Para crear una rama en git, se hace de la siguiente manera

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

Para ver las ramas que hay vasta con

```
git brach
```

El asterisco sobre la rama indica en cual ramas te encuentras.

15 Cambiar de rama

Para cambiar de rama escribimos

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

Automáticamente se cambiará la rama en la linea de comandos. Esto se puede verificar con **`git brach`**.

Existe una vía mas rápida para crear la rama y moverte a la rama creada directamente y se hace de la siguiente forma

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

16 Cambiar el nombre de una rama

Para cambiar el nombre de una rama, lo primero es estar en la rama y luego el siguiente comando

```
git branch -m <new-nombre-de-la-rama>
```

También se puede cambiar el nombre de la rama sin estar sobre ella

```
git branch -m <nombre-actual> <new-nombre>
```

17 Eliminar una rama

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

No se debe estar en la rama en la que se va a eliminar.

18 Crear commit en una rama

Lo primero es situarte sobre esa rama y luego añadir los cambios (`add`) del proyecto a dicha rama. Después se realiza el commit como se hacia en la rama principal. Una vez hecho esto, dicho commit se agregó a la rama en la cual estábamos trabajando.

NOTA: La verdadera utilizad de las ramas es que podemos movernos entre versiones de nuestro proyecto y hacer pruebas sin correr el riesgo de arruinar el proyecto principal.

19 Git log para ramas

Muestra la información de los commit realizados sobre la rama en la que te encuentras.

NOTA: Cuando se tienen demasiados commit, el mensaje que devuelve **`log`** es muy largo, por tanto dichos mensajes se pueden personalizar. Se pueden agregar las siguientes opciones

Opciones	Información
`--oneline`	Muestra el hash, la rama en la que se encuentra el commit y la cabecera del mensaje; todo esto en una sola rama.
`--p`	Muestra información detallada de los cambios realizados en los commit.

Para salir de dichos mensajes personalizados, basta con presionar “q”.

20 Fusionar ramas

Una rama tarde o temprano debe fusionarse a la rama principal (si el experimento salió bien).

Fusionar ramas	Proceso que permite combinar varias líneas independientes de desarrollo en una sola rama.

Para hacer esto se hace con el siguiente código, pero atención porque para hacerlo se debe estar en la rama que perdurará en la fusión.

```
git merge <nombre-de-la-rama-que-será-combinada>
```

NOTA: Al combinar ramas no se elimina dicha rama, ya que se puede utilizar en el futuro. Prácticamente al fusionar ramas lo que hacemos es fusionar los commit de ambas ramas.

21 Conflictos al fusionar ramas

Prácticamente un conflicto al fusionar una rama es cuando se tiene una linea (o varias) en una rama de una forma diferente a la otra, por lo tanto se debe decidir como dejar la rama resultante: o de la forma de la rama que se trae, o de la rama base, o ambas ramas juntas o una combinación de las dos ramas. Todo esto es mucho más fácil de hacer si se realiza con el editor de texto (VSC) de forma gráfica que con la linea de comandos de Git.

Una vez que se arregla el conflicto se regresa a la linea de comandos y el estado de la rama quedara como (por ejemplo): **`(main | MERGING)`**. La cual indica que debe continuarse el *merge*, lo cual se logra de la siguiente forma

```
git merge --continue
```

Ya una vez hecho el *merge*. Se realiza el *commit* correspondiente y casi siempre dicho commit se pasas con el mensaje: **`merge branch <nombre de la rama fusionada>`**, para salvar el merge anterior.

Resumiendo todo un poco: lo primero es hacer el merge, luego la linea de comandos señala que tienes un conflicto y te enviá al editor por defecto; aquí se arregla el conflicto y se regresa a la linea de comandos en donde se le dice que quieres continuar con el merge; y por último, se guarda toda la fusión en un commit.

22 GitHub

GitHub	Es una plataforma pensada para colaboración entre personas en un proyecto o para almacenar el repositorio de un proyecto en la nube. Formalmente: es un servicio de hosting que nos permite almacenar proyectos de desarrollo de software y control de versiones usando Git.

23 Crear una cuenta en GitHub

- Para esto hace falta tener un correo para la cuenta.
- Luego debes poner un usuario, lo cual debe ser algo a pensar bien ya que con este nombre se conocerá tu repositorio.
- También se debe configurar cuantas personas trabajarán con esa cuenta y si eres estudiante o profesor.
- GitHub nos suministra muchas aplicaciones internas con las cuales se puede hacer todo el proceso de administración del proyecto aun más fácil.

24 Crear un repositorio