# GitHub

### *Diplomatura en Ciencias Sociales Computacionales y Humanidades Digitales*

#### 1/6/2023

# ¿Por qué Github?

[GitHub](https://github.com/) es una plataforma diseñada para **almacenar y administrar código**, al mismo tiempo nos permite llevar un registro y control de cualquier **cambio** sobre este código.

Es una de las herramientas fundamentales para la **colaboración** y el **trabajo en equipo**.

Tener un registro de todos los cambios (y la posibilidad de deshacerlos!) se vuelve más importante a medida que nos dedicamos a proyectos más complejos y colaboramos con equipos más grandes. GitHub nos permite almacenar información sobre qué cambios se hicieron, cuándo, quién los hizo y por qué se hicieron.

# Dinámica de trabajo

La estructura básica para trabajar con GitHub son los **repositorios**:

Un repositorio o proyecto de Git comprende toda la colección de archivos y carpetas asociados con un proyecto, en conjunto con el historial de revisión de cada archivo.

El **flujo básico de trabajo** en GitHub es el siguiente:

1. Crear/clonar un repositorio.
2. Agregar/modificar código localmente.
3. Enviar nuestras modificaciones al repositorio (en la nube).
4. Descargar a nuestro repositorio local las modificaciones de nuestro equipo.
5. Hacer modificaciones nuevas y enviarlas a GitHub.
6. Repetir infinitamente 4 y 5.

A diferencia de otros softwares de control de versiones (Google Drive, Dropbox, etc.), está especialmente diseñado para trabajar cómodamente con código, y permite formas de colaboración más complejas: básicamente, no sólo hacer una modificación, sino proponer caminos alternativos.

<https://docs.github.com/es/get-started/quickstart/hello-world>

# Cómo interactuar con GitHub?

Hay muchas [opciones](https://docs.github.com/es/get-started/onboarding/getting-started-with-your-github-account#3-elegir-c%C3%B3mo-interactuar-con-github) para interactuar con GitHub, comentamos algunas:

* [GitHub Desktop](https://docs.github.com/es/desktop/installing-and-configuring-github-desktop/overview/getting-started-with-github-desktop)
* [Visual Studio Code](https://code.visualstudio.com/)
* [Rstudio + Git](https://happygitwithr.com/rstudio-git-github.html)

# Algunos recursos

[¿Cómo crear una cuenta?](https://docs.github.com/es/get-started/onboarding/getting-started-with-your-github-account)

[Una guía con ejemplos para empezar a usar Git](https://github.com/git-guides)

[Principales comandos](https://git-scm.com/docs):

* **cd permite fijar el directorio local sobre el que trabajaremos. Esto sirve en una primera instancia para indicar el sitio en el que solicitaremos que Git clone el repositorio que se encuentra en GitHub. Luego, servirá para indicar sobre que directorio local deseamos que trabaje Git. Un repositorio clonado desde GitHub en Git se llamará (master).**

**Un ejemplo de cómo indicar a Git el directorio sobre el que deseamos trabajar es:**

$ cd /c/Users/Chia/Desktop/Martín/Big-Data-y-Cs-Ss

**Cuando tenemos ubicaciones que están nombrados con espacios debemos colocar dicha ubicación entre “”. Por ejemplo, si la carpeta Big-Data-y-Cs-Ss en lugar de tener guiones tuviera espacios, deberíamos escribirla como “Big Data y Cs Ss”.**

* **git config –l, sirve para conocer la configuración.**
* **git clone** clona el repositorio, es decir, nos permite descargar a nuestro directorio local un repositorio.

Por ejemplo:

$ git clone https://github.com/martita17/Big-Data-y-Cs-Ss.git

* **git remote –v permite conocer el repositorio remoto sobre el que estamos trabajando (que en este caso debería ser la misma dirección que hemos clonado anteriormente). Esto puede ser importante más adelante, porque hay casos en los que un directorio local puede estar vinculado a más de un directorio remoto.**
* **git status** muestra nuestra situación: el estado de los cambios locales, en qué rama estamos, etc.
* **git pull** actualiza la línea de desarrollo local con actualizaciones de sus contrapartes remotas. Es decir, descargar a nuestro repositorio local las modificaciones de nuestro equipo.

El código para poder actualizar de manera completa el repositorio de GitHub en el repositorio local es:

$ git pull origin main

El código anterior está compuesto por el “nombre remoto” (**origin**, porque al estar trabajando en el master este ya se encuentra fijado por defecto como el origen) y el nombre de la rama (**main**, en este caso debido a que todavía no estamos trabajando con branchs secundarios).

Por lo que he estado leyendo en https://www.freecodecamp.org/espanol/news/git-pull-explicado/, resulta importante tener todos los cambios confirmados en el repositorio de GitHub, porque en caso de que esto no sea así el pull fallará.

* **git fetch / git merge la primera permite descargar las actualizaciones a nuestro repositorio local como un nuevo branch, sin que reemplace todo de manera direcata como git pull. Luego, git merge permite fusionar los cambios a nivel local del (master) y el branch creado a través de git fetch. Un poco más de info en: https://platzi.com/tutoriales/1557-git-github/11358-git-fetch-y-merge-vs-git-pull-cual-hacer/**
* **git add** + **git commit** + **git push** nos permiten enviar nuestros cambios locales al repositorio. https://www.atlassian.com/es/git/tutorials/saving-changes
  + git add es el primer paso, que agrega archivos nuevos o modificados en el directorio local de trabajo al área de staging (puesta en escena) de Git.

$git add <file>

* + git commit -m “bla bla bla estuve haciendo el cmabio x, y , z” guarda nuestros cambios al historial de versiones. Todo lo que se haya almacenado provisionalmente con git add pasará a formar parte del historial. Con este comando incluimos el mensaje explicando los cambios que hicimos.

$ git commit -m "Primer commit"

* + git reset sirve para deshacer una acción de git add (instantánea preparada) o de git commit (confirmación)
  + git push envía nuestros cambios y así actualiza el repositorio remoto con las modificaciones (ya commiteadas) realizadas localmente.

$ git push origin master

* **Git log: Para ver el historial de cambios realizados en un repositorio.**
* **git reset –hard + número de versión: sirve para volver a una versión anterior del script eliminando las versiones posteriores**
* **git reset –soft + número de versión: permite volver a una versión anterior sin borrar las modificaciones posteriores.**

Un poco más avanzado:

* **git branch** muestra las ramas en las que se trabaja localmente.
* **git branch + nombre de una “nueva rama”, permite la creación de una nueva rama de desarrollo. Por ejemplo**

$ git branch desarrollo-martin

* **git branch –M + “nuevo nombre”: para cambiar el nombre de la rama en la que estoy posicionado.**
* **git checkout nos permite posicionarnos en una determinada rama. Por ejemplo**

$ git checkput desarrollo-martin

* **git checkout –b + “nombre nueva rama”: nos permite crear una nueva rama y posicionarnos sobre ella.**
* **git merge** combina las líneas de desarrollo. Este comando habitualmente se utiliza para combinar los cambios que se realizan en dos ramas distintas. Por ejemplo, un desarrollador podría hacer una fusión cuando necesite combinar los cambios de una rama de característica en la rama de desarrollo principal.
* **Git remote add orgin + dirección del repositorio en Git Hub: para vincular desde la consola de VSC un repositorio local con uno de GitHub**
* **Git remote –v: para saber a qué repositorio remoto se vinculan las acciones**
* **git branch –d + “nombre de rama”:** para borrar una rama. Es importante no estar posicionados sobre dicha rama para poder borrarla.