

Diplomatura en programación web full stack con React JS



Módulo 1:

Introducción a las tecnologías web

Unidad 2:

Git



Presentación

En esta unidad vemos cómo son los pasos para trabajar con sistemas de control de versiones, permitiendo conocer el estado de un proyecto, los cambios que se realizan, y las personas que intervienen.



Objetivos

Que los participantes logren...

- Aprender a trabajar con Git en forma local y remota.
- Conocer cuáles son las herramientas para trabajar con control de versiones.
- Realizar operaciones básicas de los sistemas de control de versiones.
- Realizar la instalación completa de un ambiente de trabajo Git para un proyecto real.



Bloques temáticos

1. ¿Qué es control de versiones?
2. Estados principales.
3. Agregar y commitear archivos y cambios.
4. Ver el log.
5. Github, repositorio remoto.
6. Enviando y trayendo cambios desde un repositorio remoto.

1. ¿Qué es control de versiones?

Git es un sistema de control de versiones distribuido de código abierto diseñado para ser rápido y eficiente.

Un sistema de control de versiones permite a sus usuarios almacenar revisiones diferentes de un mismo archivo y compararlas, restaurarlas y a veces mezclarlas.

Comparado con sistemas de control de versión antiguos como SVN o CVS, Git no necesita de un repositorio (lugar donde se alojan los archivos y sus versiones) central. Esto es lo que lo hace un sistema distribuido lo cual da una flexibilidad imposible con otros sistemas.

Los repositorios de cada usuarios son un clon completo y no una copia parcial de la última versión, esto permite, entre otras cosas, trabajar en modo offline, para luego, en caso de ser necesario, sincronizar los cambios con un repositorio remoto (trabajo colaborativo).

Primeros pasos: Configuración inicial

Para descargar Git, nos dirigimos a <https://git-scm.com/> y descargamos la última versión disponible para nuestro sistema operativo (algunos sistemas ya lo tienen incluido).

Una vez instalado abrimos la consola y escribimos los siguientes comandos, reemplazando por nuestros datos

```
git config --global user.name "Juan Perez"
git config --global user.email "juan.perez@ejemplo.com"
```

Vamos a desglosar los comandos:

git: hace referencia al programa.

config: indica a git que vamos a modificar un elemento de configuración.

--global: el elemento de configuración se va a cambiar de forma global (en toda la computadora) y se usará como valor por defecto cuando no exista de forma local.

user.name y user.email: son las configuraciones que cambiamos.

De esta forma, le estamos diciendo a Git con que nombre y dirección de mail tiene que guardar los cambios que realicemos.

Otro comando que se recomienda correr es

```
git config --global color.ui auto
```

Esto hará que veamos la salida (o respuesta) de los comandos de forma más amigable.

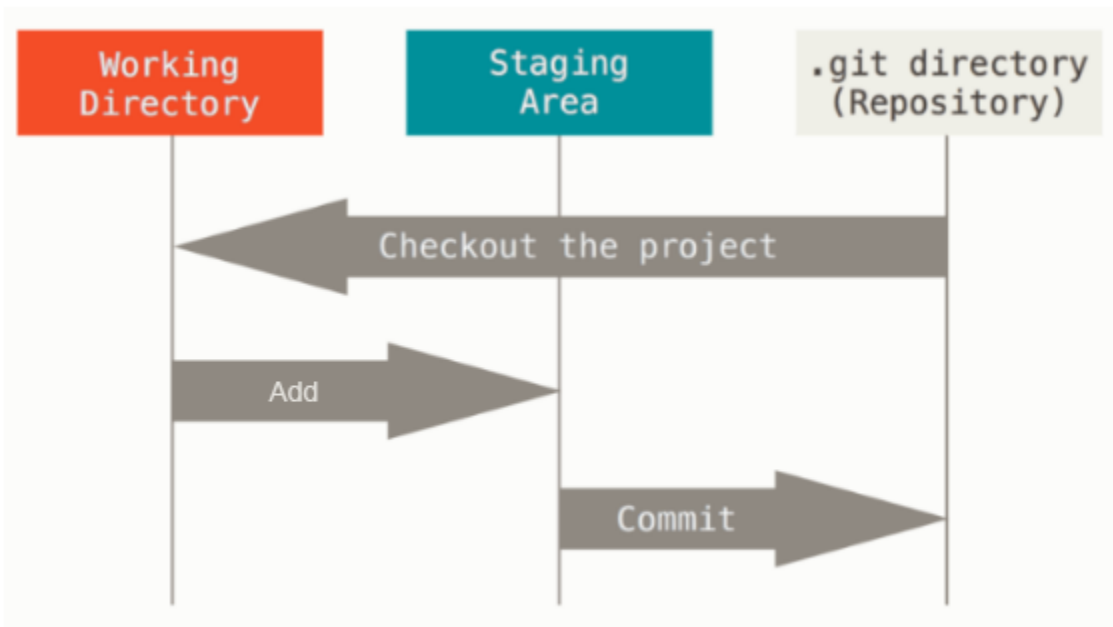
2. Estados principales

Es importante mencionar que git cuenta con 3 estados: **confirmado (committed)**, **modificado (modified)** y **preparado (staged)**.

Modificado significa que modificamos el archivo pero todavía no lo hemos confirmado a nuestro repositorio.

Preparado significa que marcamos un archivo modificado en su versión actual para que vaya en tu próxima confirmación.

Confirmado significa que los datos están almacenados de manera segura en nuestro repositorio local.



3. Agregar y commitear archivos y cambios

Creando nuestro primer repositorio

La creación del repositorio es sumamente sencilla, para ello navegamos con la consola hasta el directorio donde deseamos crear el repositorio (preferentemente un directorio vacío) y escribimos el siguiente comando:

```
git init
```

La salida debería ser similar a

```
Initialized empty Git repository in  
E:/wamp64/www/flavia/webmaster/introduccion_a_git/.git/
```

Podemos corroborar que se creó verificando que se haya creado en ese directorio una carpeta con el nombre .git. Es en esta carpeta donde Git almacena todos los cambios y configuraciones de nuestro repositorio.

Para verificar el estado de nuestro repositorio ejecutamos el comando `git status`. En este caso la salida debería ser similar a esta:



```
On branch master  
  
No commits yet  
  
nothing to commit (create/copy files and use "git add" to track)
```

Con nuestro editor favorito, vamos a crear un nuevo archivo llamado compras.txt. Una vez que lo hayamos creado volvemos a ejecutar `git status`. Ahora la salida debería ser similar a esta:

```
On branch master  
  
No commits yet  
  
Untracked files:  
  
  (use "git add <file>..." to include in what will be committed)  
  
    compras.txt  
  
nothing added to commit but untracked files present (use "git  
add" to track)
```

Lo que Git nos dice acá es que tenemos "Untracked files" o archivos sin versionar y nos muestra una lista donde aparece nuestro archivo. También nos indica que para agregarlo en un commit debemos usar el comando `git add` seguido de la ruta del archivo. Entonces escribimos:

```
git add compras.txt
```

Este comando no devolverá ningún resultado. Lo que el comando add hace es pasar al archivo al estado denominado stage.

`git add` acepta otro tipo de parámetros que nos serán útiles para trabajar con varios archivos a la vez, por ejemplo:

`git add *.jpg` agregará todos los archivos cuyo nombre termine en .jpg al área de stage.

`git add ruta/a/directorio` agregará todos los archivos del directorio al área de stage.

`git add .` agregará todos los archivos modificados y/o no trackeados al área de stage.

Si volvemos a ejecutar `git status` ahora veremos lo siguiente:

```
On branch master
No commits yet
Changes to be committed:

  (use "git rm --cached <file>..." to unstage)

    new file:   compras.txt
```

Git nos indica que en el próximo commit se incluirá el archivo compras.txt que acabamos de crear y agregar al repositorio. Confirmamos la operación escribiendo:



```
git commit -m "creado el archivo de compras"
```

La salida debería ser similar a:

```
[master (root-commit) f23c9cf] creado el archivo de compras 1  
file changed, 0 insertions(+), 0 deletions(-)  
create mode 100644 compras.txt
```

El comando **git commit** confirma los cambios del área de stage y los guarda en el repositorio. El parámetro -m es requerido y se utiliza para asociar un mensaje al commit que estemos haciendo. Este mensaje va siempre entre comillas dobles. Es muy importante que los mensajes sean lo suficientemente descriptivos como para que si el día de mañana revisamos la historia de nuestro repositorio sepamos donde se hizo cada cambio o conjunto de cambios.

Si ejecutamos ahora el comando **git status** veremos que nos indica que no hay cambios y que nuestro espacio de trabajo se encuentra "limpio".

Usando nuestro editor favorito vamos a agregar contenido al archivo compras.txt y vamos a ejecutar una vez más el comando git status. Git ahora no indica que el archivo de compras fue modificado. Para guardar estos cambios en el repositorio ejecutamos:

```
git add compras.txt
```

```
git commit -m "nuevos items en la lista de compras"
```

Nuestros cambios ya están guardados en el repositorio.

En caso de haber modificado más de un archivo que ya teníamos en nuestro repositorio, podemos ahorrarnos el comando `git add` ejecutando la siguiente variación de `git commit`

```
git commit -am "mensaje del commit"
```

Esta variación del comando agrega al área de stage y guarda los cambios de todos los archivos ya trackeados que hayamos modificado. Es de suma utilidad aprender este tipo de atajos, ya que nos ahorran mucho tiempo a la hora de trabajar.

4. Ver el log

Para revisar el historial de nuestro repositorio usamos el comando **git log**. Si seguimos los pasos hasta acá, deberíamos ver algo similar a esto:

```
E:\wamp64\www\flavia\webmaster\introduccion_a_git>git log
commit 867d887d43b05cf7b3bf13e2b6f7fe10ea1d3b75 (HEAD -> master)
Author: Flavia Ursino <flavia.ursino@gmail.com>
Date:   Mon Nov 6 22:42:36 2020 -0300

    nuevos items en la lista de compras

commit f23c9cf09b21d35df08551925c69326decdf0f3c
Author: Flavia Ursino <flavia.ursino@gmail.com>
Date:   Mon Nov 6 22:35:45 2020 -0300

    creado el archivo de compras
```

Ahí se ven todos nuestros commits, desde el más nuevo hasta el más viejo. También se ve el autor (con los datos que ingresamos previamente) y la fecha en la que fueron realizados, así como también el mensaje que se incluyó en cada commit.

Ignorar archivos

A veces es necesario indicar a Git que no realice seguimiento alguno sobre algunos archivos. Para esos casos contamos con un archivo especial que llamaremos **.gitignore** donde podemos definir las reglas de los archivos a ignorar.

Vamos a usar nuestro editor para crear un nuevo archivo llamado privado.txt. Una vez creado, si ejecutamos el comando **git status** podemos ver que figura como

archivo no trackeado. Para hacer que Git ignore por completo este archivo, simplemente creamos el archivo **.gitignore** (*punto gitignore*) en la carpeta raíz de nuestro repositorio y escribimos en él la ruta del archivo que queremos ignorar, en nuestro caso, privado.txt.

Si ejecutamos una vez más el comando `git status`, vemos que privado.txt ya no aparece como archivo nuevo. El que aparece ahora es .gitignore, que debemos agregar y commitear para hacer permanente nuestra lista de archivos ignorados.

5. Github, repositorio remoto

¿Que es Github?

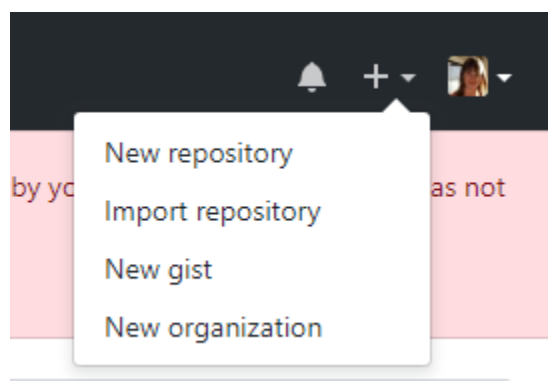
Github es un sitio web y servicio gratuito de alojamiento de repositorios Git. Cualquier persona o empresa puede abrir una cuenta y alojar la cantidad de repositorios que desee.

Es importante tener en cuenta que, por defecto, los repositorios alojados en Github son públicos. Esto quiere decir que cualquier usuario (o no) podrá acceder al código allí alojado, aunque no podrá realizar modificaciones sin que algún administrador o dueño del repositorio en cuestión le brinde acceso de escritura.

Además de brindarnos alojamiento gratuito para nuestros repositorios, Github nos ofrece varias herramientas adicionales orientadas principalmente a la colaboración.

Crear un repositorio

Una vez que tenemos una cuenta, para crear un repositorio utilizamos el signo (+) que figura en la esquina superior derecha, junto a nuestra imagen de perfil, y seleccionamos la opción **"New Repository"**




Se nos presenta un formulario con la información básica que debemos ingresar para crear el mismo.

Create a new repository


A repository contains all the files for your project, including the revision history.


Owner **Repository name**

 flaviaursino /


Great repository names are short and memorable. Need inspiration? How about **glowing-pancake**.

Description (optional)

☒  **Public**
Anyone can see this repository. You choose who can commit.

☐  **Private**
You choose who can see and commit to this repository.

☐ **Initialize this repository with a README**
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: **None** ▾ | Add a license: **None** ▾ 

Owner - Es el dueño del repositorio. En caso de pertenecer a una organización dentro de Github podemos ceder este puesto.

Repository name - Es el nombre que llevará el repositorio. Es el único dato obligatorio y no puede tener espacios.

Description - Podemos incluir una breve descripción a nuestro proyecto.

Public / Private - Aca seleccionamos la visibilidad de nuestro repositorio. Para crear repositorios privados debemos tener una cuenta paga.

Las otras opciones nos permiten crear un archivo de README automáticamente con la creación del repositorio, agregar un archivo .gitignore basado en el lenguaje o framework principal de nuestro proyecto y asignarle un tipo de licencia al mismo.

Una vez completados los datos, apretamos el botón de Create repository y seremos redirigidos a la pantalla principal de nuestro repositorio.

Quick setup — if you've done this kind of thing before

Set up in Desktop

 or

HTTPS

SSH

<https://github.com/flaviaursino/cursowebmaster.git>

We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

...or create a new repository on the command line

```
echo "# cursowebmaster" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/flaviaursino/cursowebmaster.git
git push -u origin master
```

...or push an existing repository from the command line

```
git remote add origin https://github.com/flaviaursino/cursowebmaster.git
git push -u origin master
```

6. Enviando y trayendo cambios desde un repositorio remoto

Subir y enviar cambios de un repositorio remoto

Ni bien creamos nuestro repositorio Github nos muestra los comandos necesarios para empezar a usarlo.

Siguiendo con nuestro ejemplo, vamos a subir el repositorio en el que estábamos trabajando a Github. Para eso, utilizamos el comando que nos indica la pantalla.

Abrimos una consola y escribimos.

```
git remote add origin  
https://github.com/flaviaursino/webmaster.git
```

Con este comando estamos agregando un nuevo “remote” a nuestro repositorio local de Git. Un remote es una copia del repositorio situada en otra computadora. Podemos utilizar esta copia remota para enviar y recibir cambios de nuestro código.

Además especificamos el nombre de remote, por convención, cuando tenemos solo un remote usamos el nombre origin y por último definimos la url donde está situado el remote.

El último paso es enviar nuestros archivos al servidor, para eso usamos el comando especificado



```
git push -u origin master
```

Esto envía (o pushea) a origin los cambios de la branch o rama, master (la principal del repositorio). El parámetro -u hace que la rama master de origin llamada origin/master esté vinculada con nuestra rama local master. Esto hace que Git nos facilite enviar y recibir cambios de esta branch.

```
git pull
```

El comando para traer cambios de un repositorio remoto hacia nuestra computadora, una vez que ya tenemos nuestro repositorio local, es el siguiente:

Si hubiera cambio en el repositorio remoto (en la branch) en la que estemos parados en este momento, este comando los traería hacia nuestra copia local

Bajando nuestro repositorio en otra máquina

Supongamos que hicimos nuestro trabajo en una computadora y lo subimos a Github pero ahora necesitamos seguir trabajando en otra computadora. Para esto, Git nos ofrece otro útil comando llamado **clone**. Para utilizarlo, creamos una carpeta en la nueva computadora y lo utilizamos de la siguiente forma.

```
git clone https://github.com/ekersten/webmaster.git
```

Esto hace una copia del repositorio remoto en la carpeta en la que estemos parados, trayendo toda su historia con él. Es importante notar el espacio y punto al

final del comando. Esto le dice a Git que la copia debe hacerse en el directorio actual. En caso contrario Git crea una carpeta con el nombre del repositorio y lo clona ahí.

Ahora podemos trabajar de forma normal en esta nueva computadora. A medida que vamos trabajando vamos haciendo nuevos commits y cuando estemos listos podemos correr git push para enviar los cambios a Github.

Al haber iniciado la copia en esta computadora a partir de un `git clone` no es necesario que hagamos el proceso de agregar el remote nuevamente. Con correr simplemente `git push` enviaremos nuestros cambios a Github.



Bibliografía utilizada y sugerida

Libros y otros manuscritos:

Scott Chabon, Ben Straub. Pro Git . 2nd Edition. Apress; 2014.