

Introducción a Git

Introducción a Git	1
¿Qué aprenderás?	2
Introducción	2
Introducción a git	3
Control de versiones	4
¿Cuándo debemos usar git?	4
Formas de uso de git	5
Instalando Git	7
Configurando Git	8
Uso básico de git	9
Inicializando git	9
Usando git	9
git add	10
git commit	10
git push	11
¿Local o remoto?	11
Subiendo una nueva versión	12
Gestionando los cambios	12
Ejercicio guiado: Utilizando git en un proyecto (Parte I)	14
Resumen	18



¡Comencemos!

¿Qué aprenderás?

- Aplicar las etapas del versionamiento de Git, para mantener un repositorio de versiones.

Introducción

Respaldo nuestros avances es una necesidad no solo del ámbito de la programación, pero en esta área es realmente crítico, sobretodo cuando trabajamos en proyectos de gran envergadura y con equipos variados. GIT nos permite versionar nuestro código, controlar el flujo de trabajo y crear espacios de trabajo seguros, mientras desarrollamos.

A continuación, conoceremos las funcionalidades más importantes de GIT y los comandos que nos permitirán sacar provecho de esta herramienta.

¡Vamos con todo!



Introducción a git

Existen varios sistemas de control de versiones. Nosotros utilizaremos **git**, el cual gracias a sus capacidades se convirtió en el líder indiscutido.

Git es un sistema de control de versiones gratuito, muy útil y ampliamente utilizado en el desarrollo. Fue creado con la idea de ayudar a manejar proyectos sin importar su tamaño.

Es usado por grandes empresas de desarrollo, como podemos observar en su sitio web.

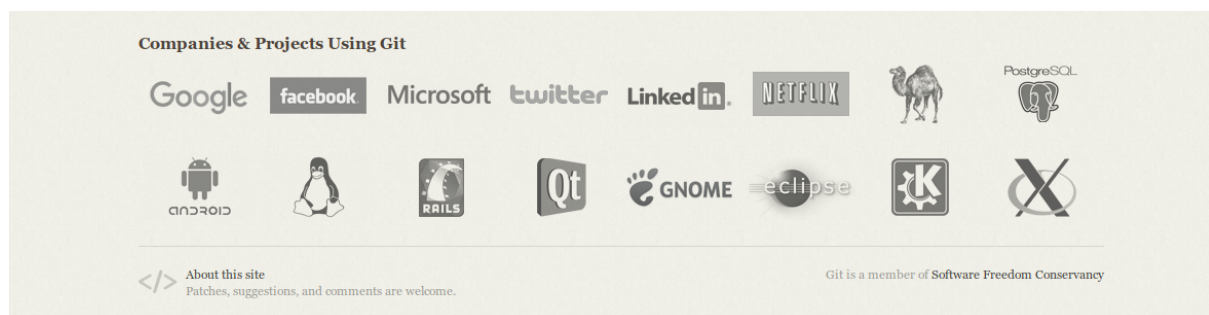


Imagen 1. Sitios que utilizan Git.

Fuente: [Openwebinars](#).

Según la encuesta anual realizada por *stack overflow*, Git es el amplio líder de los sistemas de control de versiones, ocupando un 90% de las preferencias de los desarrolladores. (Fuente [Stackoverflow](#)).

Existen muchas razones para utilizarlo, ya que es un potente software de control de versiones y nos permitirá:

- Recuperar versiones anteriores de nuestro código;
- Recuperar archivos borrados;
- Ayudar a gestionar cambios realizados por otras personas;
- Administrar un proyecto donde trabajan múltiples desarrolladores.

Además, git nos permitirá subir nuestras páginas web a GitHub y crear un portafolio profesional como desarrollador.

Control de versiones

Para entender mejor qué es un sistema de control de versiones, imaginemos un editor de documento de texto como el proporcionado por **Google**, en el cual vamos añadiendo cambios y guardándolos. Si cerramos el programa solo tendremos los últimos cambios guardados. Utilizando git tendríamos acceso a todas las versiones guardadas, permitiendo incluso volver a una de ellas.

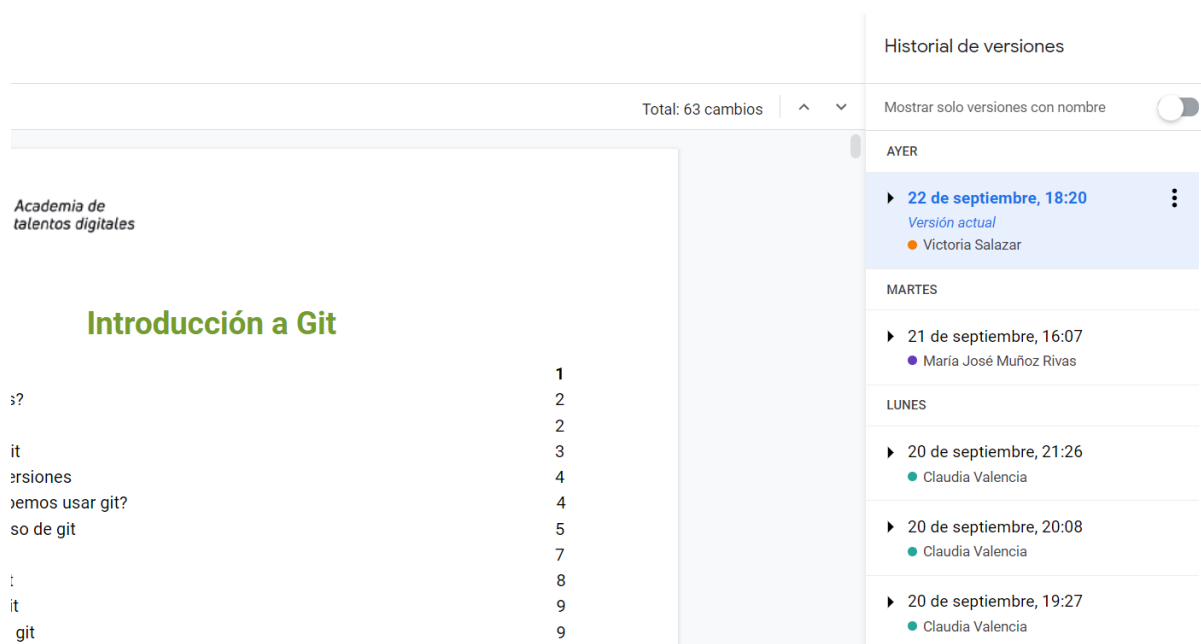


Imagen 2. Ejemplo de un control de versiones en Documentos de Google.

Fuente: Desafío Latam.

¿Cuándo debemos usar git?

La recomendación es usarlo **siempre** que trabajemos desarrollando código (o con documento en texto plano), ya que nos evitará realizar trabajo extra si ocurre algún problema como, por ejemplo, si borramos parte del código que pensábamos que no nos servía, pero luego nos dimos cuenta que sí.

Git también nos ayudará a hacer cambios en el sitio de forma ordenada sin poner en riesgo lo que ya está funcionando.

Durante esta experiencia utilizaremos git en uno de nuestros proyectos para manejar los cambios realizados con la finalidad de subir nuestro trabajo a una plataforma de colaboración o repositorio remoto, solo usando la terminal.

Formas de uso de git

Existen distintas formas de trabajar con git. Algunos editores de texto traen incorporado formas automatizadas para usarlo, por ejemplo en Atom.

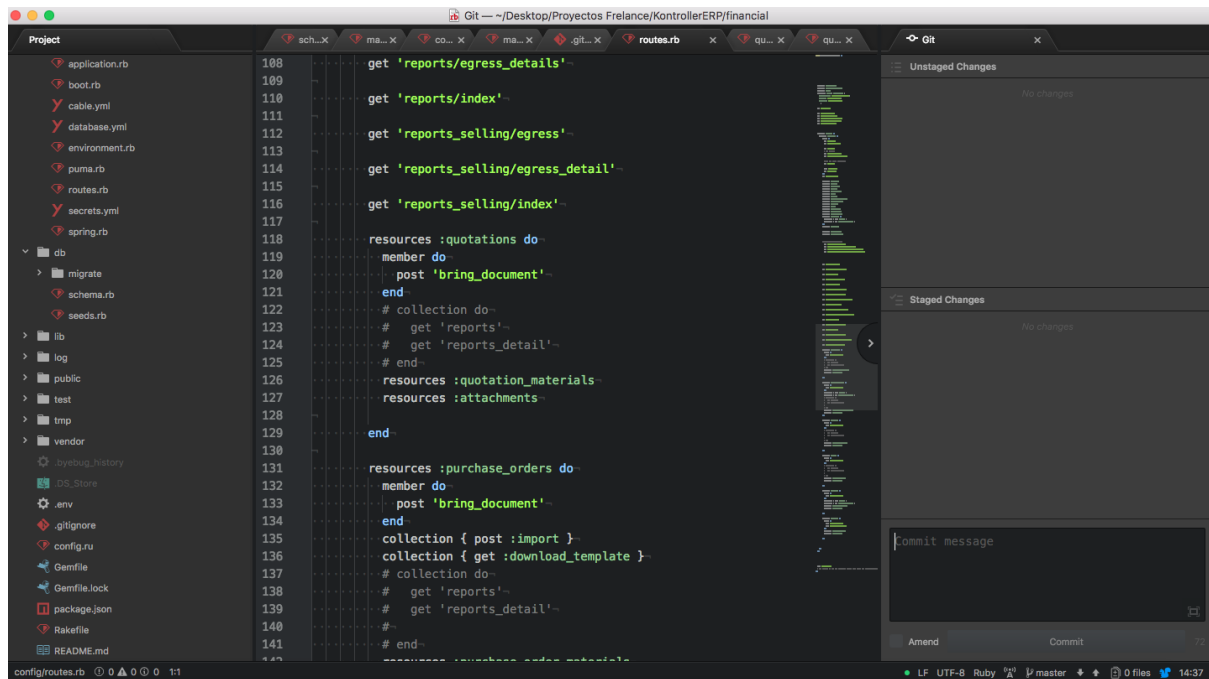


Imagen 3. Editor Atom.
Fuente: Desafío Latam.

También existen programas con interfaces gráficas como gitkraken o git Tower.

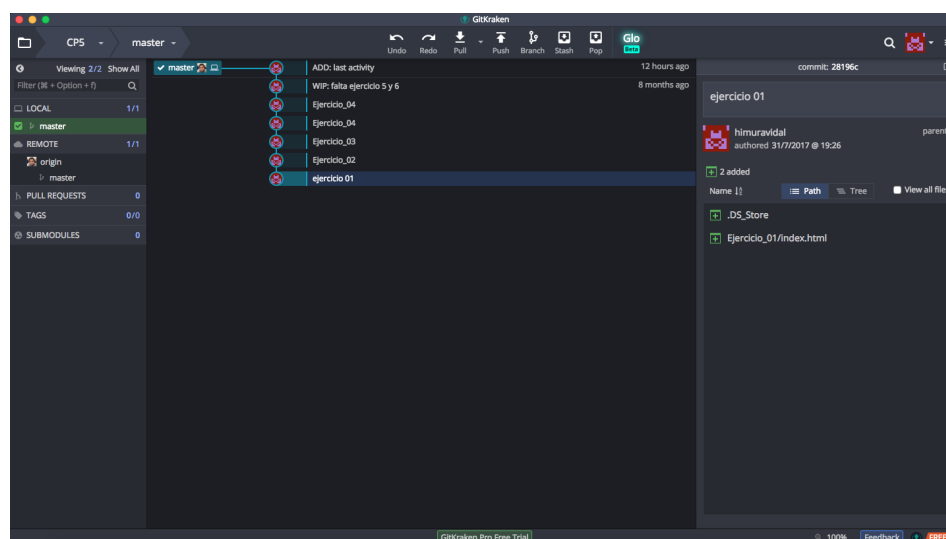


Imagen 4. Gitkraken.
Fuente: Desafío Latam.

Nosotros lo utilizaremos en nuestra terminal. Esto puede parecer a primera vista un poco más difícil, pero nos ayudará a entender mejor los conceptos más importantes.

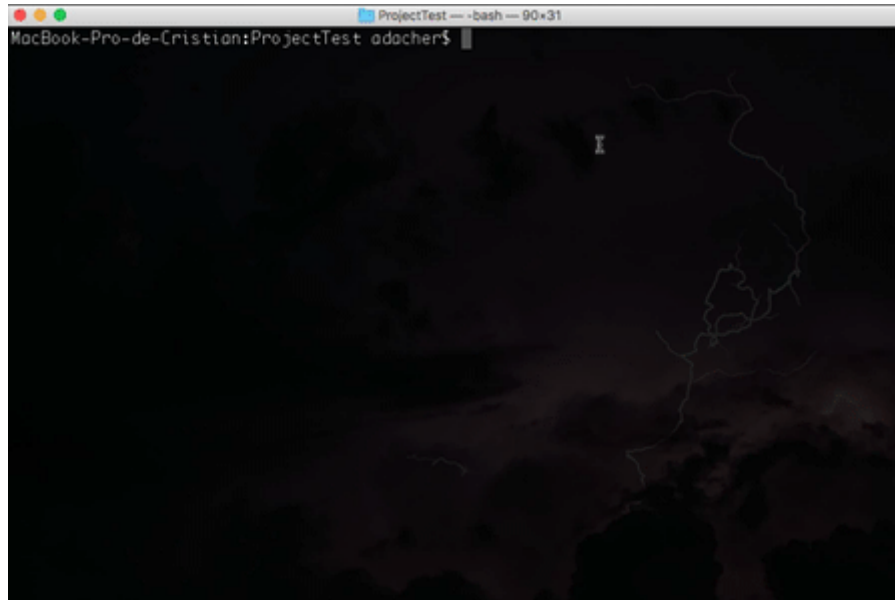


Imagen 5. Vista terminal.
Fuente: Desafío Latam.

Como has podido notar, Git es una herramienta ampliamente solicitada en el mundo del desarrollo, por lo tanto es bueno familiarizarse con ella.

Instalando Git

Vamos a instalar git en nuestro computador. Ahora veamos cuál necesitas dependiendo del sistema operativo que tengas (Mac/Linux, Windows).

Verificando si se encuentra instalado:

El primer paso es verificar si ya tenemos instalado git en nuestro sistema. Esto lo podemos realizar escribiendo el comando `git --version` en nuestra terminal.

Si está instalado, veremos que el terminal nos muestra algo como:

```
git version 2.14.3
```

En computadores con **OSX** es decir, computadores **Mac**, git viene instalado por defecto. Pero si por algún motivo no lo tienes, sigue los siguientes pasos:

1. Entra al sitio de [git](#).
2. Descarga el archivo para **OSX**.
3. Ejecuta el archivo descargado y sigue los pasos del instalador.

En **Linux**, si no está instalado, debemos utilizar los siguientes comandos en la terminal.

```
sudo apt-get install git
```

Y esperar a que termine la instalación.

En **Windows**, si seguiste las instrucciones proporcionadas en la lectura para instalar el terminal, no deberías tener problemas. Pero, si por algún motivo no lo tienes, sigue los siguientes pasos:

1. Entra al sitio [git/win](#).
2. Descarga el archivo dependiendo de tu versión del sistema operativo.
3. Ejecuta el archivo descargado y sigue los pasos del instalador.

Configurando Git

Ahora que ya tenemos **git** instalado, nuestro siguiente paso será configurarlo en nuestro equipo.

Principalmente, lo que tenemos que configurar es nuestro usuario en git.

Para ello, utilizaremos los siguientes comandos:

```
git config --global user.name "Tu Nombre"
```

```
git config --global user.email tucorreo@mail.com
```

En el primer comando debes ingresar tu nombre entre las comillas. Recuerda que este nombre será visible en cada interacción que realices.

Luego, debes ingresar tu correo electrónico, esta vez sin comillas, y también será un registro de las acciones que realices con git.

Una vez ingresados los comandos no veremos ninguna confirmación de la acción entonces puedes usar el comando.

```
git config --list
```

Y deberíamos ver dentro de la lista obtenida los siguientes dos elementos:

```
user.name=Nombre Apellido
```

```
user.email=micorreo@mail.com
```

Si ves este mensaje es porque lo lograste. Ahora que tienes instalado y configurado **git**, en el siguiente capítulo aprenderemos cómo añadirlo a nuestros proyectos.

Uso básico de git

Realizaremos una demostración de los pasos comunes para crear un proyecto con git y manejar cambios.

Inicializando git

Siempre que queramos trabajar con git, nuestro primer paso será escribir en la carpeta de proyecto lo siguiente:

```
git init
```

Todo lo que hace git lo realiza dentro de una carpeta oculta dentro del lugar donde fue inicializado. Si mostramos todos los archivos con `ls -a` podremos ver la carpeta `.git`. Todo ocurre de forma automática en el interior de este directorio.

Con git iniciado empezaremos a trabajar.

Es importante saber que la ejecución del comando `git init` solo lo debemos realizar una vez por proyecto.

Usando git

Para entender cómo funciona git utilizaremos la metáfora de una mudanza.



Imagen 6. Flujo de trabajo de Git.

Fuente: Desafío Latam.

Y realizaremos 3 acciones importantes: añadir, confirmar y enviar.

git add

En una mudanza introducimos nuestras cosas en cajas. Con git es similar. Agregamos nuestros archivos creados y cambios realizados utilizando un comando llamado git add seleccionando uno o varios archivos. Si queremos seleccionar todos los cambios debemos escribir:

```
git add --all
```

0

```
git add .
```

Esto es el equivalente a agregar los archivos a una caja.

git commit

Luego, debemos confirmar estos cambios, que equivale a cerrar la caja y agregarle una etiqueta con una descripción. Esto se logra con `git commit -m "Nombre o descripción del commit"`.

Es importante que la descripción del commit sea, valga la redundancia, descriptiva. Eso es para encontrar e identificar de manera más fácil las versiones de nuestro proyecto.

git push

El último paso del flujo consiste en enviar la caja a destino. Esto se hace vía comando:

```
git push
```

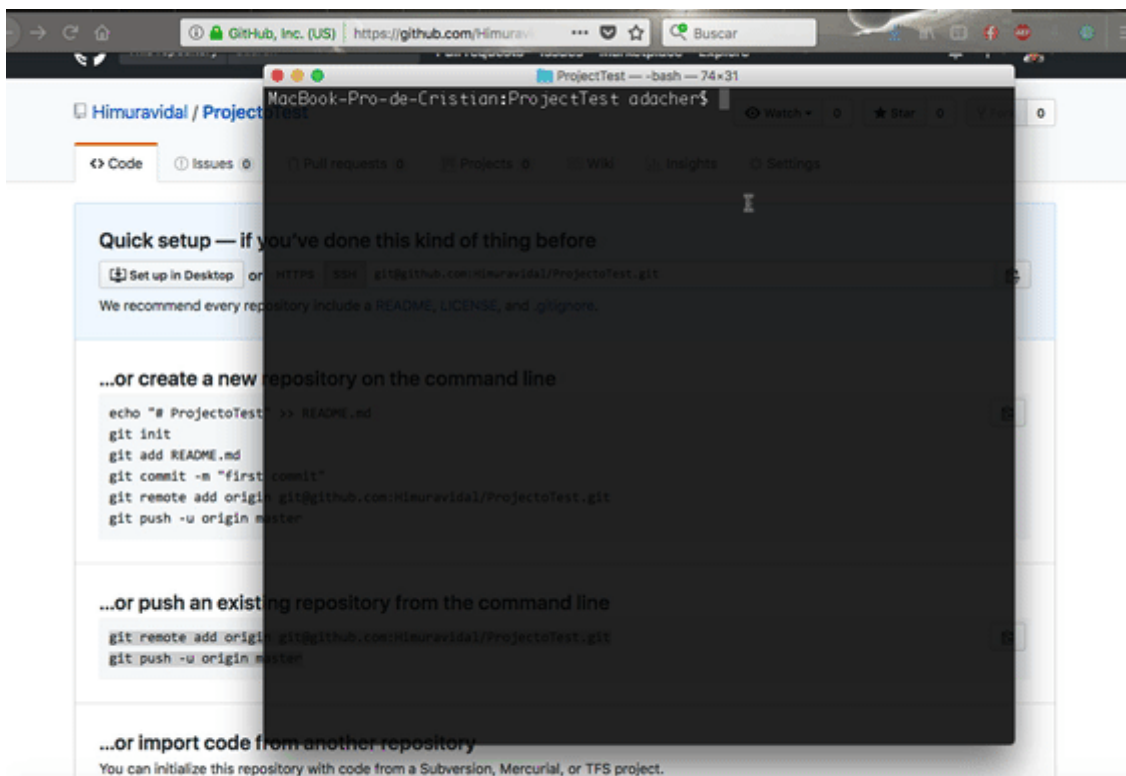


Imagen 6. Uso de git push.

Fuente: Desafío Latam.

¿Local o remoto?

Agregar `git add` y confirmar `git commit` sucede completamente dentro de nuestro computador, en el envío `git push` se usan lugares de destino. Esto lo aprenderemos en el capítulo de GitHub cuando trabajemos con él.

En resumen, el uso típico que haremos de git será `git init` para iniciar git en un proyecto y luego, por cada conjunto de cambios significativos: `git add`, `git commit` y `git push`.

A cada conjunto de cambios commiteados le llamaremos versión.

Subiendo una nueva versión

Finalmente, podemos revisar todas las versiones de un proyecto con:

```
git log
```

Gestionando los cambios

Git permite de forma sencilla ver que cambios hemos hecho contra la revisión anterior. Para probar esto, vamos a introducir otros cambios. Si queremos ver los cambios introducidos en la consola, podemos usar un comando llamado `git diff` para ver qué ha cambiado.

```
Ale@DESKTOP-6FQUK5A MINGW64 ~/Desktop/meet&coffee (master)
$ git diff
warning: LF will be replaced by CRLF in index.html.
The file will have its original line endings in your working directory
diff --git a/index.html b/index.html
index 0c787cf..e6ad049 100644
--- a/index.html
+++ b/index.html
@@ -32,7 +32,8 @@
   <section id="ubicacion">
     
     <h2>¿Donde nos juntamos?</h2>
-    <p>Todos los martes y viernes, de 19:00 a 22:00 en We Work, Calle Baker 133, Providencia, Santiago.</p>
+    <p>Todos los martes y viernes, de 19:00 a 22:00 en We Work, Calle Baker 133, Providencia, Santiago.</p>
+    <p>Este párrafo es el nuevo cambio</p>
   </section>

   <section id="proxima-charla" class="sweet-brown">
Ale@DESKTOP-6FQUK5A MINGW64 ~/Desktop/meet&coffee (master)
$
```

Imagen 7. Respuesta a `git diff`.

Fuente: Desafío Latam.

`git diff` nos muestra todas la diferencia de desde el último **commit** guardado.

Además, cuando hemos introducido cambios, podemos utilizar `git status` para ver un resumen de qué archivos se han modificado.

```
Ale@DESKTOP-6FQUK5A MINGW64 ~/Desktop/meet&coffee (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   index.html

no changes added to commit (use "git add" and/or "git commit -a")

Ale@DESKTOP-6FQUK5A MINGW64 ~/Desktop/meet&coffee (master)
$ |
```

Imagen 8. Respuesta `git status`.
Fuente: Desafío Latam.



Antes de continuar:

¿Existe algún concepto que no hayas comprendido?

Vuelve a revisar los conceptos que más te hayan costado antes de seguir adelante.

Ejercicio guiado: Utilizando git en un proyecto (Parte I)

Vamos a utilizar git dentro de un proyecto. En nuestro caso, vamos a iniciar git en el sitio creado con HTML y CSS de un proyecto anterior llamado "meet&coffee".

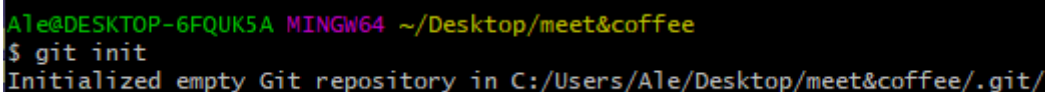
Sigue atentamente los pasos que te presentamos a continuación:

Primero, ubica dónde tienes el proyecto. Para este ejercicio, recuerda que probablemente tienes el proyecto en el escritorio. Abre la consola git bash o directamente en el Visual Studio Code y asegúrate de estar posicionado en la carpeta del proyecto. ¿Listo? Empecemos.

- **Paso 1:** inicializamos git dentro de la carpeta con:

```
git init
```

Observaremos un mensaje indicando que se inició git.



```
Ale@DESKTOP-6FQUK5A MINGW64 ~/Desktop/meet&coffee
$ git init
Initialized empty Git repository in C:/Users/Ale/Desktop/meet&coffee/.git/
```

Imagen 18. Respuesta `git init`.

Fuente: Desafío Latam.

Con esta acción hemos determinado que esta carpeta será nuestro working directory, el lugar donde se almacenarán nuestros cambios. Si utilizamos el comando `ls -a` veremos qué se creó la carpeta `.git`.

- **Paso 2:** recordemos cómo agregar los cambios. Para ello se ocupa git add seguido de todos los archivos que queremos agregar.

Con el comando:

```
git status
```

Veremos un mensaje del tipo:

```
Ale@DESKTOP-6FQUK5A MINGW64 ~/Desktop/meet&coffee (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        assets/
        favicon.png
        index.html

nothing added to commit but untracked files present (use "git add" to track)
Ale@DESKTOP-6FQUK5A MINGW64 ~/Desktop/meet&coffee (master)
$
```

Imagen 19. Respuesta git status.

Fuente: Desafío Latam.

En este punto git nos está diciendo que **No** hemos hecho ninguna confirmación y que hay archivos en nuestro directorio de los cuales no está haciendo seguimiento (regularmente se le conoce como *Tracking*), es desde aquí donde entra la metáfora de la caja.

- **Paso 3:** vamos a empezar por agregar el archivo **index.html**. Utilizaremos el comando:

```
git add index.html
```

Después de hacerlo no obtendremos ninguna información, pero si queremos revisar que sucedió podemos utilizar de nuevo `git status` y veremos:

```
Ale@DESKTOP-6FQUK5A MINGW64 ~/Desktop/meet&coffee (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   index.html

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        assets/
        favicon.png

Ale@DESKTOP-6FQUK5A MINGW64 ~/Desktop/meet&coffee (master)
```

Imagen 20. Respuesta 2 git status.

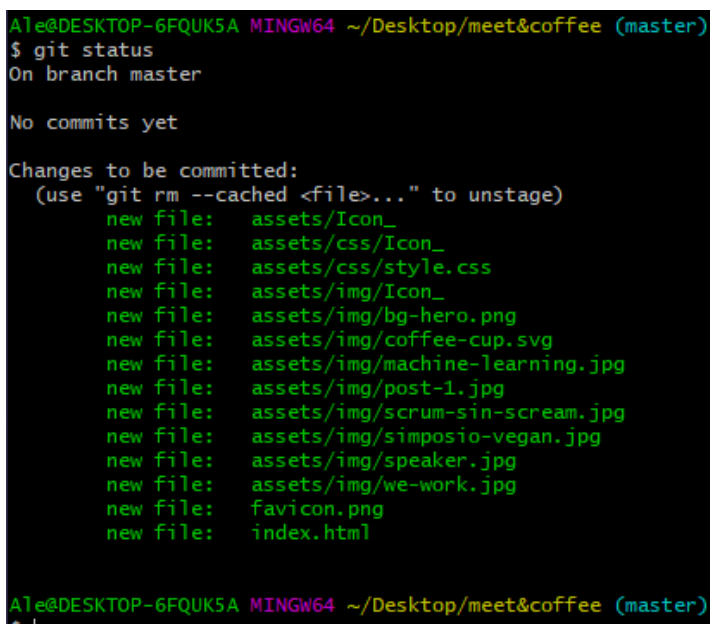
Fuente: Desafío Latam

Como podemos observar, git nos indica que hemos añadido un archivo, pero que aun tenemos otros que no están agregados.

- **Paso 4:** para añadir el resto de archivos, vamos a utilizar el comando:

```
git add .
```

Incluirá todos los archivos que no han sido añadidos aún.



```
Ale@DESKTOP-6FQUK5A MINGW64 ~/Desktop/meet&coffee (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   assets/Icon_
    new file:   assets/css/Icon_
    new file:   assets/css/style.css
    new file:   assets/img/Icon_
    new file:   assets/img/bg-hero.png
    new file:   assets/img/coffee-cup.svg
    new file:   assets/img/machine-learning.jpg
    new file:   assets/img/post-1.jpg
    new file:   assets/img/scrum-sin-scream.jpg
    new file:   assets/img/simposio-vegan.jpg
    new file:   assets/img/speaker.jpg
    new file:   assets/img/we-work.jpg
    new file:   favicon.png
    new file:   index.html

Ale@DESKTOP-6FQUK5A MINGW64 ~/Desktop/meet&coffee (master)
```

Imagen 21. Uso `git status`.

Fuente: Desafío Latam.

- **Paso 5:** lo confirmaremos utilizando `git status`. Y observamos que no queda ningún archivo por añadir.

El mensaje nos dice que los archivos ya están agregados y que nos falta hacer el commit, es decir, la confirmación. Esto será equivalente a cerrar la caja y ponerle una etiqueta con descripción de los cambios que hicimos.

- **Paso 6:** para hacer la confirmación escribiremos:

```
git commit -m "First Commit meet&Coffee"
```


La opción `-m` nos permite escribir ese mensaje en la misma línea donde confirmamos los cambios.

```
Ale@DESKTOP-6FQUK5A MINGW64 ~/Desktop/meet&coffee (master)
$ git commit -m "First Commit meet&Coffee"
[master (root-commit) 39b195c] First Commit meet&Coffee
14 files changed, 227 insertions(+)
create mode 100644 assets/Icon_
create mode 100644 assets/css/Icon_
create mode 100644 assets/css/style.css
create mode 100644 assets/img/Icon_
create mode 100644 assets/img/bg-hero.png
create mode 100644 assets/img/coffee-cup.svg
create mode 100644 assets/img/machine-learning.jpg
create mode 100644 assets/img/post-1.jpg
create mode 100644 assets/img/scrum-sin-scream.jpg
create mode 100644 assets/img/simposio-vegan.jpg
create mode 100644 assets/img/speaker.jpg
create mode 100644 assets/img/we-work.jpg
create mode 100644 favicon.png
create mode 100644 index.html

Ale@DESKTOP-6FQUK5A MINGW64 ~/Desktop/meet&coffee (master)
$ |
```

Imagen 22. Uso de `commit -m`.
Fuente: Desafío Latam.

¡Listo! Lo logramos, hemos guardado nuestra primera versión.

- **Paso 7:** para asegurarnos, utilizaremos el comando `git log`.

```
Ale@DESKTOP-6FQUK5A MINGW64 ~/Desktop/meet&coffee (master)
$ git log
commit 39b195c6d9da708b7ca23e8337145abcb6218a8d (HEAD -> master)
Author: alegonzalezcelis <alegonzalez1993@gmail.com>
Date:   Wed Oct 14 12:16:35 2020 -0300

    First Commit meet&Coffee

Ale@DESKTOP-6FQUK5A MINGW64 ~/Desktop/meet&coffee (master)
$ |
```

Imagen 23. Respuesta a `git log`.
Fuente: Desafío Latam.

Esto nos indicará cuál fue el commit realizado.

La secuencia de letras y números que vemos al comienzo es el **hash**, también se le conoce como **checksum**. Es un identificador único de cada confirmación y sirve para comparar códigos entre distintas versiones, entre otras cosas.

Además aparece el autor de cada confirmación, la fecha cuando fue realizada y el texto de la confirmación. Esto será muy útil para realizar la gestión de cambios en un proyecto donde hayan múltiples personas trabajando.

Resumen

- Git es un sistema de control de versiones gratuito, muy útil y ampliamente utilizado en el desarrollo.
- Todo lo que hace git lo realiza dentro de una carpeta oculta dentro del lugar donde fue inicializado. Si mostramos todos los archivos con `ls -a` podremos ver la carpeta `.git`.
- `git add` permite agregar archivos creados y cambios realizados seleccionando uno o varios archivos.
- `git commit` permite confirmar estos cambios, que equivale a cerrar la caja y agregarle una etiqueta con una descripción: `git commit -m "Nombre o descripción del commit"`.
- `git push` corresponde al envío.
- `git log` permite revisar todas las versiones de un proyecto.
- `git diff` nos muestra todas la diferencia de desde el último commit guardado.