

# Terminal, GIT, GitHub y GitHub pages (Parte I)

## Terminal y manejo de archivos

### Competencias

- Realizar operaciones de navegación de directorios, usando los comandos básicos del terminal, para crear y manipular archivos y directorios.

### Introducción

En un desarrollo es importante realizar la trazabilidad sobre los cambios que se realizan en el código fuente, también poder compartirlo y a la vez mantener registros de los cambios por los que va pasando.

Existen sistemas de versionado de código que se basan en respaldar archivos y carpetas que conforman un repositorio y que permiten operaciones como: volver atrás modificaciones, conocer las diferencias entre las versiones, obtener una copia y publicar los cambios realizados.

Estos sistemas nos van a ser de mucha utilidad ya que nos permitirán gestionar rápidamente los proyectos y visualizar los estados por lo que pasa un proyecto.

A lo largo de esta unidad conocerás cómo controlar las versiones de tu código y cómo respaldar de forma online, esto te permitirá compartirlo y trabajar colaborativamente.

## Introducción al terminal

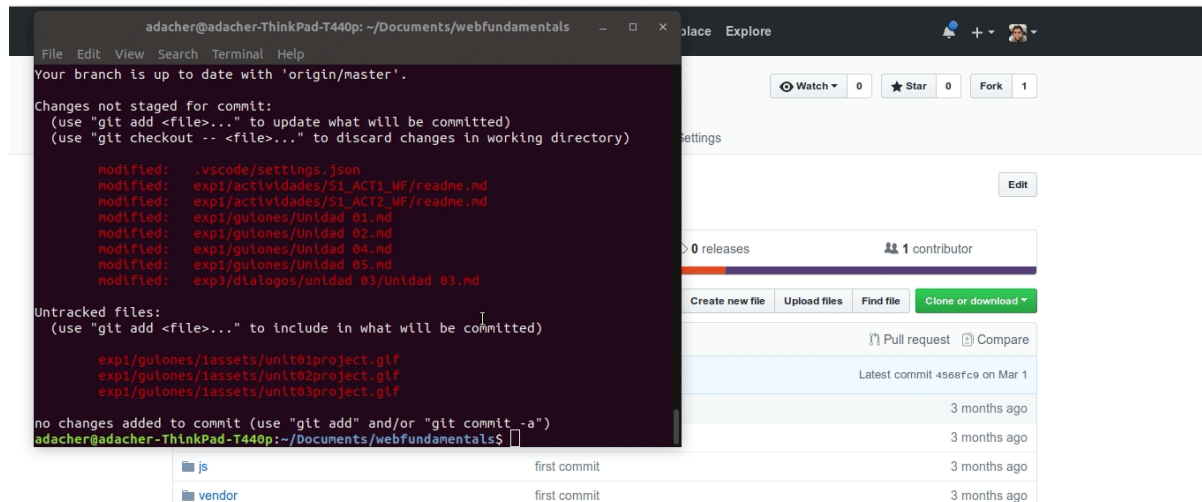


Imagen 1. Instalar GitHub.

Fuente: Desafío Latam.

Para realizar todas estas tareas, necesitarás utilizar una herramienta muy poderosa llamada terminal, que es lo que veremos durante esta experiencia.

El terminal, es una poderosa herramienta basada en una interfaz de texto que sirve para comunicarse directamente con un computador.

Utiliza líneas de comandos para navegar por archivos y directorios, al mismo tiempo se utiliza para interactuar con programas que no tienen interfaz gráfica.

```
pc_usuario~usuario $ ls
```

Ahora, aprenderemos a usar el terminal.

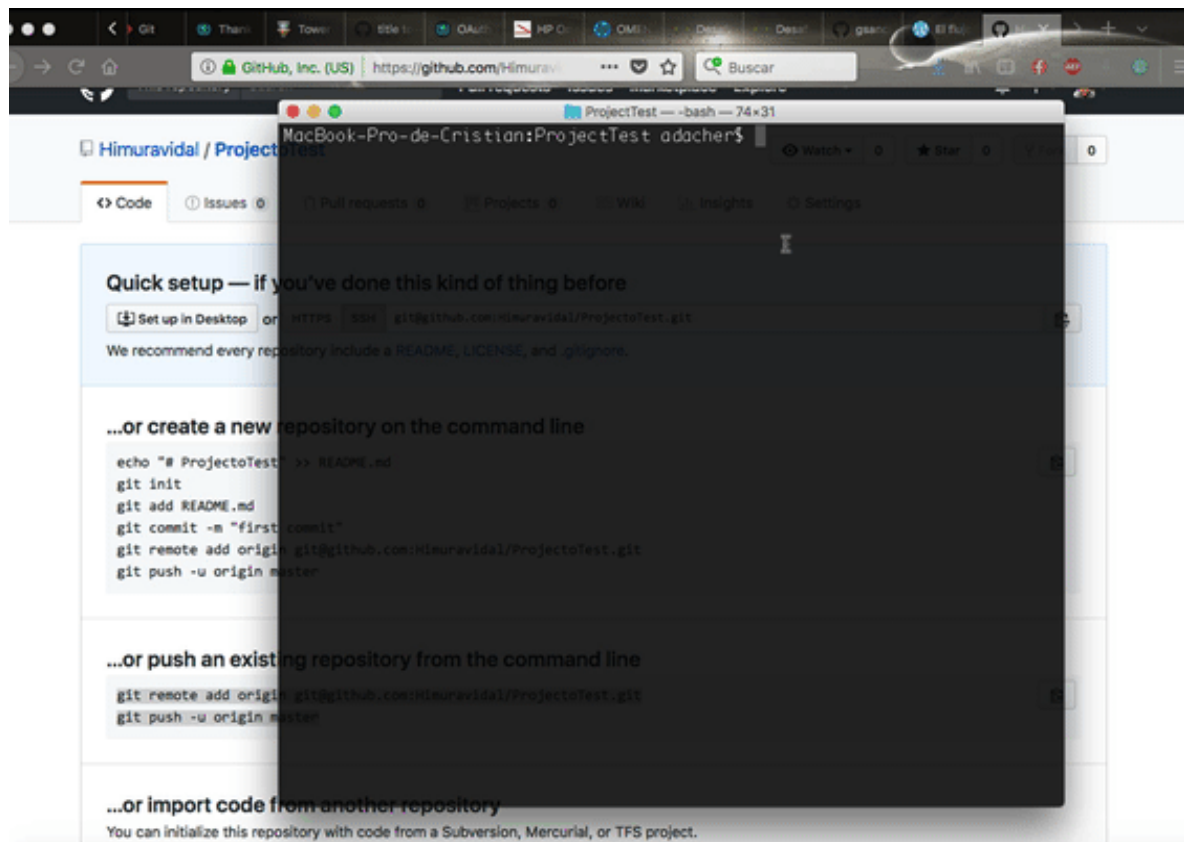


Imagen 2. Terminal.

Fuente: Desafío Latam.

Si estás trabajando en un sistema operativo Linux o OSX (Mac), ya tienes instalada la terminal. En el caso de que estés ocupando Windows, debes bajar un programa especial llamado [git Bash](#).

## Inicialización de terminal

Para inicializar el terminal utilizaremos un atajo:

- **En Linux:** Presiona `ctrl + alt + t`.
- **En Mac:** Presiona `⌘ + espacio`, busca por spotlight terminal.
- **En Windows:** Busca el programa `git bash` y ábrelo.

Nos encontraremos con la siguiente interfaz:

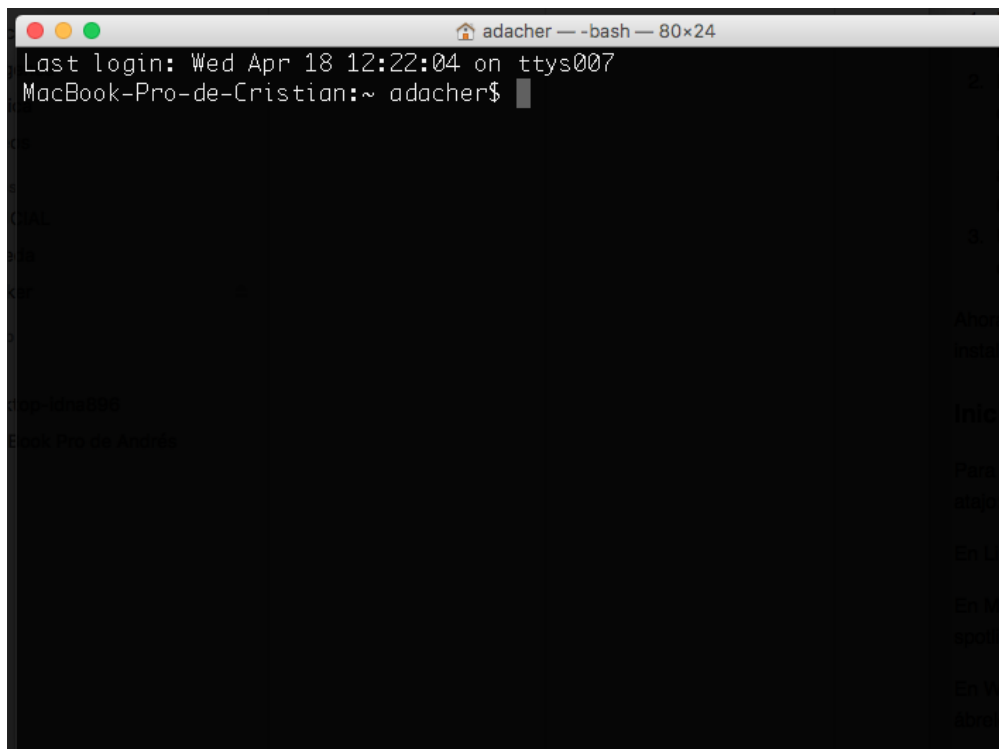


Imagen 3. Muestra de Interfaz.  
Fuente: Desafío Latam.

## Utilizando el terminal

Estudiaremos comandos básicos que nos ayudarán a movernos y ubicarnos dentro de los directorios de nuestro computador, como también listar los archivos o carpetas al interior de un directorio.

Es necesario que sepamos movilizarnos y utilizar la terminal, ya que algunas herramientas solo permiten la interacción a través de ella. En nuestro caso la utilizaremos para controlar las versiones de nuestro proyecto.

## Explicación de las estructuras de directorio

Los directorios o carpetas de nuestro computador tienen una estructura del tipo árbol.

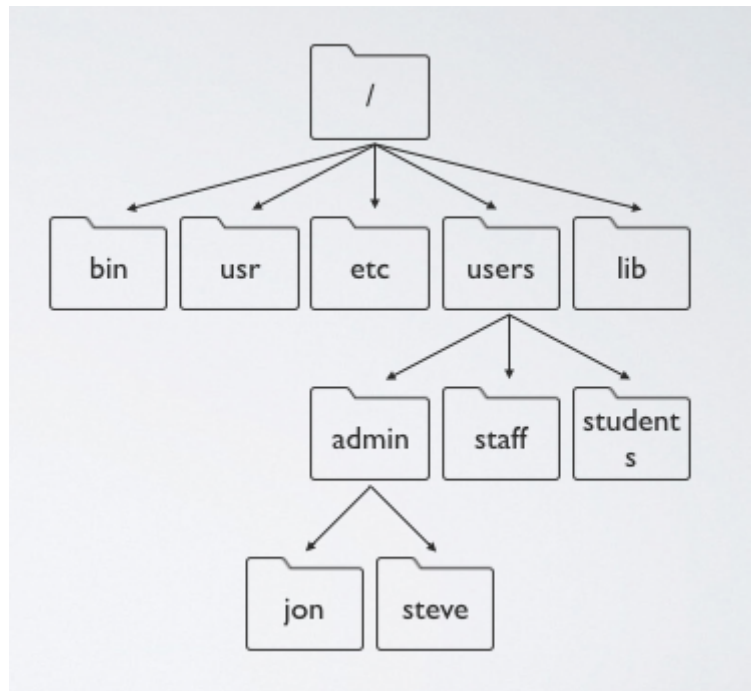


Imagen 4. Estructura de escritorio.

Fuente: Desafío Latam.

Esto quiere decir que el árbol de directorios comienza en la raíz y contiene ramas o directorios, al mismo tiempo que al interior de estos directorios pueden existir archivos u otros directorios.

A veces se utiliza el término técnico **nodo** para referirse a un archivo o un directorio.

El nodo principal de tu computador regularmente se identifica con el símbolo /, también llamado raíz.

Todos los directorios y/o archivos de nuestro computador están dentro de este súper directorio general.

Otra cosa que debes saber es que la estructura de directorios puede cambiar dependiendo del sistema operativo con el cual estés trabajando, así que puede que en tu computador no veas los mismos directorios.

## Conocer en que directorio estamos

Una de las cosas más importantes que necesitamos saber al trabajar con la terminal, es conocer en qué directorio estamos trabajando. Para ello existe un comando que nos entregará esta información.

Escribe en tu terminal lo siguiente `pwd` y presiona enter.

```
[MacBook-Pro-de-Cristian:~ adacher$ pwd  
/Users/adacher  
MacBook-Pro-de-Cristian:~ adacher$
```

Imagen 5. Uso de `pwd`.

Fuente: Desafío Latam.

En tu consola se imprimirá la ruta en la cual estás posicionado. Esto quiere decir que aparecerá como texto.

Recuerda que siempre que escribamos un comando, necesitaremos presionar la tecla `enter` para que se ejecute. Además, recuerda para que Windows estaremos utilizando la consola que nos proporciona `Git bash`.

## Listar archivos

Vamos a observar los archivos o directorios que están en tu carpeta raíz o usuario, para ellos utilizaremos el comando `ls` o sea, list, que como su nombre lo indica, hará una lista de los archivos y directorios contenidos en el directorio en el que se está ejecutando el comando. Para usarlo escribe `ls` en tu terminal y presiona enter.

```
Last login: Thu Apr 19 13:10:41 on ttys000  
MacBook-Pro-de-Cristian:~ adacher$ ls  
Adlm                Music                geth.log  
AndroidStudioProjects  Pictures             logfile  
Applications         ProjectsKeys         mapsexample01.jks  
Desktop              Public               mapsexperiment.jks  
Documents            StudioProjects       node_modules  
Downloads             Test.jks             package-lock.json  
Dropbox               deply@167.99.158.104 prueba.jks  
Library              final.md             prueba1.jks  
LoginExampleSocial.jks flashg4key.jks        prueba3.jks  
Movies               foo  
MacBook-Pro-de-Cristian:~ adacher$
```

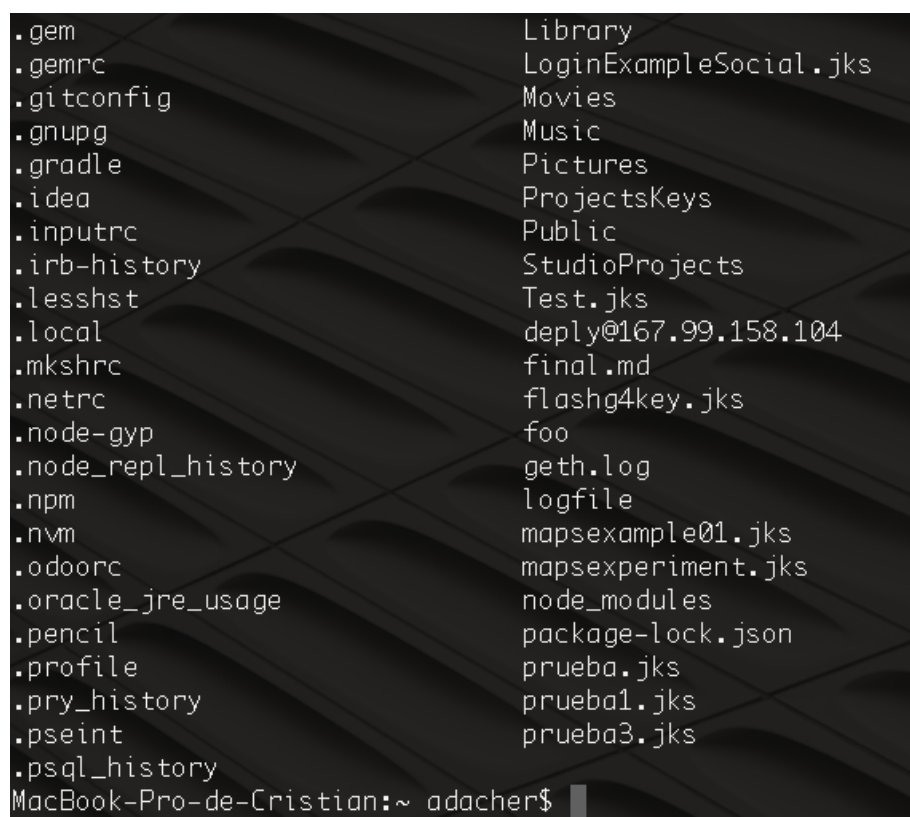
Imagen 6. Listar archivos.

Fuente: Desafío Latam.

Existe también una opción del comando `ls` que nos permitirá observar los archivos ocultos de un directorio.

Los archivos ocultos, son ficheros que tienen un atributo que hace que de forma normal no aparezcan visibles, regularmente son archivos de configuración que no tienen que ser modificados por usuarios normales.

Para poder verlos debemos escribirlo así: `ls -a` el resultado será un listado de directorios y archivos ocultos que comienzan con un `.` antes de su nombre:



```
.gem
.gemrc
.gitconfig
.gnupg
.gradle
.idea
.inputrc
.irb-history
.lesshist
.local
.mkshrc
.netrc
.node-gyp
.node_repl_history
.npm
.nvm
.odoorc
.oracle_jre_usage
.pencil
.profile
.pry_history
.pseint
.psql_history
Library
LoginExampleSocial.jks
Movies
Music
Pictures
ProjectsKeys
Public
StudioProjects
Test.jks
deploy@167.99.158.104
final.md
flashg4key.jks
foo
geth.log
logfile
mapsexample01.jks
mapsexperiment.jks
node_modules
package-lock.json
prueba.jks
prueba1.jks
prueba3.jks
MacBook-Pro-de-Cristian:~ adacher$
```

Imagen 7. Respuesta `ls -a`.

Fuente: Desafío Latam.

## Comandos de navegación entre directorios

Ahora que sabemos en qué directorio estamos posicionados y qué archivos u otros directorios hay dentro de él, aprenderemos a navegar entre directorios. Para ello utilizaremos el comando llamado `cd` lo que significa change directory.

Escribe `cd` / y presiona enter, en la terminal.

Este comando te posiciona directamente a la raíz del sistema de directorios de tu computador, lo que puedes comprobar escribiendo `pwd`.

O sea, si quisiéramos volver al home de nuestro equipo, bastará con escribir `cd` / y clickear enter en la terminal.

`cd` te lleva a la raíz de tu usuario dentro del computador.

Si queremos recorrer e introducirnos en algún otro directorio con `cd` debemos escribir `cd` + la ruta al archivo que queremos llegar. Por ejemplo, accederemos a la carpeta `Desktop` o `Escritorio`, dependiendo de tu sistema operativo, escribiendo:

```
cd Desktop
```

Podemos comprobar que estamos en la carpeta correcta escribiendo `pwd`.

Podemos, nuevamente, escribir `ls` para ver qué archivos o directorios hay dentro de esa carpeta `Desktop` e introducirnos en alguno de ellos con `cd`.

Si ingresamos a un directorio y necesitamos volver atrás, podemos escribir lo siguiente: `cd ..` así seremos dirigidos hacia la carpeta contenedora.

## Anatomía de un comando

Todos los comandos tienen un nombre que los distingue, por ejemplo `ls` y `pwd`, serían el nombre del comando.

Algunos comandos como `cd` además tienen uno o más argumentos. Ejemplo `cd carpeta`. En algunos comandos los argumentos son opcionales y en otros obligatorios.



Hay comandos que pueden recibir opciones. Las opciones las especificamos anteponiendo `-` o `--` al igual que el con el comando `ls` que ya habíamos realizado `ls -a` ya que el `-a` da la opción de ver archivos ocultos.

## Importante

Linux es sensible a las mayúsculas. Esto implica que es distinto escribir `CD` o `cd`. OSX no lo es, pero tendremos esto en cuenta y para seguir la convención escribiremos todos los comandos en minúsculas.

Si quieres configurar los colores y estilo de tu terminal puedes leer esta [lectura adicional](#).

## Manejo de archivos y carpetas

A continuación veremos algunos comandos útiles al trabajar con archivos y carpetas, que optimizarán nuestro tiempo una vez que adquiramos un poco de práctica.

### Creación de archivos

Para crear un archivo nuevo, utilizaremos el comando `touch`. La forma de utilizarlo es `touch nombre_del_archivo.extension`, esto creará un nuevo archivo con el nombre que hayamos ingresado y la extensión que apunta al tipo de archivo.

```
touch archivo.extension
```

### Creación de directorios

Para crear un directorio, se utiliza el comando `mkdir`, que significa make directory, de la siguiente manera:

```
mkdir directorio
```

Esto creará un nuevo directorio con el nombre que lo acompañe.

## Copia de archivos

Para copiar archivos, se utiliza el comando `cp`, el nombre del comando, luego se añade el archivo que vamos a copiar, y finalmente la ruta donde queremos copiarlo, de la siguiente forma:

```
cp archivo.extension ruta_destino/archivo_nuevo.extension
```

## Copia de directorios

Para copiar directorios, también se utiliza el comando `cp`, agregándole la opción `-r`, la carpeta a copiar y la carpeta de destino.

```
cp -r directorio_copiado directorio_destino
```

## Mover archivos y directorios

Para mover archivos, se utiliza el comando `mv` que significa move, el cual se utiliza de una forma muy similar que el anterior `cp`.

```
mv archivo.extension directorio_destino/
```

`mv` también nos permite renombrar el archivo que estemos moviendo, indicando el nuevo nombre en la ruta de destino.

Para mover un directorio utilizaremos la misma sintaxis que el comando `cp`:

```
mv directorio_origen directorio_destino
```

## Borrar archivos

Otro comando importante que debemos conocer y manejar con mucho cuidado es `rm`, ya que con este comando podremos eliminar un archivo y con una opción en su sintaxis, también directorios completos.

Para eliminar un archivo utilizaremos el siguiente comando:

```
rm archivo.extension
```

Ten mucho cuidado, ya que los archivos eliminados de esta forma no van a parar en la papelera de reciclaje de tu computador.

Si por algún motivo escribimos el comando y no existe al interior del directorio, obtendremos la siguiente respuesta:

```
No such file or directory
```

Para eliminar un directorio completo, utilizaremos el mismo comando `rm` pero con una opción.

```
rm -r directorio
```

Esto eliminará por completo el directorio, por lo que debemos tener mucho cuidado al utilizar este comando.

Como ves, la terminal es una poderosa herramienta. Existen muchos comandos más que los que revisaremos, pero por el momento con lo aprendido ya puedes moverte libremente por los directorios, verificar en qué carpetas estás y además crear, copiar, mover y eliminar a través de la terminal.

## Ejercicio guiado: Manejando archivos y carpetas

### Contexto

Como vimos en este capítulo, existen múltiples comandos que nos ayudarán a navegar a través de los archivos y directorios, facilitando nuestro trabajo. Para familiarizarnos con ellos, haremos un recorrido por los comandos que acabamos de revisar:

- **Paso 1:** Lo primero que haremos será crear un nuevo directorio con un comando. Para ello, vamos a la carpeta raíz de nuestro computador. Si no estamos en esta carpeta, vamos a ella escribiendo `cd`.

Ahora escribiremos en la consola:

```
mkdir proyecto1
```

Esto creará un nuevo directorio llamado `proyecto1`.

- **Paso 2:** Crearemos un archivo desde la terminal con el comando `touch`, con el nombre `index.html`:

```
touch index.html
```

Esto crea el archivo `index.html` en la ubicación actual, o sea la carpeta raíz del computador. Si utilizamos `ls` podremos ver el archivo creado.

- **Paso 3:** Ahora que ya sabemos crear un directorio y un archivo, copiaremos el archivo `index.html` dentro del directorio que habíamos creado, o sea en la carpeta `proyecto1`, de la siguiente manera:

```
cp index.html /proyecto1/index.html
```

- **Paso 4:** Utilizando este mismo comando también podremos cambiarle el nombre a los archivos copiados. Volvamos al directorio anterior con `cd ..`, luego ocuparemos `cp` nuevamente pero con un nuevo nombre de archivo.

```
cp index.html /proyecto1/index2.html
```

Si ingresamos a `proyecto1` y listamos los archivos con `ls`, debería aparecer nuestro archivo copiado con el nuevo nombre.

- **Paso 5:** Si queremos copiar un archivo dentro de la misma carpeta donde se encuentra, solo debemos escribir el nombre del archivo seguido del nombre que le queremos poner.

```
cp index.html index3.html
```

- **Paso 6:** Como vimos anteriormente, con el comando `cp` también podremos copiar un directorio. Para esto, en nuestra terminal volveremos a nuestro home con `cd` y crearemos un nuevo directorio llamado `assets` con:

```
mkdir assets
```

- **Paso 7:** A continuación, copiaremos esta carpeta al interior de `proyecto1` de la siguiente forma:

```
cp -r assets proyecto1
```

- **Paso 8:** Revisaremos que se haya realizado ingresando con `cd` a `proyecto1` y listando los archivos con `ls`. Aparecerá la carpeta copiada.

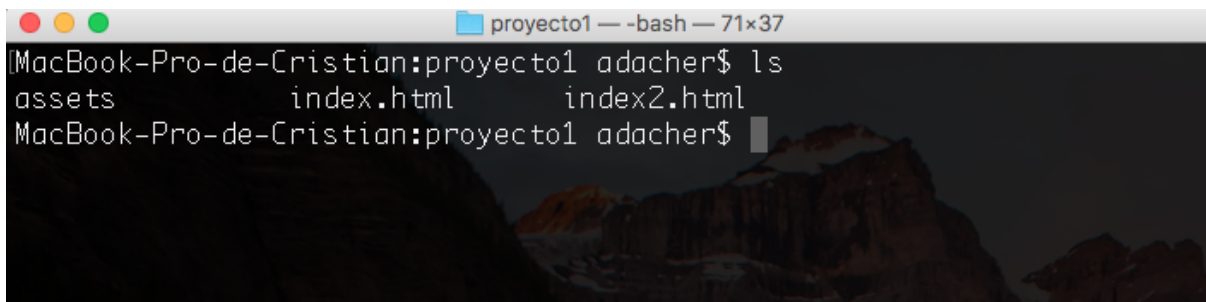


Imagen 8. Lista de carpeta `proyecto1`.  
Fuente: Desafío Latam.

- **Paso 9:** Ahora copiaremos un archivo. Volvamos a nuestro home con `cd`, crearemos un nuevo archivo llamado `touch readme.txt` y ahora lo moveremos dentro de `proyecto1`, con la siguiente línea de comandos:

```
mv readme.txt proyecto1/
```

- **Paso 10:** Ingresamos a `proyecto1` y listamos los archivos para comprobarlo con `ls`. Debería existir nuestro nuevo archivo.

`mv` también nos permite renombrar el archivo que estemos moviendo, indicando el nuevo nombre en la ruta de destino.

- **Paso 11:** Ahora probemos copiando un directorio. Para esto, volveremos a nuestra carpeta de usuario con `cd`, luego crearemos otra carpeta con `mkdir img` y la copiaremos dentro de `proyecto1/assets` con el siguiente comando:

```
mv img proyecto1/assets
```

- **Paso 12:** Para comprobar que todo haya funcionado correctamente, ingresaremos a la carpeta `assets` del `proyecto1` con:

```
cd proyecto1/assets
```

Y listamos los archivos con `ls`.

- **Paso 13:** Finalmente, probaremos el comando para eliminar archivos. Para ello, ingresamos a nuestra carpeta `proyecto1` y eliminaremos nuestro archivo `readme.txt` con el siguiente comando:

```
rm readme.txt
```

- **Paso 14:** Ahora probemos eliminando el directorio `proyecto1`. Volvamos a nuestro home con `cd` y escribamos el siguiente comando.

```
rm -r proyecto1
```

Recuerda que esto eliminará por completo el directorio, por lo que debemos tener mucho cuidado al utilizar este comando.

## Ejercicio Propuesto (1)

Tenemos la siguiente estructura de carpetas:

```
alumno |--- apuntes      |--- modulo1.doc
        |--- modulo2.doc
        |--- modulo3.doc
    |--- documentos
    |--- ejercicios |--- modulo1.html
                |--- modulo2.html
                |--- modulo3.html
                |--- modulo4.html
                |--- modulo5.html
```

1. Se pide crear la estructura de carpetas y archivos por medio de consola.

2. Se pide crear una carpeta por cada módulo, manteniendo la estructura de subcarpetas, de la siguiente forma:

```
alumno |--- modulo1      |--- apuntes          |--- modulo1.doc
      |--- documentos
      |--- ejercicios      |--- modulo1.html
|--- modulo2 |--- apuntes      |--- modulo2.doc
      |--- documentos
      |--- ejercicios      |--- modulo2.html

|--- modulo3 |--- apuntes      |--- modulo3.doc
      |--- documentos
      |--- ejercicios      |--- modulo3.html
|--- modulo4 |--- apuntes
      |--- documentos
      |--- ejercicios      |--- modulo4.html
|--- modulo5 |--- apuntes
      |--- documentos
      |--- ejercicios      |--- modulo5.html
                        |--- proyecto1
```

3. Se pide eliminar las carpetas obsoletas:

- alumno/apuntes.
- alumno/documentos.
- alumnos/ejercicios.



## Introducción a Git

### Competencias

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

### Introducción

Respaldo nuestros avances es una necesidad no sólo 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.

### 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 no importando su tamaño.

Git es usado por grandes empresas de desarrollo, como podremos observar en su sitio web.

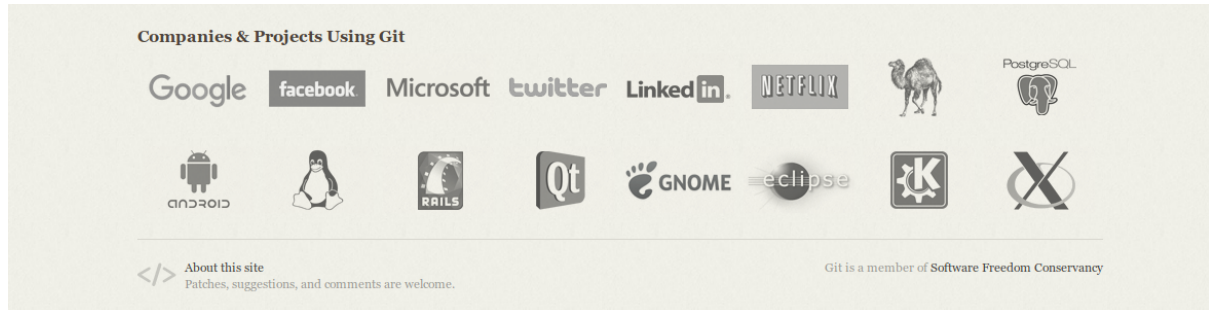


Imagen 9. Sitios que utilizan Git.

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 utilizar git. 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 nuestra 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 **Word**, 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.

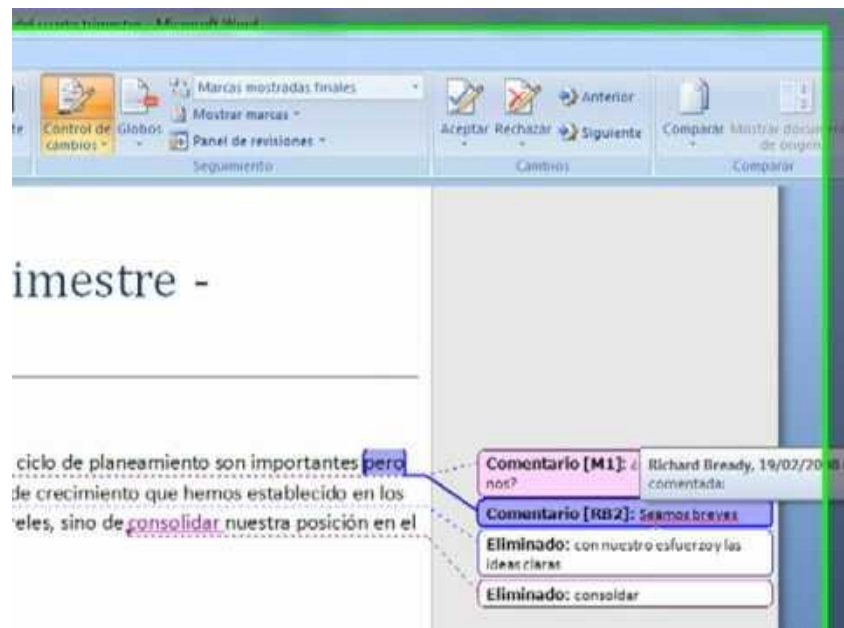


Imagen 10. Git en word.

Fuente: Desafío Latam.

¿Cuándo debemos usar git?

¿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.

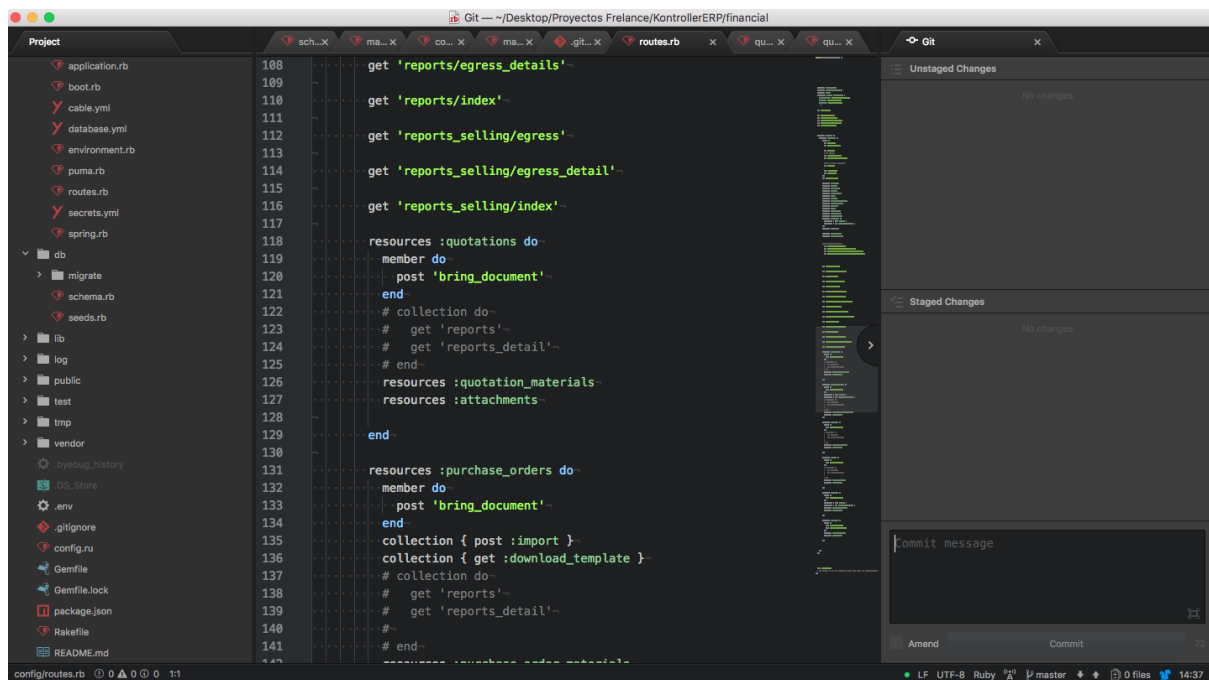


Imagen 11. Editor Atom.

Fuente: Desafío Latam.

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

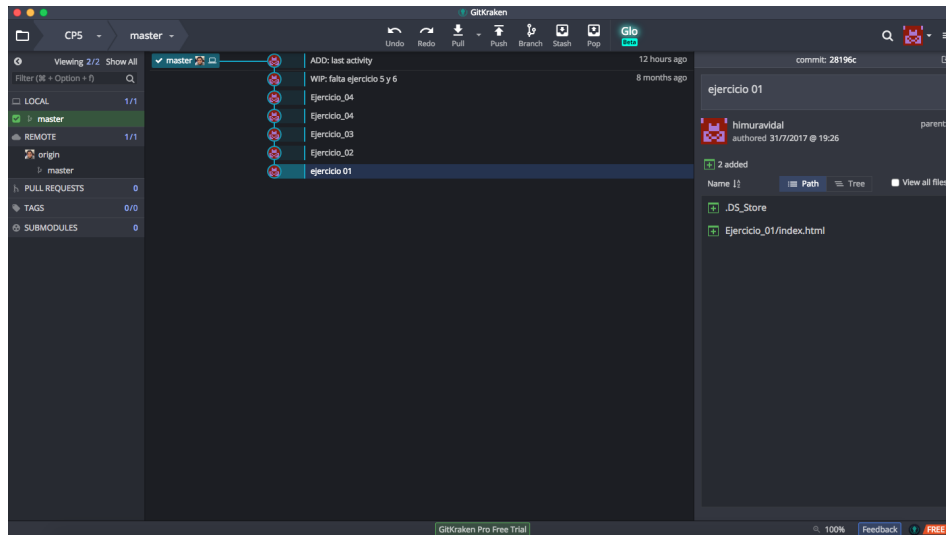


Imagen 12. Gitkraken.  
Fuente: Desafío Latam.

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

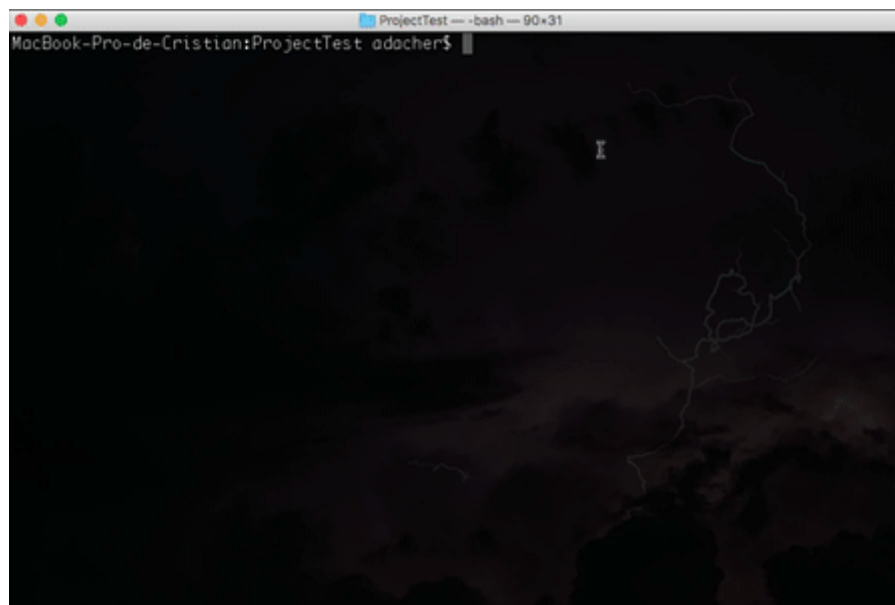


Imagen 13. Vista terminal.  
Fuente: Desafío Latam.

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 qué instalación 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 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=Francisca Medina
```

```
user.email=fbmedina@uc.cl
```

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 14. 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 seleccionarlos 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
```

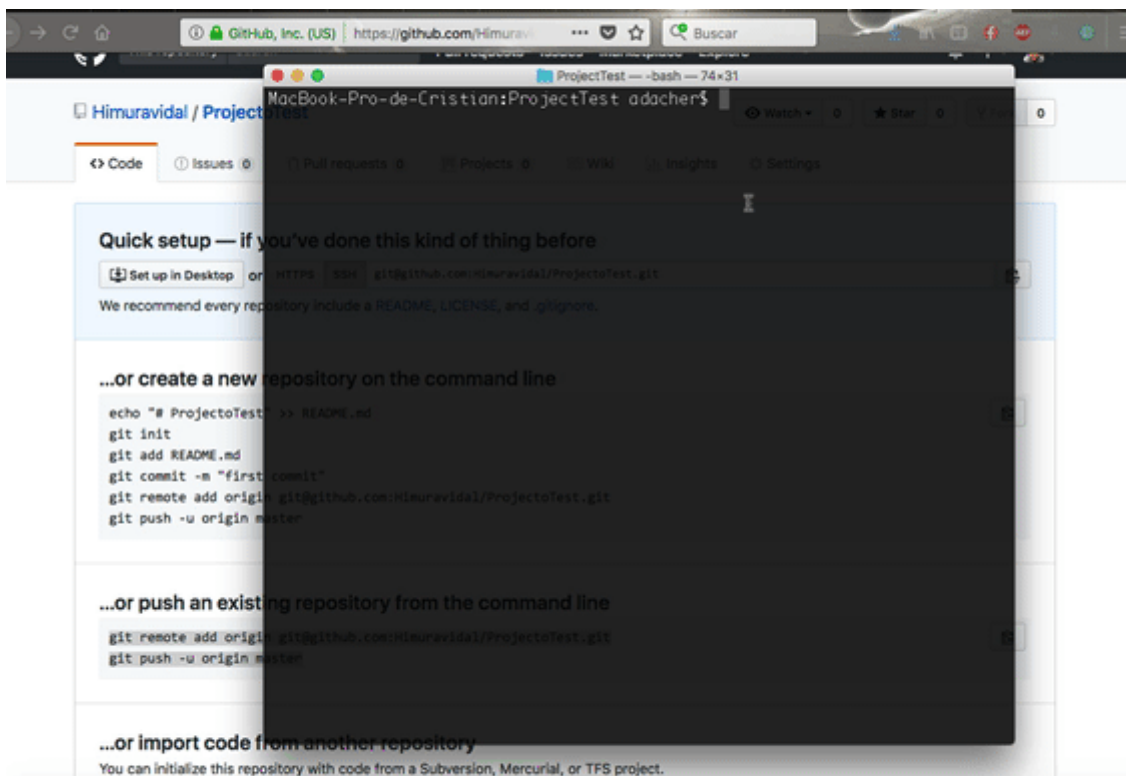


Imagen 15. Uso de git push.

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 16. 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.

```
Al@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")

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

Imagen 17. Respuesta `git status`.  
Fuente: Desafío Latam.

## 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 una experiencia anterior llamado "meet&coffee".

Primero, ubicamos donde tenemos el proyecto. Para este ejercicio, tenemos el proyecto en el escritorio. Abriremos la consola e ingresamos a esa ruta con `cd`.

Si no tienes el proyecto, puedes descargarlo desde este [enlace](#).

Proseguiremos los siguientes pasos:

- **Paso 1:** Ingresamos a la carpeta contenedora del proyecto meet&coffee, que en este caso de `Desktop`.

```
cd Desktop
```

- **Paso 2:** Ingresamos a la carpeta meet&coffee.

```
cd meet\&coffee
```

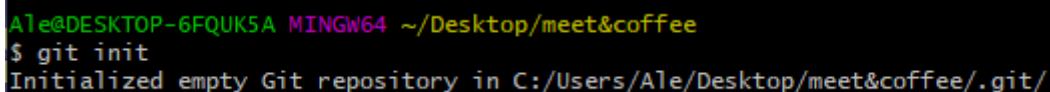
- **Paso 3:** Podemos corroborar que estamos dentro de ella con:

```
pwd
```

- **Paso 4:** Inicializamos git dentro de la carpeta con:

```
git init
```

Observaremos un mensaje indicando que se inició git.

A terminal window with a black background and green text. The prompt is 'Ale@DESKTOP-6FQUK5A MINGW64 ~/Desktop/meet&coffee'. The command '\$ git init' has been entered, and the output is 'Initialized empty Git repository in C:/Users/Ale/Desktop/meet&coffee/.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 -la` veremos qué se creó la carpeta `.git`.

- **Paso 5:** 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 6:** 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.

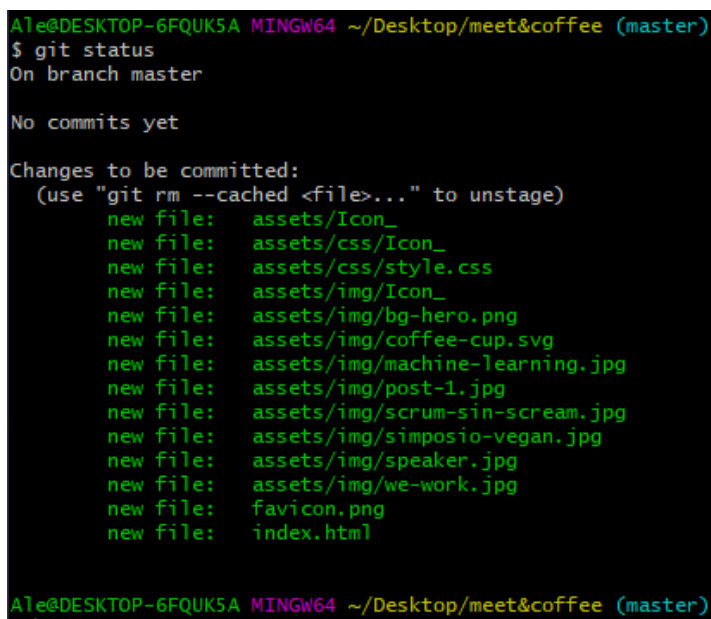
Fuente: Desafío Latam

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

- **Paso 7:** Para añadir el resto de archivos, vamos a utilizar el comando:

```
git add .
```

Que incluirá todos los archivos que no han sido añadidos aún.

A terminal window with a black background and green text. The prompt is 'Ale@DESKTOP-6FQUK5A MINGW64 ~/Desktop/meet&coffee (master)'. The command '\$ git status' has been executed. The output shows 'On branch master' and 'No commits yet'. Under 'Changes to be committed:', it lists 14 new files: assets/Icon\_, assets/css/Icon\_, assets/css/style.css, assets/img/Icon\_, assets/img/bg-hero.png, assets/img/coffee-cup.svg, assets/img/machine-learning.jpg, assets/img/post-1.jpg, assets/img/scrum-sin-scream.jpg, assets/img/simposio-vegan.jpg, assets/img/speaker.jpg, assets/img/we-work.jpg, favicon.png, and index.html. A hint '(use "git rm --cached <file>..." to unstage)' is shown above the list.

```
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 8:** 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 9:** 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 10:** 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.

- **Paso 11:** Vamos a hacer un `git add .` y probaremos de nuevo con `git status`.

En este caso veremos que `git status` nos muestra que los cambios han sido añadidos, pero falta confirmarlos, que es lo que haremos a continuación.

```
git commit -m "added new text to the index of meet&coffee project"
```

- **Paso 12:** Si revisamos con `git status` veremos que ya no hay información nueva que confirmar.

## Ejercicio Propuesto (2)

Toma un proyecto anterior o incluso la estructura de carpetas que hemos creado en el ejercicio 1 y realiza los pasos para iniciar git y versionar los cambios de tu proyecto.

1. Inicializa git en la carpeta contenedora.
2. Agrega los archivos al stage.
3. Genera el primer commit.
4. Muestra el status.

## Solución de Ejercicios Propuestos

### Solución Ejercicio Propuesto (1)

1. Se pide crear la estructura de carpetas y archivos por medio de consola.

- **Paso 1.** Creamos la estructura de las carpetas:

```
mkdir alumno
mkdir apuntes
mkdir documentos
mkdir ejercicios
touch modulo1.doc
touch modulo2.doc
touch modulo3.doc
touch modulo1.html
touch modulo2.html
touch modulo3.html
touch modulo4.html
touch modulo5.html
```

- **Paso 2:** Movemos los directorios a su estructura correspondiente:

```
mv apuntes alumno
mv documentos alumno
mv ejercicios alumno
```

- **Paso 3:** Movemos los archivos a su directorio correspondiente:

```
mv modulo1.doc alumno/apuntes
mv modulo2.doc alumno/apuntes
mv modulo3.doc alumno/apuntes
mv modulo1.html alumno/ejercicios
mv modulo2.html alumno/ejercicios
mv modulo3.html alumno/ejercicios
mv modulo4.html alumno/ejercicios
mv modulo5.html alumno/ejercicios
```

2. Se pide crear una carpeta por cada módulo, manteniendo la estructura de subcarpetas, de la siguiente forma:

- **Paso 1:** Creamos las nuevas carpetas (hacemos lo mismo con el resto de los módulos):

```
mkdir modulo1
mkdir apuntes
mkdir documentos
mkdir ejercicios
```

- **Paso 2:** Movemos los directorios a su estructura correspondiente (hacemos lo mismo con el resto de los módulos):

```
mv modulo1 alumno
mv apuntes modulo1
mv documentos modulo1
mv ejercicios modulo1
```

- **Paso 3:** Movemos los archivos a su estructura correspondiente:

```
mv modulo1.doc alumno/modulo1/apuntes
mv modulo2.doc alumno/modulo2/apuntes
mv modulo3.doc alumno/modulo3/apuntes
mv modulo1.html alumno/modulo1/ejercicios
mv modulo2.html alumno/modulo2/ejercicios
mv modulo3.html alumno/modulo3/ejercicios
mv modulo4.html alumno/modulo4/ejercicios
mv modulo5.html alumno/modulo5/ejercicios
```

3. Se pide eliminar las carpetas obsoletas:

```
rm -r alumno/apuntes
rm -r alumno/documentos
rm -r alumnos/ejercicios
```

## Solución Ejercicio Propuesto (2)

Tomaremos como ejemplo la carpeta con la estructura que generamos en el ejercicio 1.

- **Paso 1.** Vamos a la carpeta contenedora:

```
cd alumno
```

- **Paso 2.** Inicializa git en la carpeta contenedora.

```
git init
```

- **Paso 3.** Agrega los archivos al stage.

```
git add .
```

- **Paso 4.** Genera el primer commit.

```
git commit -m "initial commit"
```

- **Paso 5.** Muestra el status.

```
git status
```