

Terminal, Git, GitHub y GitHub Pages (Parte I)

Terminal

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

Además, aprenderás a subir tu sitio a un espacio gratuito llamado **GitHub Pages**. También, obtendremos un dominio gratuito para asignarlo a nuestro sitio.

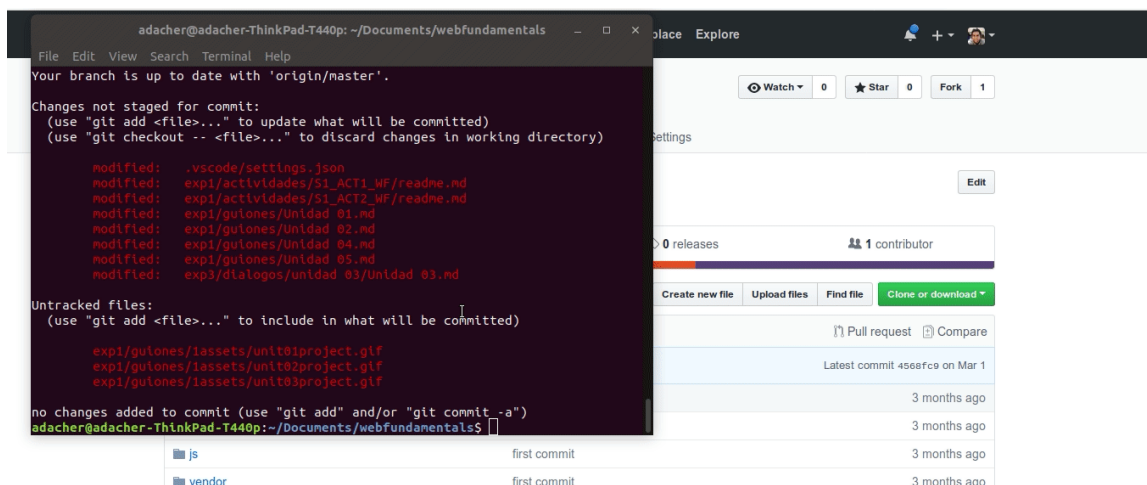


Imagen 1: Proyecto en GitHub Pages

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

Si estas trabajando en un sistema operativo **Linux** o **OSX** (Mac), ya tienes instalada la terminal. En el caso de que estés ocupando Windows, tendrás que bajar un programa especial.

¿Qué es 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.

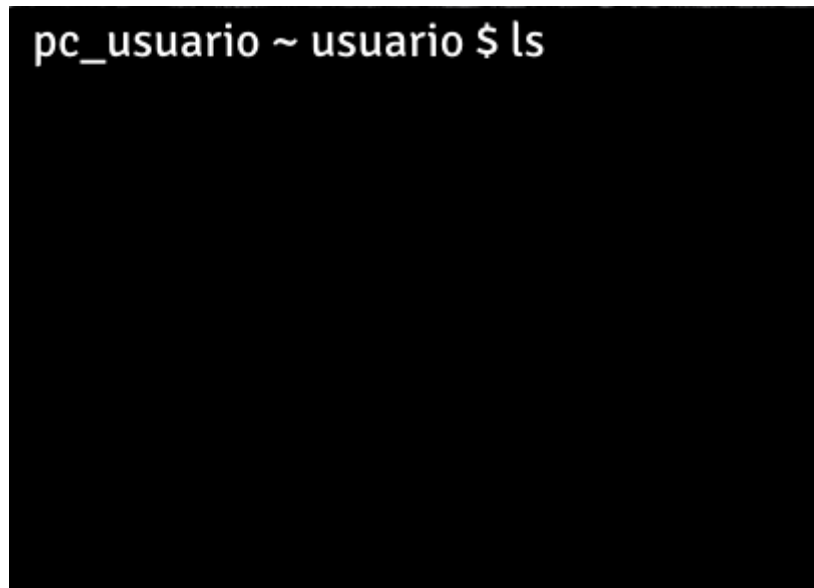


Imagen 2: Terminal

Ahora, aprenderemos a usar el terminal.

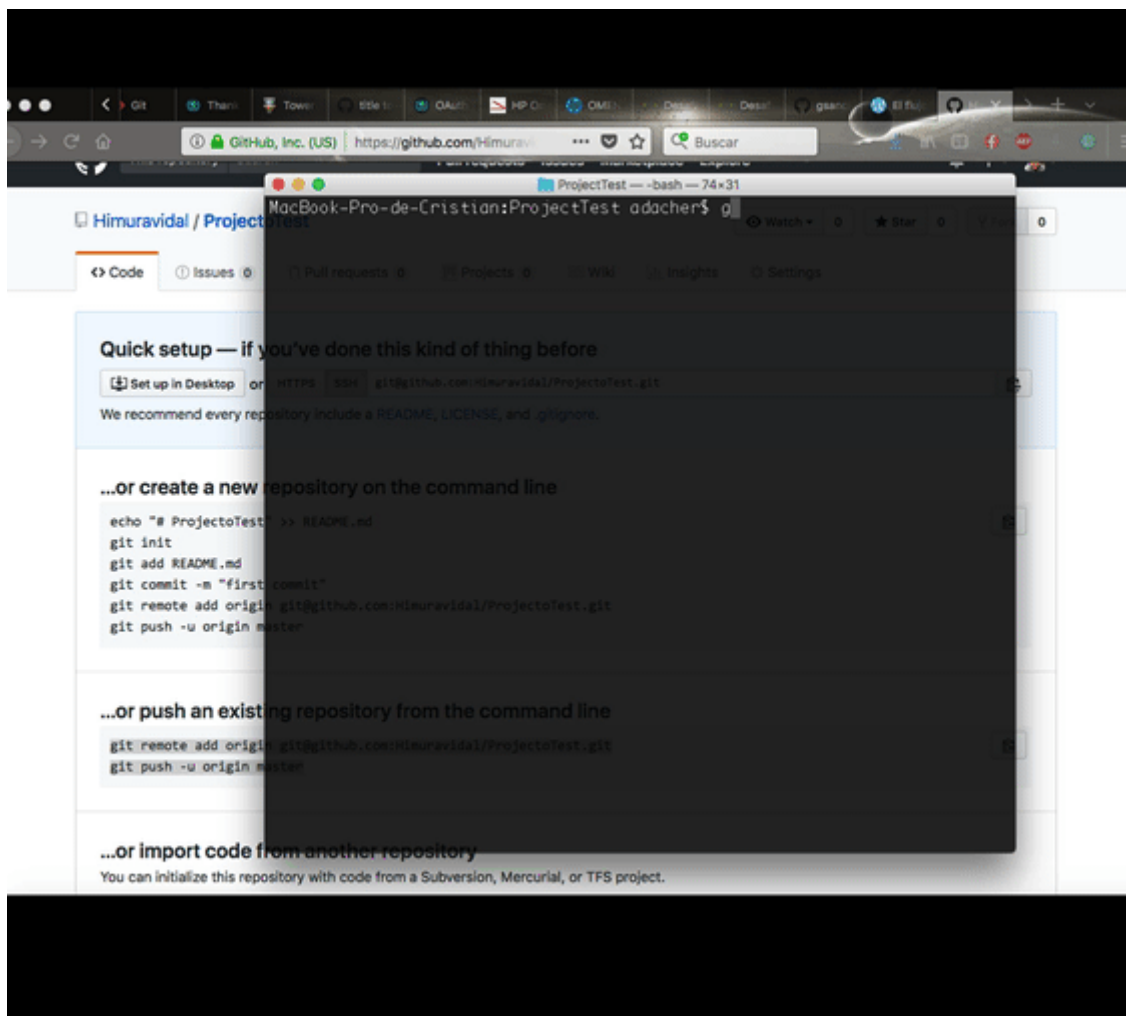


Imagen 3: Utilizar el Terminal

Como ya lo hablamos anteriormente, si estas 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:

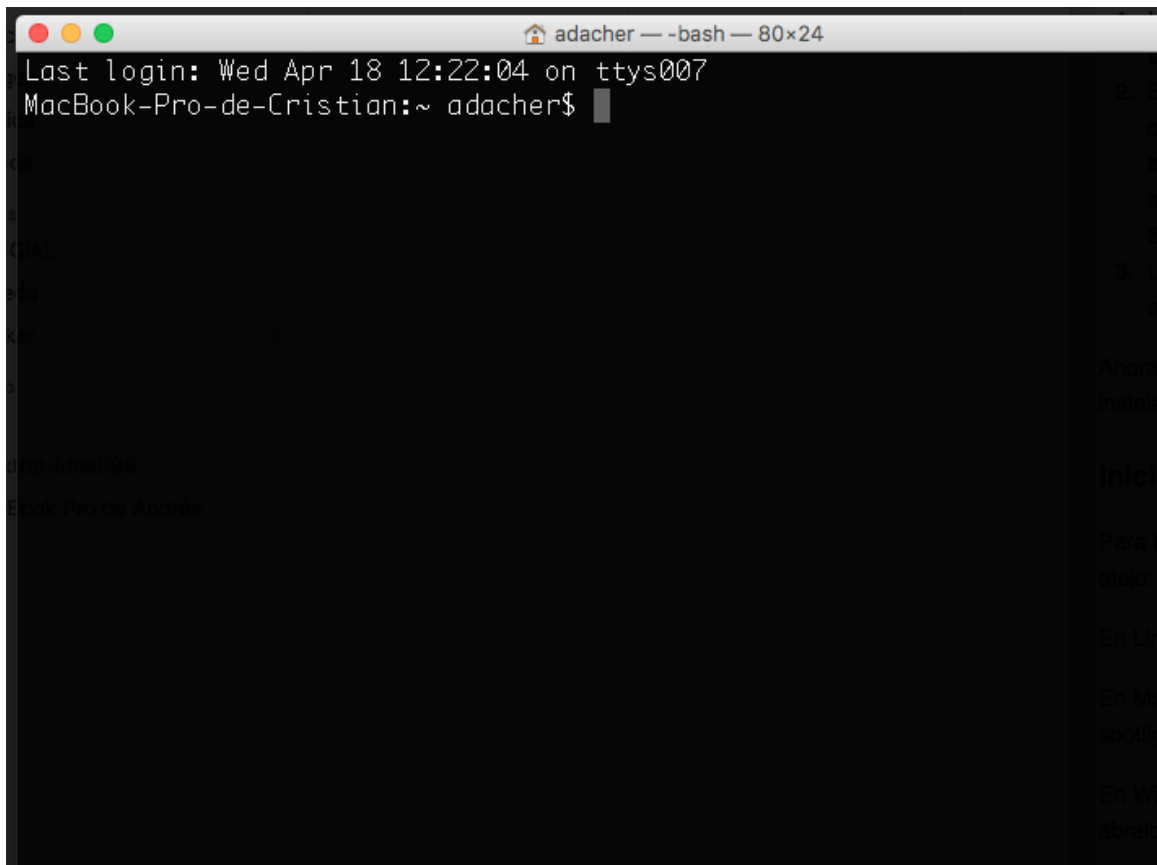


Imagen 4: Iniciar Terminal

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.

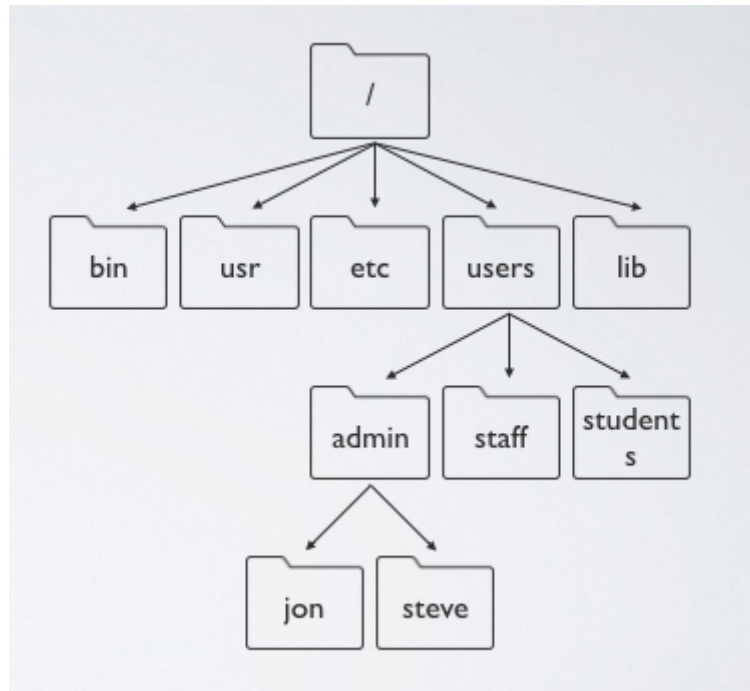


Imagen 5: Directorios

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 cuál estés trabajando, así que puede que en tu computador no veas los mismos directorios.

Conocer en que directorio estamos

Una de las cosas mas importantes que necesitamos saber al trabajar con la terminal, es conocer en que 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$
```

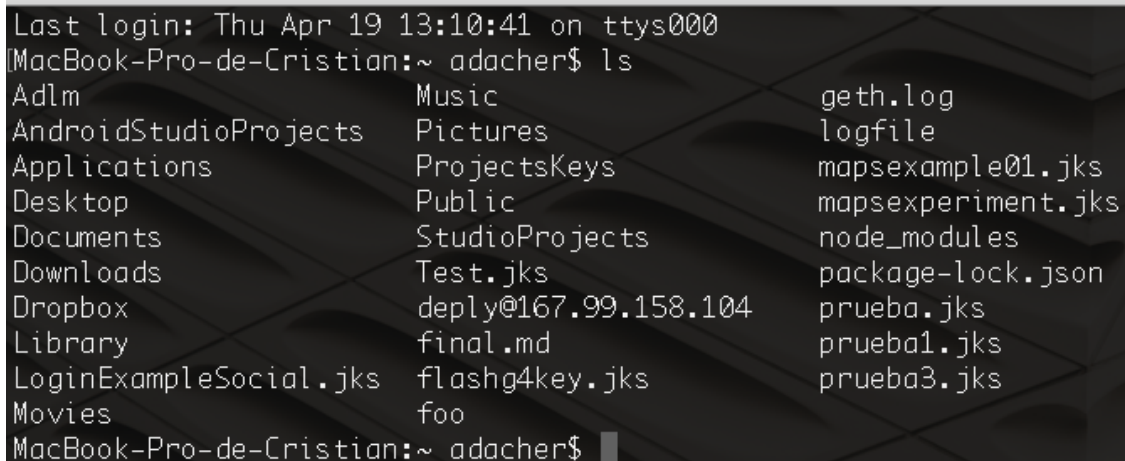
Imagen 6: Conocer en qué directorio estamos

En tu consola se imprimirá la ruta en la cuál 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.

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 7: Listar archivos

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:

```
.dropbox
.emulator_console_auth_token
.gem
.gemrc
.gitconfig
.gnupg
.gradle
.idea
.inputrc
.irb-history
.lessshst
.local
.mkshrc
.netrc
.node-gyp
.node_repl_history
.npm
.nvm
.odoorc
.oracle_jre_usage
.pencil
.profile
.pry_history
.pseint
.psql_history
Downloads
Dropbox
Library
LoginExampleSocial.jks
Movies
Music
Pictures
ProjectsKeys
Public
StudioProjects
Test.jks
deply@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 8: Visualizar archivos ocultos

Un listado de directorios y archivos ocultos, estos comienzan con un `.` antes de su nombre.

Comandos de navegación entre directorios

Ahora que sabemos en que 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, bastara 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](#).

Manejando archivos y carpetas (Parte I)

Aprenderemos a crear y borrar archivos y directorios desde el terminal.

Creando directorios

Lo primero que haremos será crear un nuevo directorio con un comando llamado `mkdir`, que significa make directory. Primero 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 con el nombre que lo acompañe, de modo que ahora revisaremos los archivos con `ls` y aparecerá el nuevo directorio.

Creando archivos

Cuando necesitemos crear un archivo desde la terminal utilizaremos el comando `touch`. Con este comando podremos crear un archivo vacío.

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. Crearemos un archivo HTML entonces. Escribiremos:

```
touch index.html
```

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

Copiando archivos

Ahora que ya sabemos crear un directorio y un archivo, aprenderemos a copiarlos.

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

Practiquemos. En nuestra carpeta raíz ya teníamos la carpeta el archivo `index.html` y lo vamos a copiar dentro del directorio que habíamos creado, o sea en la carpeta `proyecto1`.

Utilizaremos el comando:

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


En este caso, el primer argumento es el nombre del archivo existente y el segundo es la ruta donde copiaremos el archivo. Si ingresamos a `proyecto1`, a través de `cd proyecto1` y listamos los archivos. con `ls`, debería aparecer nuestro archivo copiado.

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.

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
```

Copiar un directorio

Con el comando `cp` también podremos copiar un directorio.

Para aprender cómo, en nuestra terminal volveremos a nuestro home con `cd`, crearemos un nuevo directorio llamado `assets` con:

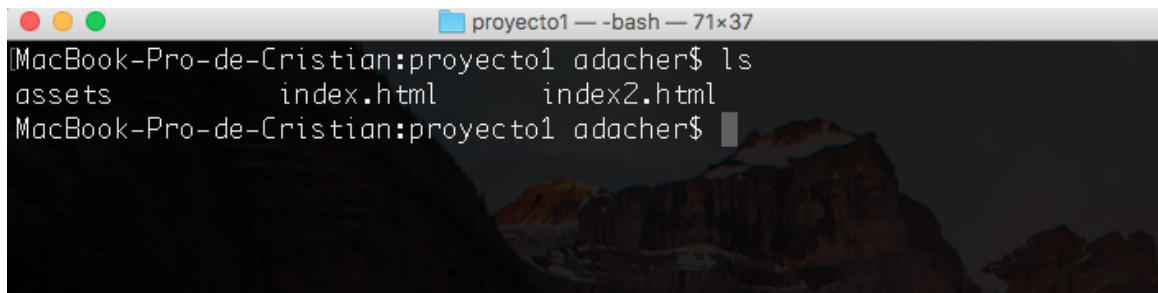
```
mkdir assets
```

A continuación, copiaremos esta carpeta al interior de `proyecto1` de la siguiente forma:

```
cp -r assets proyecto1
```

Al comando llamado `cp` le añadimos la opción `-r`

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

A screenshot of a macOS terminal window titled "proyecto1 — -bash — 71x37". The terminal shows the command "ls" being executed in the directory "MacBook-Pro-de-Cristian:proyecto1 adacher". The output of the command is "assets", "index.html", and "index2.html". The terminal background features a dark, mountainous landscape with snow-capped peaks under a twilight sky. The window has standard macOS window controls (red, yellow, green buttons) in the top-left corner.

```
MacBook-Pro-de-Cristian:proyecto1 adacher$ ls
assets      index.html  index2.html
MacBook-Pro-de-Cristian:proyecto1 adacher$
```

Imagen 9: Copiar un directorio

Manejando archivos y carpetas (Parte II)

Mover archivos y directorios

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

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/
```

Ingresaremos a `proyecto1` y listaremos 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.

Para mover un directorio utilizaremos la misma sintaxis que el comando `cp`. Para comprobarlo, 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
```

Para comprobar que todo haya funcionado correctamente, ingresaremos a la carpeta `assets` del `proyecto1` con

```
cd proyecto1/assets
```

y listaremos los archivos con `ls`.

Borrando 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.

Ingreseemos a nuestra carpeta `proyecto1` y eliminaremos nuestro archivo `readme.txt` con el siguiente comando:

```
rm readme.txt
```

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.

Veámoslo con un ejemplo. Volvamos a nuestro home con `cd` y escribamos el siguiente comando.

```
rm -r proyecto1
```

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 que carpetas estás y además crear, copiar, mover y eliminar a través de la terminal.

Git

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 manejar proyectos no importando su tamaño.

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

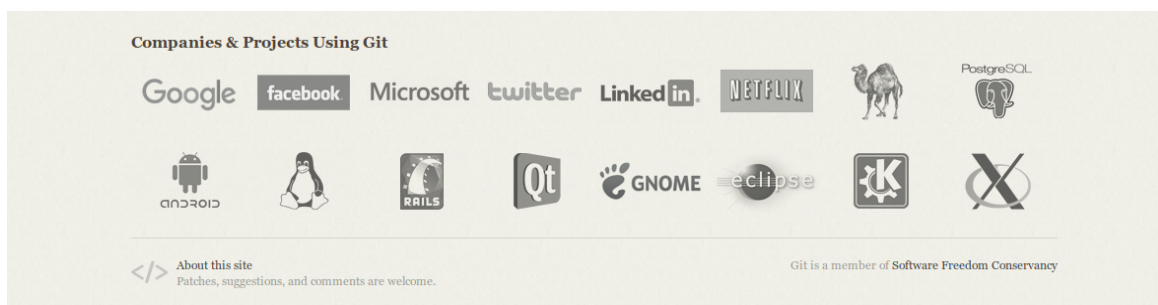


Imagen 10: Compañías y proyectos que utilizan Git

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](#))

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 a 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.

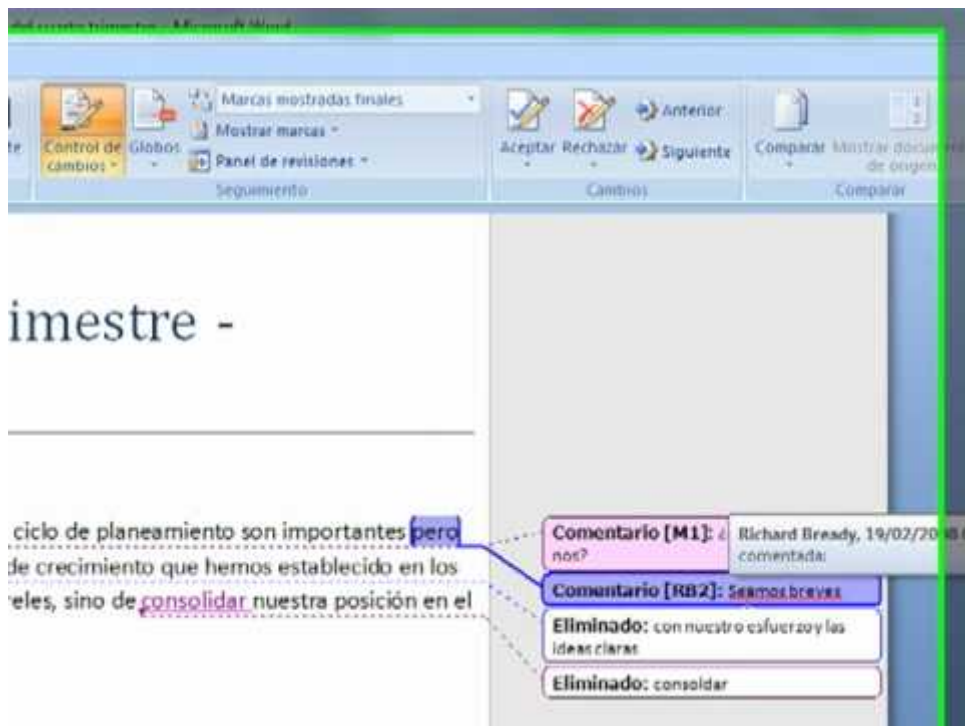


Imagen 11: Control de versiones

¿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**.

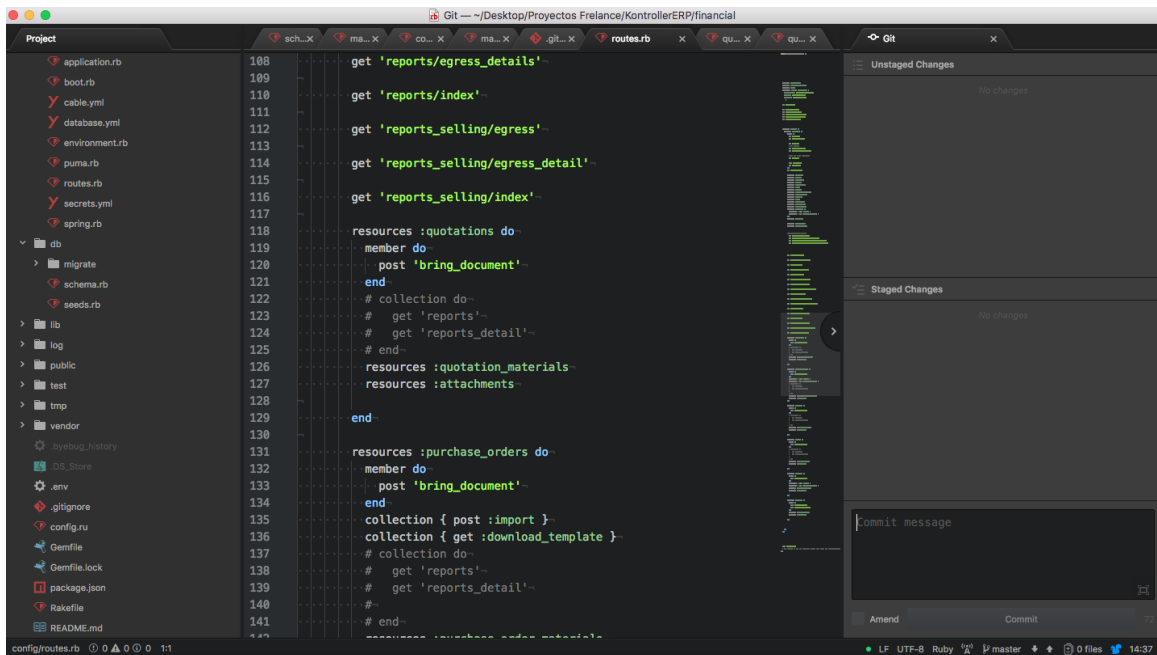


Imagen 12: Editor ATOM

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

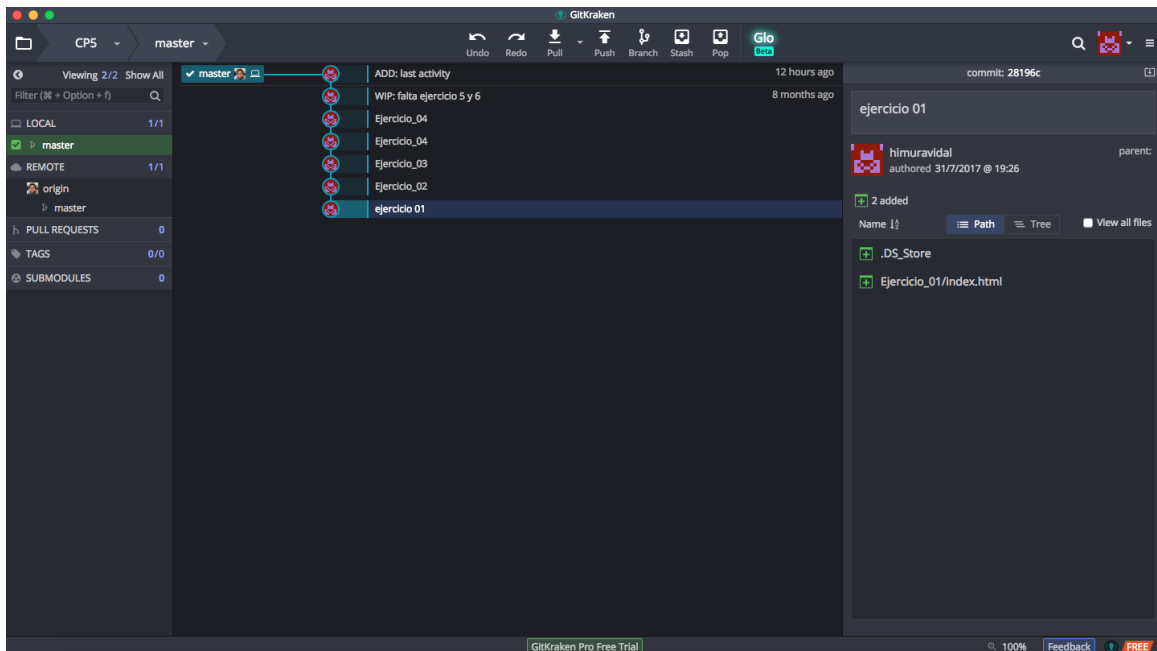


Imagen 13: Editor GitKraken

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.

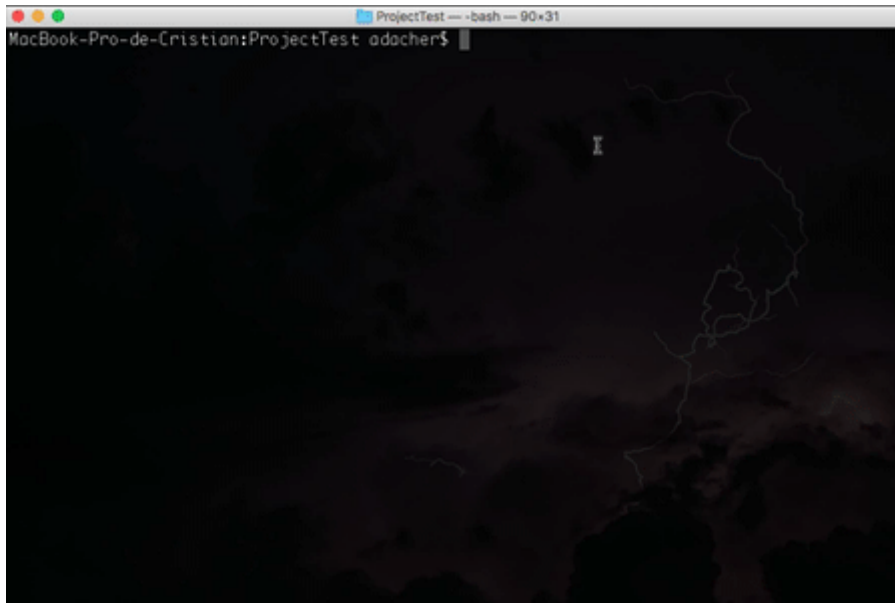


Imagen 14: Terminal

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 esta instalado, veremos que el terminal nos muestra algo como:

```
git version 2.14.3
```

Instalando git

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 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 de 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 como 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

```
git init
```

en la carpeta del proyecto.

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` sólo lo debemos realizar una vez por proyecto.

Usando git

Para entender como funciona **git** utilizaremos la metáfora de una mudanza.

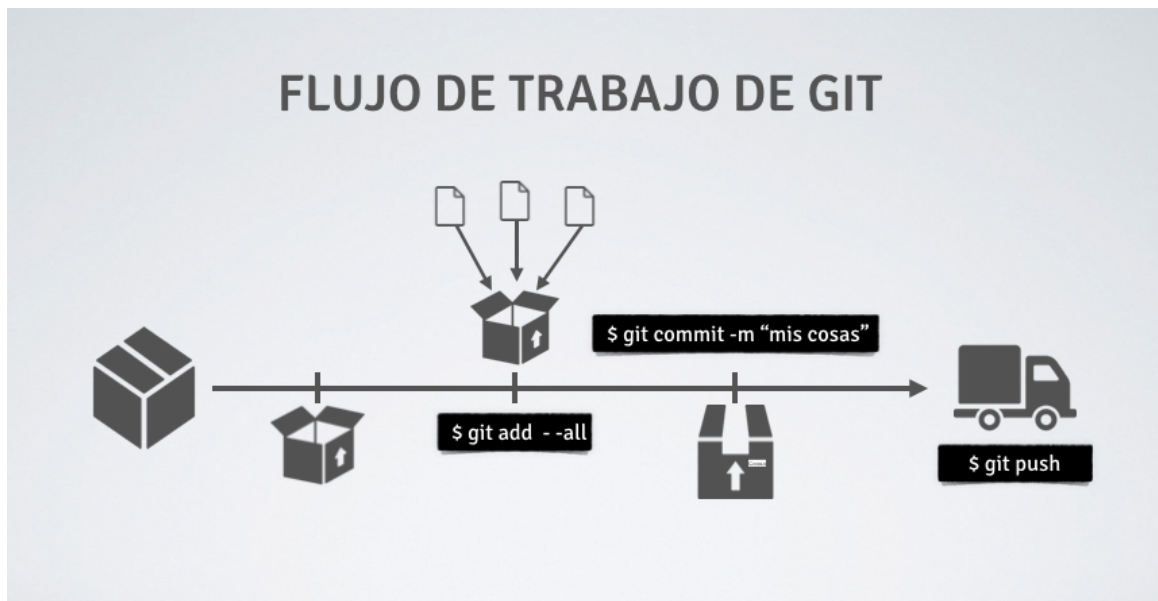


Imagen 15: Flujo de trabajo de Git

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
```

o

```
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 via comando

```
git push
```

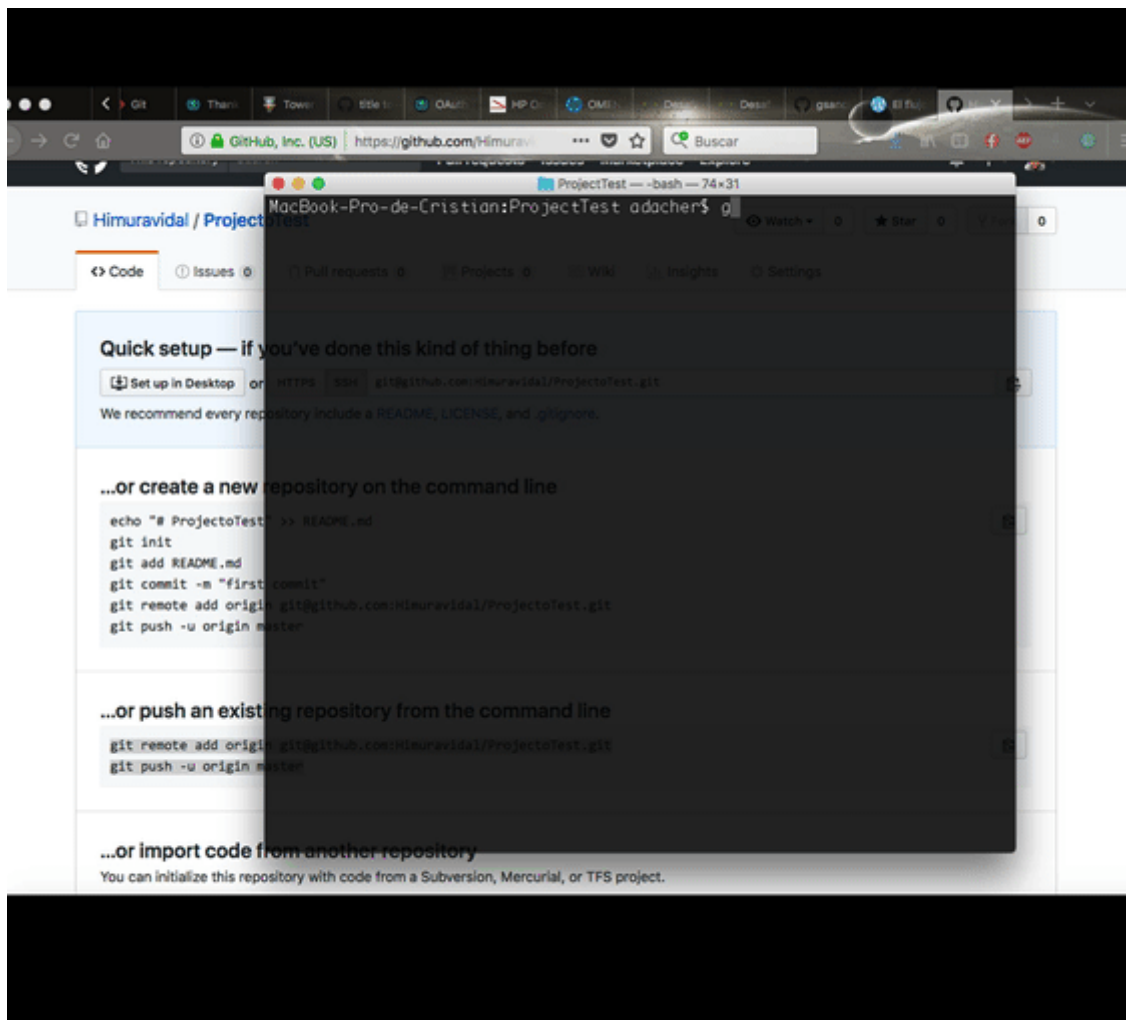


Imagen 16: Git push

¿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
```

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

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

Seguiremos los siguientes pasos:

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

```
cd Desktop
```

- Ingresamos a la carpeta `meet&coffee`

```
cd meet\&coffee
```

- Podemos corroborar que estamos dentro de ella con

```
pwd
```

- Inicializamos git dentro der la carpeta con:

```
git init
```

Observaremos un mensaje indicando de que se inicio git.

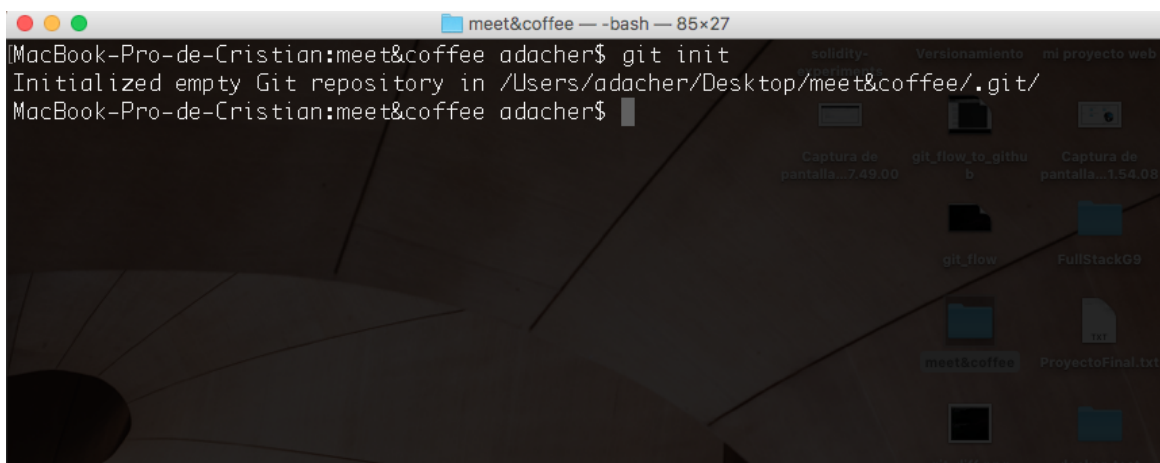


Imagen 17: Inicio de Git

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

Agregando los cambios

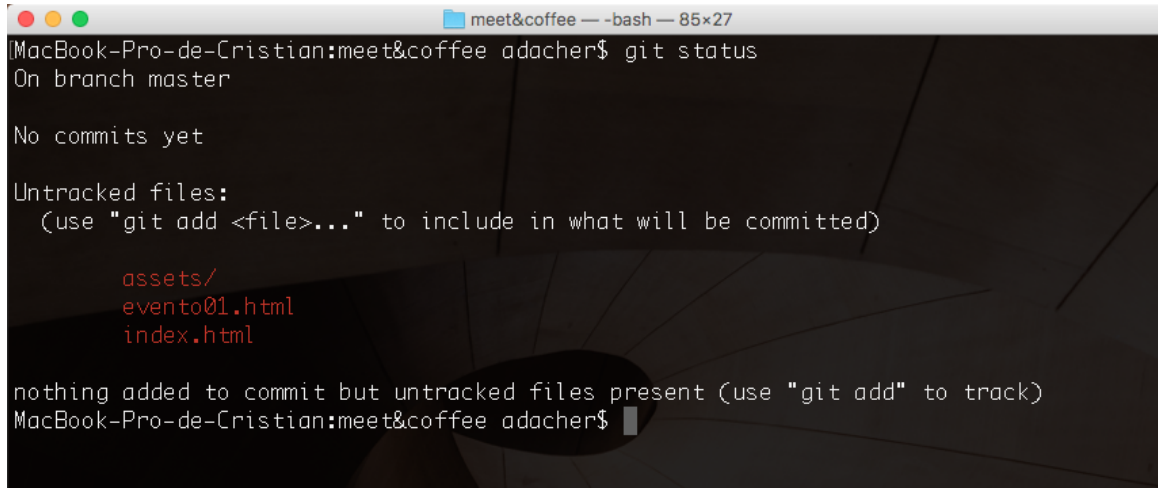
Recordemos cómo agregar los cambios.

Para ello se ocupa `git add` seguido de todos los archivos que queremos agregar.

Con el comando:

```
git status
```

veremos un mensaje del tipo:



```
MacBook-Pro-de-Cristian:meet&coffee adacher$ git status
On branch master

No commits yet

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

        assets/
        evento01.html
        index.html

nothing added to commit but untracked files present (use "git add" to track)
MacBook-Pro-de-Cristian:meet&coffee adacher$
```

Imagen 18: Agregando cambios

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. 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:

```
MacBook-Pro-de-Cristian:meet&coffee adacher$ 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/
    evento01.html

MacBook-Pro-de-Cristian:meet&coffee adacher$
```

Imagen 19: Git status

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

Para añadir el resto de archivos, vamos a utilizar el comando

```
git add .
```

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

```
MacBook-Pro-de-Cristian:meet&coffee adacher$ git add .
MacBook-Pro-de-Cristian:meet&coffee adacher$ git status
On branch master

No commits yet

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

    new file:   assets/css/style.css
    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:   evento01.html
    new file:   index.html

MacBook-Pro-de-Cristian:meet&coffee adacher$
```

Imagen 20: Añadir archivos

Lo confirmaremos utilizando `git status`.

Y observamos que no queda ningún archivo por añadir.

Confirmando nuestro primer cambio

El mensaje nos dice es 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.

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.

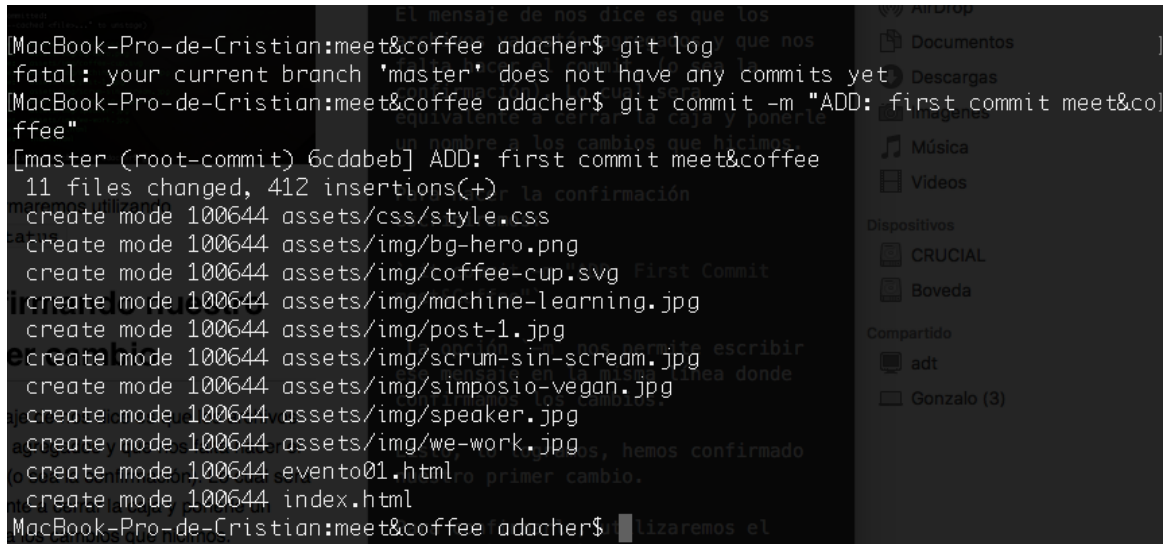


Imagen 21: Escribir mensaje

¡Listo! Lo logramos, hemos commitiado nuestra primeras versión.

Para asegurarnos, utilizaremos el comando `git log`.

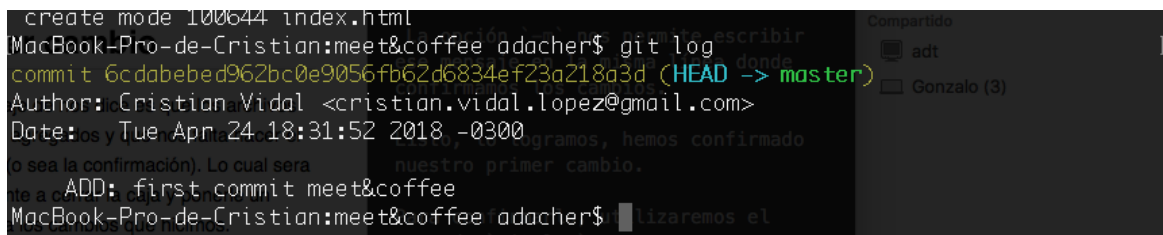


Imagen 22: Git log

Esto nos indicará cual 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.

Utilizando git en un proyecto (Parte II)

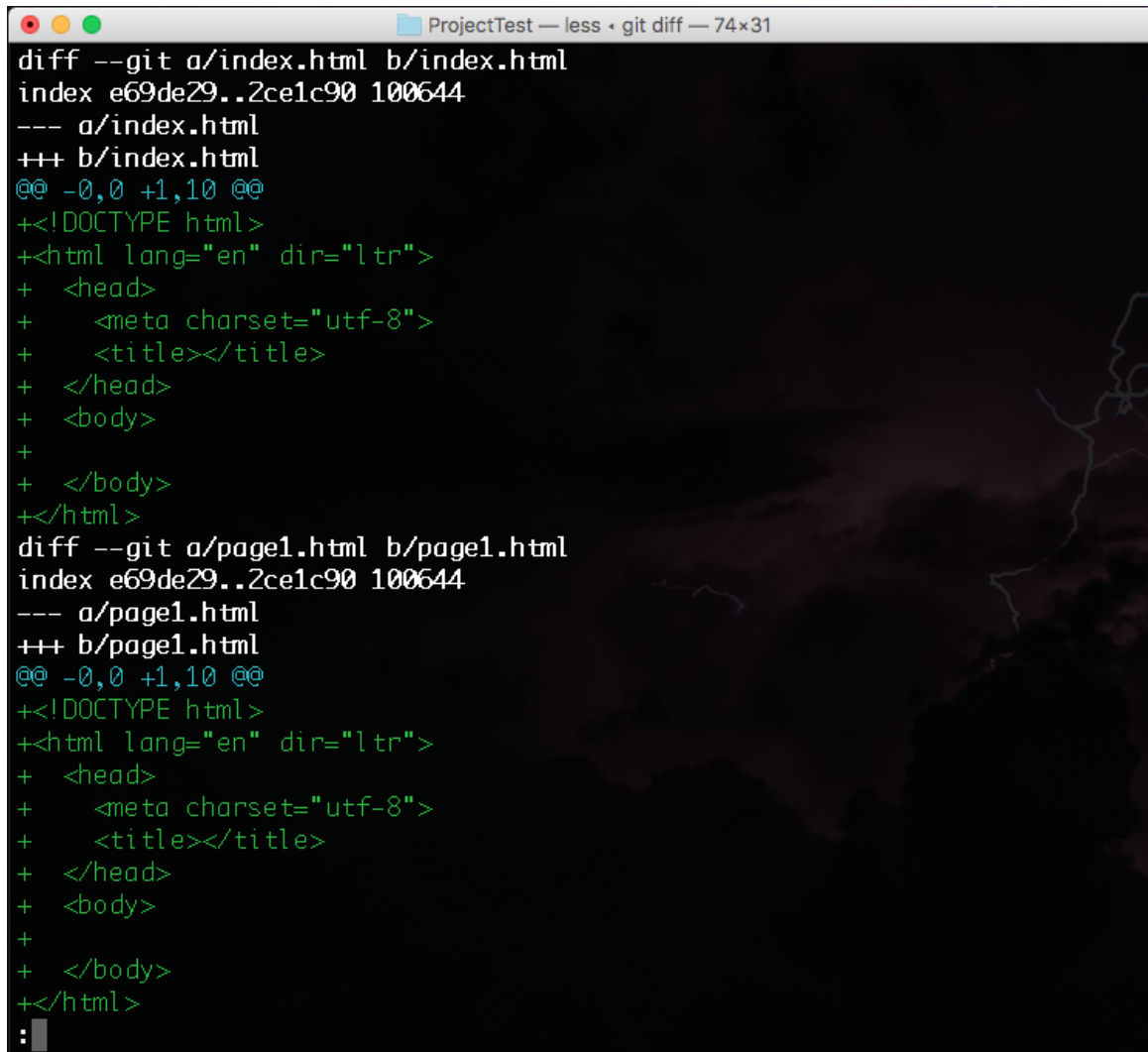
Gestionando los cambios

git permite de forma sencilla ver que cambios hemos hecho contra la revisión anterior.

Para probar esto vamos a introducir otros cambios.

(Agregar texto al html)

Si queremos ver los cambios introducidos en la consola, podemos usar un comando llamado `git diff` para ver que ha cambiado.



```
diff --git a/index.html b/index.html
index e69de29..2ce1c90 100644
--- a/index.html
+++ b/index.html
@@ -0,0 +1,10 @@
+<!DOCTYPE html>
+<html lang="en" dir="ltr">
+  <head>
+    <meta charset="utf-8">
+    <title></title>
+  </head>
+  <body>
+
+  </body>
+</html>
diff --git a/page1.html b/page1.html
index e69de29..2ce1c90 100644
--- a/page1.html
+++ b/page1.html
@@ -0,0 +1,10 @@
+<!DOCTYPE html>
+<html lang="en" dir="ltr">
+  <head>
+    <meta charset="utf-8">
+    <title></title>
+  </head>
+  <body>
+
+  </body>
+</html>
:
```

Imagen 23: Gestionar cambios

`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 que archivos se han modificado.

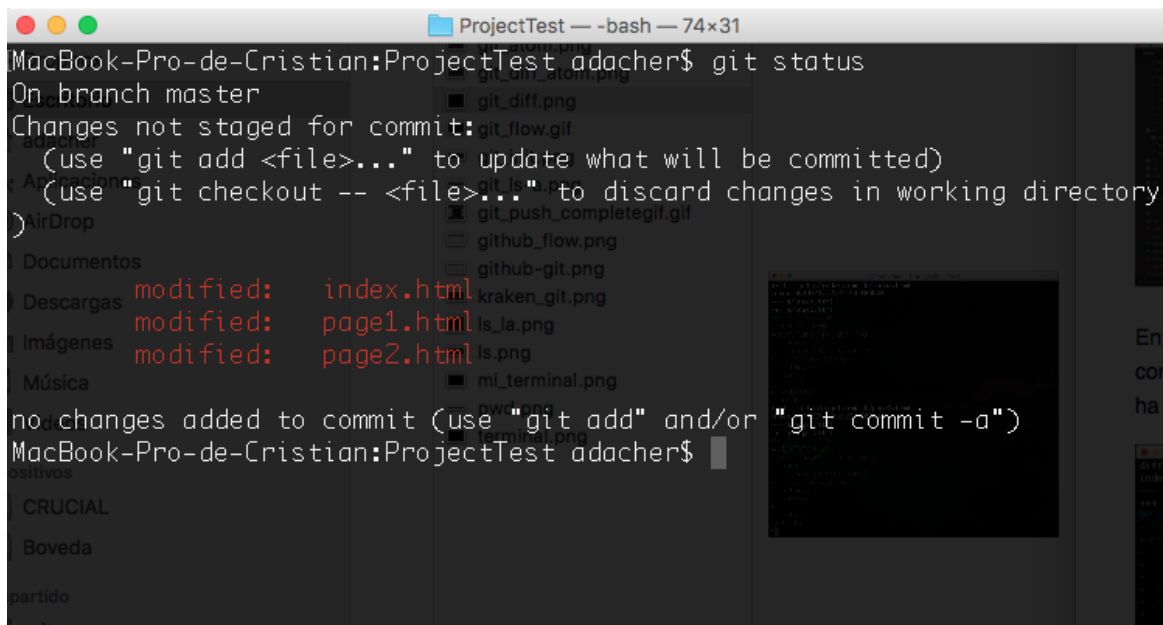


Imagen 24: Resumen de archivos modificados

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"
```

y si revisamos con `git status` veremos que ya no hay información nueva que confirmar.