GITHUB

## Trabajando de manera local con Git

Existen dos maneras de obtener un proyecto Git: creando un proyecto Git en un repositorio o clonando un repositorio ya existente mediante la orden **git clone ruta-git-del-repositorio**.

(**git clone git@github.com:BeaGAGA/beagaga.github.io.git)**

O

git init

Esta orden crea un**subdirectorio que contiene todos los archivos necesarios**. Todavía no tenemos dentro ningún archivo bajo seguimiento. Vamos a crear un archivo vacío llamado: **README.md**

La extensión **md** nos indica que es un archivo escrito en **Markdown**, lenguaje de marcas utilizado habitualmente para escribir documentación y que veremos en el último apartado del tema.

Echo “repasando Git” > DEADME.md

git status

git add

git diff orden que te devuelve una lista de los nuevos archivos, archivos eliminados y exactamente todas las diferencias de las modificaciones. Importante para comparar los cambios entre nuestro repositorio de trabajo y el área de preparación.

git commmit -m “Add README”

git rm : para eliminar un archivo de Git . Lo elimina de nuestros archivos de seguimiento (área preparación) y después confirma cambios.

git log: permite ver el histórico de nuestros cambios

Los proyectos en los que se utiliza un sistema de control de versiones utilizan ramas. Las**ramas** son una prestación fundamental de estos sistemas, ya que permiten trabajar en paralelo sobre una nueva prestación (por ejemplo, implementar un nuevo comportamiento de un botón) y, una vez esté terminada, fusionarla (mergearla) con la rama principal, llamada **master**. Este trabajo en paralelo no afecta a la versión principal del proyecto.

## Trabajando con repositorios remotos

Cuando nuestro proyecto se encuentra en un estado que queremos compartir, tenemos que enviarlo a un repositorio remoto. El comando que nos permite hacer esto es **git push [nombre-remoto] [nombre-rama]. Ojo! Sólo funcionará si tenemos permisos de escritura y si nadie más ha enviado info.**

ENVIAR:

git push origin master

RECIBIR:

git pull

Git almacena instantáneas de nuestro proyecto cada vez que hacemos una confirmación (**commit**). **Una rama Git**es simplemente un **apuntador móvil apuntando a una de esas confirmaciones**.

La rama por defecto de Git es la rama **master**. Cuando realizamos el primer **commit** (la primera confirmación de los cambios) se crea esta rama principal **master**apuntando a dicha confirmación. La rama va avanzando con cada **commit** apuntando siempre a la última confirmación realizada.

Cuando creamos una **nueva rama, creamos un nuevo apuntador para poder moverlo libremente**.

Crear Nueva rama:

git branch test

Para desplazarnos a la nueva rama usarmos comando checkput:

git checkout test

Podemos trabajar en esta nueva rama **test**y realizar todos los cambios necesarios, añadir nuevas funcionalidades, etc. **Su comportamiento es el mismo** que todo lo que hemos aprendido para la rama **master**.

git checkout test

touch newFile.txt (creamos archivo de la rama)

Fusionar rama con el master: merge

git checkout master

git merge test

ls: listar contenidos del directorio actual

Con ls se puede comporbar que el archivo de la rama test se encuentra ahora también en master. No es proceso sencillo, puede generar conflictos.

Ignorando archivos en Git

En ocasiones nos puede interesar que Git no reconozca algunos archivos que tenemos en el directorio de trabajo para que nunca los añada al proyecto. Pueden ser notas, archivos de **log**, documentación o simplemente directorios donde realizamos pruebas.

Para este propósito existe el archivo **.gitignore**

**GitHub**

GitHub es una **plataforma de desarrollo colaborativo** que permite alojar proyectos utilizando Git como sistema de control de versiones. El código de los proyectos alojados en esta plataforma se almacena de manera pública, aunque, con una cuenta de pago, los usuarios pueden tener repositorios privados. En GitHub se almacenan**repositorios remotos**.

1. Lo primero que debemos hacer para empezar a trabajar con GitHub es **crear una cuenta en la plataforma**.
2. El siguiente paso es **crear nuestra clave SSH**. Esta clave permite la autenticación de los usuarios a través de claves SSH. Las claves se almacenan en el directorio **~/.ssh**del usuario.  OK!!
3. Existen**dos maneras de contribuir a un proyecto existente en GitHub**:
   1. **con permisos de escritura (como hemos trabajado en el curso).** Si tenemos permiso de escritura sobre un repositorio y queremos enviar nuestros cambios al repositorio remoto el flujo sería el siguiente:
4. Añadir los archivos al área de seguimiento: **git add README.md**
5. Confirmar los cambios: **git commit -m "Update README"**
6. Enviar los cambios al repositorio remoto: **git push origin master**
7. **Git pull:** actualizar nuestro repositorio
   1. **sin permisos de escritura.** Si queremos contribuir a un proyecto en GitHub en el que no tenemos permisos de escritura el procedimiento para lograrlo es mediante la realización de un **fork**. De esta forma, los proyectos no necesitan añadir colaboradores con acceso de escritura (**push**). Los usuarios pueden bifurcar (***forkear***) un proyecto, enviar sus propios cambios a su copia y luego remitir esos cambios al repositorio original para su aprobación mediante una **Pull Request**. Flujo de trabajo:
      1. Se crea una rama a partir de master.
      2. Se realizan algunos commits hacia esa rama.
      3. Se envía esa rama hacia tu copia (fork) del proyecto.
      4. Abres un **Pull Request** en GitHub.
      5. Se participa en la discusión asociada y, opcionalmente, se realizan nuevos **commits**.
      6. El propietario del proyecto original cierra el Pull Request, bien fusionando la rama con tus cambios o bien rechazándolos.

# Markdown

Markdown es un**lenguaje de marcas** creado en 2004. Este lenguaje tiene como objetivo permitir crear contenido de una manera sencilla, mediante una sintaxis legible para el autor y que en todo momento mantenga un diseño legible y sencillo.

Markdown está **escrito en texto plano**. Este lenguaje nos permite crear aquellos elementos básicos que se utilizan en HTML para dar formato a un texto –**encabezados**(**<h1>**), **negrita** (**<b>**), **itálica** (**<i>)**, **links** (**<a/>**) o **imágenes** (**<img/>**)– de una manera rápida y sencilla

Si HTML es el lenguaje para publicar documentos web, Markdown es el lenguaje para escribir y leer documentos. Markdown es tan sencillo de utilizar y leer que muchas páginas web procesan sus contenidos (por ejemplo, una entrada de un blog) en este lenguaje y lo transforman a HTML. Por lo tanto, **Markdown necesita un intérprete o procesador** para convertir este archivo de texto plano en HTML, en Pdf o Epub.

GitHub permiten escribir en Markdown. Este lenguaje también es muy utilizado para tomar apuntes en clases, reuniones o incluso escribir un libro o manual, ya que es fácil exportarlo a otros formatos.

**ara crear una palabra o conjunto de palabras en itálica** utilizamos el guion bajo (**\_**).

**\*\*Esta\*\* palabra estará en \*\*negrita\*\***

crear encabezados (**<h1>**) en seis niveles gracias al símbolo #

* Incluyendo el texto del link entre corchetes: **[Enlace]**
* Y la dirección a la que tiene que dirigir entre paréntesis:**https://github.com**
* utilizar el símbolo almohadilla (#) para convertir el enlace en un **link local,** una referencia a otra sección del documento
* **Para las imágenes a los links les antecede el símbolo de exclamación** (**!**).



Listas:



**Cualquier elemento creado en Markdown se convierte en un párrafo** (**<p></p>**) salvo que especifiquemos lo contrario. Aunque produzcamos un salto de línea manualmente, Markdown no lo interpreta y muestra todo el contenido en una sola línea. Para producir un salto de línea debemos incluir al menos dos espacios al final de la clase.

# Tu porfolio en GitHub Pages (II)

Vamos a crear un repositorio en GitHub que se llame de la siguiente manera: **nombreusuario.github.io**

**NO** hay que marcar la opción de inicializar el repositorio con un README.

A continuación vamos a clonar este repositorio vacío (solo tiene el directorio **.git** en su interior) en nuestro equipo. Dentro de nuestro repositorio hacemos clic en la opción Clone or Download. Si no vemos la opción Clone with SSH, seleccionamos Use SSH. Copiamos la url que nos ofrece GitHub o pinchamos en el icono copiar (figura 1).

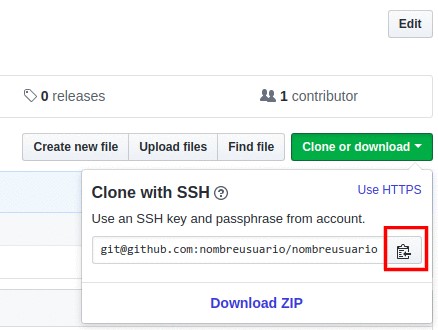


Figura 1. Clonando un repositorio en Github.

$ git clone git@github.com:nombreusuario/nombreusuario.github.io.git

Esta orden crea una carpeta, en el directorio desde el que estemos ejecutando esta orden, con el nombre nombreusuario.github.io. A continuación copiamos los archivos en el interior de esta nueva carpeta, sean los archivos de la práctica del tema «Hojas de estilo» o un sencillo **index.html** con una lista con nuestros datos.

Los siguientes pasos serán propios del trabajo con Git. El primer paso será añadir nuestros archivos al área de seguimiento:

$ git add --all

El argumento **--all** nos permite poner todo el contenido de la carpeta, de manera recursiva, bajo seguimiento. Ahora todo nuestro proyecto se encuentra en **staged**.

Podemos ir trabajando sobre los archivos y cuando deseemos crear una instantánea de nuestro proyecto realizamos una confirmación de los cambios mediante:

$ git commit -m "Initial commit"

*Creamos el primer commit junto con un mensaje*

Por último, cuando queramos enviar nuestros cambios al repositorio remoto, a GitHub, tenemos que hacer un envío mediante la orden **push**.

$ git push origin master

(Una vez hayamos especificado qué rama es la que queremos enviar a nuestro proyecto, los siguientes **push** que hagamos pueden ser simplemente mediante la órden **git push**).

Cuando hayamos enviado nuestros cambios al repositorio remoto nuestra página estará *online* al cabo de unos minutos en la dirección **nombreusuario.github.io** donde **nombreusuario** es nuestro nombre de usuario en GitHub, el mismo que tenemos en nuestro perfil. Todos los usuarios tienen también un perfil público en GitHub donde se ven todas sus contribuciones (<https://github.com/nombreusuario>).

**Este repositorio tiene que incluir un archivo README.md** (en mayúsculas) escrito en Markdown y que incluya varios elementos Markdown. Los archivos README son una pequeña descripción de los repositorios. En este caso deberá incluir un encabezado descriptivo (por ejemplo, vuestro nombre: Frankie Ruiz porfolio), una pequeña descripción de tu perfil (dos líneas), link a algún trabajo que hayas publicado, etc.

Si en algún momento modificamos algún archivo desde GitHub u algún otro equipo y luego queremos actualizar nuestro repositorio local, también debemos especificar la rama con la que queremos actualizar nuestro repositorio.

$ git pull origin master

GitHub también permite editar los archivos desde la interfaz web.

1. Create a new repository: nombre usuario + github.io
2. Empezar a trabajar:
   1. Clonar este repositorio: copiar la dirección del repositorio (GitHub)
   2. desde nuestro equipo
3. Abrir terminal o consola y localizamos dónde estamos

MacBook-Pro-de-Beatriz:~ BeatrizGalvez$ pwd

/Users/BeatrizGalvez

MacBook-Pro-de-Beatriz:~ BeatrizGalvez$ cd /Users/BeatrizGalvez/Desktop/pruebas-terminal/pruebaexamen

MacBook-Pro-de-Beatriz:pruebaexamen BeatrizGalvez$ ls

PracticaWeb.html

1. creamos una carpeta “pruebaexamen”:

MacBook-Pro-de-Beatriz:~ BeatrizGalvez$ mkdir pruebaexamen

1. Creamos un archivo con contenido escrito en Markdown

MacBook-Pro-de-Beatriz:pruebaexamen BeatrizGalvez$ echo "#Prueba Examen" >> README.md

1. Iniciamos proyecto git init

MacBook-Pro-de-Beatriz:pruebaexamen BeatrizGalvez$ git init

1. Añadimos los archivos al área de seguimiento

MacBook-Pro-de-Beatriz:pruebaexamen BeatrizGalvez$ git add README.md

1. Hacemos un commit

MacBook-Pro-de-Beatriz:pruebaexamen BeatrizGalvez$ git commit -m "Initial commit"

1. Añadimos el repositorio remoto al directorio de git

MacBook-Pro-de-Beatriz:pruebaexamen BeatrizGalvez$ git remote add origin git@github.com:BeaGAGA/pruebaexamen.github.io.git

1. Enviamos los cambios al repositorio remoto

MacBook-Pro-de-Beatriz:pruebaexamen BeatrizGalvez$ git push -u origin master

1. Incluir un sencillo archivo index.html

MacBook-Pro-de-Beatriz:pruebaexamen BeatrizGalvez$ echo > PracticaWeb.html

MacBook-Pro-de-Beatriz:pruebaexamen BeatrizGalvez$ git add PracticaWeb.html

MacBook-Pro-de-Beatriz:pruebaexamen BeatrizGalvez$ git commit -m "Initial commit"

MacBook-Pro-de-Beatriz:pruebaexamen BeatrizGalvez$ git push -u origin master

MacBook-Pro-de-Beatriz:pruebaexamen BeatrizGalvez$ ls

PracticaWeb.html README.md

## ESCRIBIR EN MARKDOWN

1. 1.ABRIR EL ARCHIVO .MD EN ATOM
2. 2. Crear y modificar el contenido

3. Añadirlo al repositorio git add DEADME.md

4. Git commit -m “modification”

5. git push -u origin master

**Markdowncheatsheet**

►Negritas: \*\*esto es negrita\*\*

►Itálica: \*esto es itálica\*ó\_esto también es itálica\_

►Encabezados: # Equivale a un h1, ## Equivale a un h2

►Enlaces: [texto-enlace](dirección-enlace)

►Imágenes: ![texto-se-muestra-si-no-carga](ruta-imagen)

►Listas: es fundamental respectar la indentación

►Desordenadas: un símbolo ., -, +ó\*por cada ítem

►Ordenadas: con números 1., 2., 1., cambia automáticamente

►Párrafos: cualquier texto es un párrafo. Para crear un salto de línea hacen falta dos espacios al final de una línea

# 

Trabajando con ramas en Git

**Una rama Git**es simplemente un **apuntador móvil apuntando a una de esas confirmaciones**.

La rama por defecto de Git es la rama **master**. Cuando realizamos el primer **commit** (la primera confirmación de los cambios) se crea esta rama principal **master**apuntando a dicha confirmación. La rama va avanzando con cada **commit** apuntando siempre a la última confirmación realizada.

Cuando creamos una **nueva rama, creamos un nuevo apuntador para poder moverlo libremente**. El apuntador especial HEAD apunta a la rama local en la que estemos en ese momento. En nuestro caso, y si no hemos hecho ninguna modificación, es la rama máster.

1. Para crear una nueva rama:

Git branch test

1. Para cambiarnos a la rama “test”:

Git checkout test

1. Creamos un texto plano:

Touch newFile.txt

Podemos trabajar en esta nueva rama **test**y realizar todos los cambios necesarios, añadir nuevas funcionalidades, etc. **Su comportamiento es el mismo** que todo lo que hemos aprendido para la rama **master**. Realizamos cambios, los añadimos al área de seguimiento (**staged**) y por último confirmamos los cambios (**commit**).

Una vez hemos terminado de desarrollar una nueva funcionalidad o arreglado un problema **el siguiente paso será fusionar esta rama** **test** con **master**. Este acto se conoce como **merge**. Necesitamos volver a **master**, pero, antes,**todos los cambios**en la rama que queremos fusionar tienen que estar **confirmados** (**commit**). Volvemos a master con:

1. Volvemos a la rama master

Git cheackout master

1. Mezclamos contenidos

Git merge test

En este ejemplo habíamos creado en la rama test un archivo llamado **newFile.txt.** Después de hacer el **merge**, si hacemos un **ls** para listar los contenidos del directorio actual, veremos que este archivo que solo había existido en la rama **test** se encuentra ahora también en **master**. Podemos eliminar una rama con el comando **git branch -d [nombre-rama]**.

GLOSARIO PROGRAMACIÓN

**escape character (carácter escape).** Un carácter de texto que no se toma literalmente pero tiene un significado especial cuando se combina el carácter o caracteres que lo consigue. El carácter \ es un carácter de escape.