

**ESCUELA POLITÉCNICA
SUPERIOR DE CÓRDOBA**
Universidad de Córdoba



SITIOS SERVERLESS CON JEKYLL Y DESPLIEGUE EN GITHUB PAGES



UNIVERSIDAD DE CÓRDOBA



ÍNDICE

1.	Introducción.....	3
2.	Generadores de sitios estáticos.....	4
3.	Instalación.....	4
4.	Directorios.....	7
5.	Liquid.....	9
6.	YAML.....	10
7.	Plantillas.....	12
8.	Posts.....	13
9.	Assets.....	14
10.	Collections.....	17
11.	Navegabilidad de nuestro sitio web.....	21
12.	Despliegue en GitHub Pages.....	22
13.	Bibliografía.....	24

1.INTRODUCCIÓN

Para empezar, tenemos que ponernos en situación y antes de profundizar, explicaremos que es jekyll, para qué sirve y los conceptos que trataremos en nuestro trabajo a grandes rasgos.

Jekyll es un generador de sitios web estáticos escrito en lenguaje ruby, esto significa que permite crear fácilmente un directorio organizado para contener nuestros htmls, sus estilos, y cualquier información o recurso que queramos compartir en nuestra página web. Ante cada cambio en sus archivos se debe recompilar todo el sitio para que dichos cambios se aplique a nuestro sitio web. Debido a que no servimos a internet directamente los archivos que modificamos, ya que nosotros los editamos haciendo uso de lenguajes como yaml y liquid, los cuales serán interpretados por jekyll para generar los htmls y css que si se proveerán al cliente.

Jekyll permite desarrollar nuestro sitio web de manera sencilla, rápida y segura. Haciendo uso de un sistema de plantillas que facilita la escritura de muchos htmls, la incorporación de lenguaje markdown, o las funcionalidades de los lenguajes de yaml y liquid para mostrar y hacer uso del contenido almacenado en nuestro directorio. Los sitios web estáticos compilan su contenido en el ordenador host local antes de ser servido, esto supone reducir las vulnerabilidades de nuestra base de datos, así como de la posibilidad de que se infiltre código malicioso.

Jekyll no permite gran interacción con el usuario, se usa generalmente para mostrar documentación de proyectos, blogs, y en definitiva sitios con pocos cambios sobre el contenido ya publicado, esto es lo que caracteriza los sitios web estáticos. Fundamentalmente el nuevo contenido que será añadido a nuestro sitio web serán nuevos posts o publicaciones.

Se introducirá al uso de los lenguajes Liquid, YAML y ruby como principales herramientas provistas por Jekyll, así como a la estructura que debe mantener nuestro sitio web para mejor navegabilidad y desempeño.

2. GENERADORES DE SITIOS ESTÁTICOS

Cuando realizas una búsqueda de generadores de este tipo, encontrarás fácilmente a Jekyll, hugo, hexo, pelican, entonces ¿Por qué Jekyll?

Todos estos compiladores de sitios web estáticos, son soportados por github-pages, sin embargo, solo Jekyll es propuesto por GitHub. Esto provoca que la mayoría de debutantes en la página opten por buscar información de jekyll en lugar de ninguno de los compiladores anteriormente mencionados, lo que incurre a su vez en una comunidad de jekyll que crece más que el resto.

Esta comunidad es el primer aspecto positivo al que nos vamos a referir, comentarios, plantillas, plugins y soluciones son brindadas de manera gratuita por la comunidad, lo que hace de jekyll un buen generador con el que comenzar.

Por otra parte, y aún más importante para aquellos debutantes no solo en los sitios estáticos sino en la programación web, jekyll no requiere un conocimiento elevado de programación web, en realidad, es muy sencillo, como veremos a continuación.

3. INSTALACIÓN

INSTALACIÓN EN UBUNTU

La última versión disponible de Jekyll es la 4.0 y un dato interesante que hemos observado es que esta versión requiere que el usuario desarrolle su sitio web estático desde cero ya que durante la instalación solo se creará una carpeta con los post y los archivos básicos para mostrar la página por defecto que Jekyll nos proporciona. Sin embargo, en versiones anteriores como la 3.1.6, genera más carpetas y archivos ya disponibles para el que el usuario las utilice.

Jekyll está escrito en ruby, un lenguaje de programación interpretado, reflexivo y orientado a objetos, por lo tanto, primero tendremos que instalar el entorno apropiado mediante el siguiente comando:

```
sudo apt-get install ruby-full build-essential zlib1g-dev
```

Con el comando anterior, instalamos librerías, dependencias y el entorno apropiado para el lenguaje ruby, según la distribución GNU/Linux que utilizemos será necesario utilizar un gestor de paquetes u otro, como por ejemplo: yum, pacman, brew, etc.

Si durante la instalación nos aparece alguno de los siguientes errores:

No se reconoce el paquete ruby-full

Unable to access ruby-full

Tendremos que instalar por separado las dependencias y el entorno de la siguiente manera:

```
sudo apt-get install ruby build-essential zlib1g-dev  
sudo apt-get install ruby-dev
```

A continuación, tenemos que añadir una serie de variables de entorno al path:

-Con el comando echo lo que hacemos es volcar el contenido que escribamos en el fichero indicado, en este caso ~/.bashrc.

-Y el comando source nos permite conservar el valor asignado a estas variables de entorno.

```
echo '# Install Ruby Gems to ~/gems' >> ~/.bashrc  
echo 'export GEM_HOME=\"$HOME/gems\"' >> ~/.bashrc  
echo 'export PATH=\"$HOME/gems/bin:$PATH\"' >> ~/.bashrc  
source ~/.bashrc
```

Ahora vamos a instalar bundler, que es un gestor de dependencias o “gemas” como se las conoce en el ámbito del lenguaje ruby.

Instalar individualmente cada una de las gemas de las que depende un proyecto puede ser muy tedioso, además de que sería necesario mantener una documentación actualizada con la lista de gemas y la versión de cada una por si cambiamos de máquina o entra un nuevo integrante a nuestro equipo.

Además, otro problema es que puedes tener varias versiones de la misma gema instalada. Por defecto Ruby utiliza la última versión de la gema, pero es posible que esa no sea la que nuestro proyecto necesite, por tanto Bundler nos servirá de gran ayuda.

Para instalarlo utilizamos el comando:

```
gem install bundler
```

Finalmente, después de instalar todas las dependencias necesarias podemos instalar Jekyll con el comando:

```
gem install jekyll bundler
```

INSTALACIÓN EN WINDOWS 10

Para la instalación en Windows tenemos dos opciones:

1. INSTALACIÓN MEDIANTE RUBYINSTALLER:

Descargamos una de las versiones más recientes y que se ajuste con las características de nuestro equipo, del entorno Ruby+Devkit desde el siguiente enlace:

<https://rubyinstaller.org/downloads/>

Es importante que la versión que instalemos sea superior a la 2.4 ya que Jekyll no proporciona soporte para versiones anteriores.

Durante el proceso de instalación seguimos las opciones por defecto, pero en el último paso del asistente de instalación tenemos que marcar la opción `ridk install` necesaria para que se instalen las gemas con extensiones nativas.

Por último, abrimos una terminal y tecleamos el siguiente comando para instalar Jekyll:

```
gem install jekyll bundler
```

Una vez terminada la instalación si queremos comprobar si Jekyll se ha instalado correctamente podemos usar el comando `jekyll -v`.

2. INSTALACIÓN MEDIANTE BASH:

Primero accedemos a una instancia de bash mediante el comando: `bash`
Después, nos aseguramos de que todos los paquetes y repositorios estén actualizados:

```
sudo apt-get update -y && sudo apt-get upgrade -y
```

A continuación, instalaremos Ruby y todo el entorno necesario a través de un repositorio que incluye versiones optimizadas y actualizadas de Ruby desde la plataforma BrightBox.

```
sudo apt-add-repository ppa:brightbox/ruby-ng  
sudo apt-get update  
sudo apt-get install ruby2.5 ruby2.5-dev build-essential dh-autoreconf
```

El siguiente paso será actualizar las gemas de Ruby:

```
gem update
```

Y por último, instalamos Jekyll con el siguiente comando:

```
gem install jekyll bundler
```

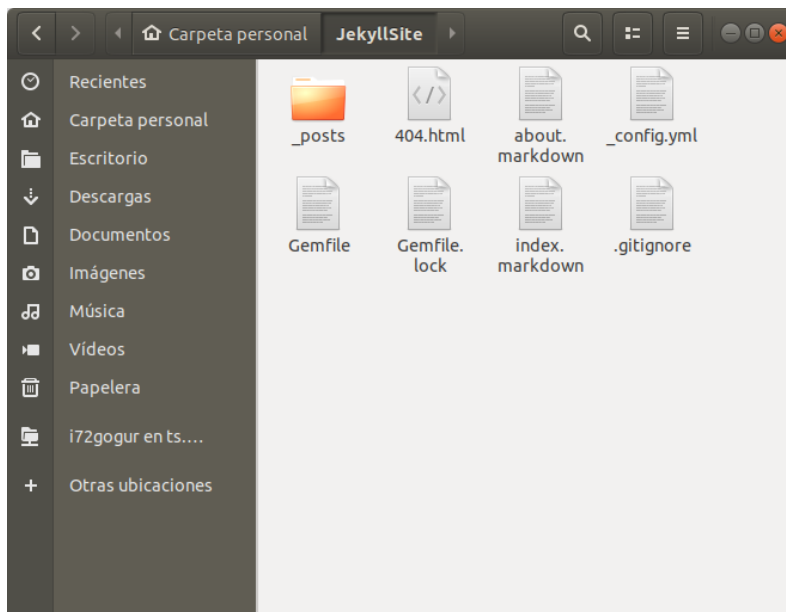
Como hemos indicado antes, para comprobar que el proceso de instalación ha terminado con éxito, utilizamos el comando `jekyll -v`

4. DIRECTORIOS

Comenzar con jekyll es tan sencillo como escribir un comando:

```
jekyll new JekyllSite
```

Este comando genera una nueva carpeta con el nombre JekyllSite, que por defecto con la versión 4.0 de Jekyll contendrá los siguientes archivos:



Aunque con vistas al desarrollo completo de nuestro sitio web recomendamos la siguiente estructura y archivos para el nuevo directorio:

- `_config.yml`: Este archivo creado por defecto, contendrá variables de configuración general de nuestro sitio, las cuales no pretendemos variar con frecuencia.
- `index.html` o `index.markdown`: Página principal de nuestro sitio web.
- `_layouts`: Carpeta para plantillas html.
- `_includes`: Carpeta para partes de código html que se pueden reutilizar.
- `_posts`: Carpeta con htmls de artículos, blogs, trabajos y contenido que queramos mostrar en nuestra web en general.

- `_sass` y `css`: Carpetas para las hojas de estilo. Más adelante explicaremos el por qué de tener dos carpetas, ya que nos ayudarán a hacer más eficiente el desarrollo.
- `js`: Carpeta que contendrá los scripts en lenguaje javascript que utilizemos en nuestro sitio web.
- `_data`: Archivos en formato Yaml, JSON, CSV y TSV a los que podremos acceder por medio de plantillas Liquid.
- `_site`: Esta carpeta no la creamos nosotros manualmente, ya que se genera tras el primer `jeekyll build` y contiene los `htmls` del sitio, formateados y listos para ser servidos. Es importante recordar que debemos hacer `jeekyll build` para que los cambios en nuestro sitio web se actualicen, además de subir dichos cambios a nuestro repositorio de github como explicaremos más adelante.
- `gemfile`: Archivo de configuración del bundler. Aquí editaremos las versiones que queremos emplear, y ejecutamos `bundle install` para que bundler se encargue de las dependencias, las cuales veremos reflejadas en `gemfile.lock`.
- `.gitignore`: Archivo que nos será de utilidad para trabajar con github ya que nos permite excluir cualquier archivo que no queramos añadir a nuestro repositorio.

En esta introducción, debemos destacar dos comandos fundamentales y con los que trabajaremos en el día a día:

bundle exec jeekyll serve, Jekyll sirve el sitio en local, modificando en tiempo real los cambios que hagamos a nuestros archivos. Podemos acceder a través del puerto 4000, poniendo en el navegador 127.0.0.1:4000 o localhost:4000.

Jekyll indica inicialmente la carpeta de la que lee, la carpeta en la cual escribe, el tiempo que ha tardado en generar el sitio. Para cada cambio advierte de errores de sintaxis, de configuraciones erróneas y el tiempo que ha tardado en regenerar parte de la página. Toda esta información la podemos ver en la terminal después de introducir este comando y mientras interactuamos con la página.

La opción `--incremental` permite recompilar solo los archivos en los que se realizan los cambios.

jeekyll build, Jekyll compila el sitio web entero en formato `html`, para que se vea como cualquier otro, sin liquid, yml, ni markdown, y lo vuelca en la carpeta `_site` de nuestro directorio, dispuesto para ser servido.

5. LIQUID

Liquid es un lenguaje de plantillas que nos permite crear “un puente” entre un archivo HTML y un registro de datos. En nuestro caso, el registro de datos será, por

ejemplo, el archivo `_config.yml` del cuál utilizaremos algunas variables generales, como la url del sitio, su nombre, etc.

A continuación, vamos a ver las funcionalidades que incorpora este lenguaje de marcado y su utilización para desarrollar nuestro sitio web con Jekyll:

1. Objetos: También llamadas etiquetas de texto, son palabras reservadas que hacen referencia a una entidad y sus propiedades, por entidades nos referimos al archivo `config.yml` con la palabra `site`, por ejemplo. Se usan mediante dos braces para abrir y cerrar `{{ variable.propiedad }}`. Entre las variables por defecto están: “`site`” que tendrá tantas propiedades como queramos añadirle mediante lenguaje `yaml`, “`posts`” que hace referencia a la carpeta donde se alojan nuestros blogs se usa `site.posts`, “`pages`” hace referencia a los archivos `md` en el directorio raíz se usa `site.pages`, “`data`” hace referencia a la carpeta `_data`, podremos acceder a ella y sus subcarpetas mediante `site.data.NombreCarpeta`.
2. Tags: Las etiquetas nos permiten crear cierta lógica en los `htmls`, como recorrer los elementos de una carpeta de nuestro directorio, por ejemplo. La sintaxis se compone de braces y porcentajes para abrir y cerrar, `{% for post in site.posts %} {% endfor %}`
Podemos emplear `for`, `unless`, `case/when`, `ifs`, `if-else`, `tablerow`...
<https://shopify.github.io/liquid/tags/comment/> En este enlace encontraréis más etiquetas.
3. Filtros: Liquid también permite cambiar reformatear una cadena de entrada en una salida distinta, existen diferentes criterios de formateo, que son los distintos filtros, hay gran variedad, cabe destacar el uso del filtro `markdownify` para pasar `md` a `html`.
<https://jekyllrb.com/docs/liquid/filters/> En este enlace encontraréis los filtros más utilizados para jekyll y al final un enlace a otra web con todos ellos.

A modo de tutorial hemos realizado un siguiente video explicativo para documentar nuestra explicación de una forma más visual que podeis encontrar en nuestra página web: <https://i72gogur.github.io/Jekyll-GithubPages/>

6. YAML

Jekyll hace uso de “otro lenguaje de marcado más”, que permite guardar datos de forma serializada, legible para el ser humano. De esta forma podremos acceder a ellos y recorrerlos fácilmente con liquid. Sus características son las siguientes:

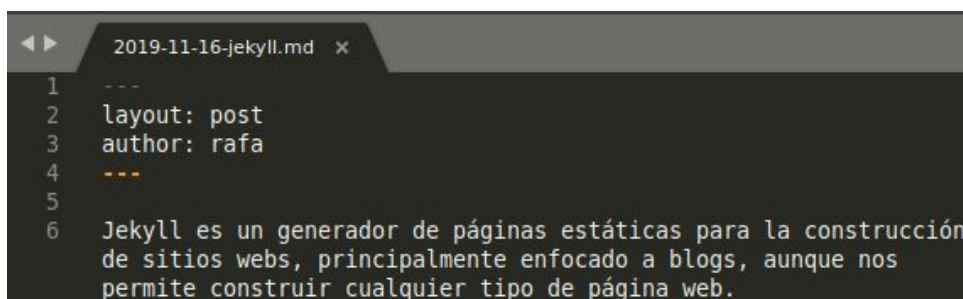
1. Front matter: YAML utiliza el asunto principal al comienzo de los archivos markdown para indicar información adicional acerca de los mismos, en Jekyll no solo lo utilizaremos con este fin. Además de utilizarlo también en documentos de tipo html, nos servirá para indicar ciertos parámetros a Jekyll a la hora de la compilación del documento, por ejemplo:

Si añadimos la línea **layout : default** en el front matter de nuestro archivo principal index.html estamos indicando a Jekyll que el archivo html no está completo, y que debe hacerse uso del archivo llamado default.html que se encuentra en la carpeta _layouts (plantillas).

Podemos añadir distintos valores al front matter, de tal forma que podrán ser usados por la plantilla que empleen en el contenido del archivo.

Además todos aquellos archivos con front matter, serán recompilados durante el servicio local (bundle jekyll serve) si se modifican.

La sintaxis son 3 guiones, pares *atributo: valor* uno por cada línea, y 3 guiones en la siguiente línea. Además, tenemos que dejar una línea en blanco entre los últimos 3 guiones y el contenido del archivo.



```
2019-11-16-jekyll.md x
1 ---
2 layout: post
3 author: rafa
4 ---
5
6 Jekyll es un generador de páginas estáticas para la construcción
  de sitios webs, principalmente enfocado a blogs, aunque nos
  permite construir cualquier tipo de página web.
```

2. Objetos: Se declaran con un par *atributo: valor*, los cuales pueden tener objetos anidados (para indicar que un objeto pertenece a otro, se añade en la línea debajo de él u de otro objeto anidado, y se le añaden dos espacios delante). Es importante utilizar el espaciado simple y no el tabulado ya que yaml solo reconoce el espaciado simple y nos ayudará a prevenir errores.
3. Array: Se declaran con un un guión, espacio y valor, en cada línea un valor.
4. Array de objetos u objetos con arrays: Se pueden combinar ambos elementos, para elaborar la estructura de almacenamiento de datos que más nos convenga.

La sintaxis es muy importante en todo lenguaje, en YAML se basa en guiones, espacios y dos puntos. Hay que prestar especial atención pues es propenso a errores cuando te estás iniciando en el uso de este lenguaje.

<https://learn-the-web.algonquindesign.ca/topics/markdown-yaml-cheat-sheet/#yaml>

Se añade un vídeo tutorial en el que se explica el uso generalizado de yaml en jekyll, también en nuestra web: <https://i72gogur.github.io/Jekyll-GithubPages/>

7. PLANTILLAS

Una gran utilidad de jekyll es la reutilización de código por medio de plantillas. Esta herramienta nos hará mucho más fácil y rápido el desarrollo de nuestro sitio web ya que podemos escribir un html para cada sección que se nos plantee: blogs, investigaciones, posts. Por ejemplo, aunque tengamos un número considerable de posts, tan solo tendremos que tener una plantilla para ellos, la cuál añadiremos al directorio `_layouts`.

Las plantillas funcionan gracias a la etiqueta `{{ content }}`, redactamos la plantilla completa, dejando un hueco para el contenido variable de cada post, publicación, etc. en el que escribimos la etiqueta `content`, de tal forma que cada html que utilice la plantilla, reemplazará el hueco en blanco por su contenido.

Este recurso nos facilita el cumplimiento del principio de diseño de consistencia ya que nos permitirá que nuestra web sea homogénea y mejorará la navegabilidad del usuario.

Por tanto, utilizando esta herramienta nuestros htmls no necesitarán etiqueta `doctype`, `html`, `head` ni `body`, ya que estos estarán incluidos en las plantillas, solo necesitarán un front matter indicando la plantilla a usar con la sintaxis *layout: plantilla*.

Facilitamos un video tutorial en el que explicamos detalladamente cómo crear plantillas, y como reutilizar el código con la etiqueta `{%include%}` y la carpeta `_include`. <https://i72gogur.github.io/Jekyll-GithubPages/>

Otro aporte curioso que hemos descubierto es que se pueden descargar plantillas gratuitamente en internet, las cuales nos facilitan un directorio parecido al que nos genera jekyll pero con los estilos y htmls del autor. El inconveniente que conlleva es que a menudo estas plantillas pueden presentar problemas de dependencias de versiones de gemas de ruby, pero como hemos instalado previamente el gestor de gemas bundler, no es un problema.

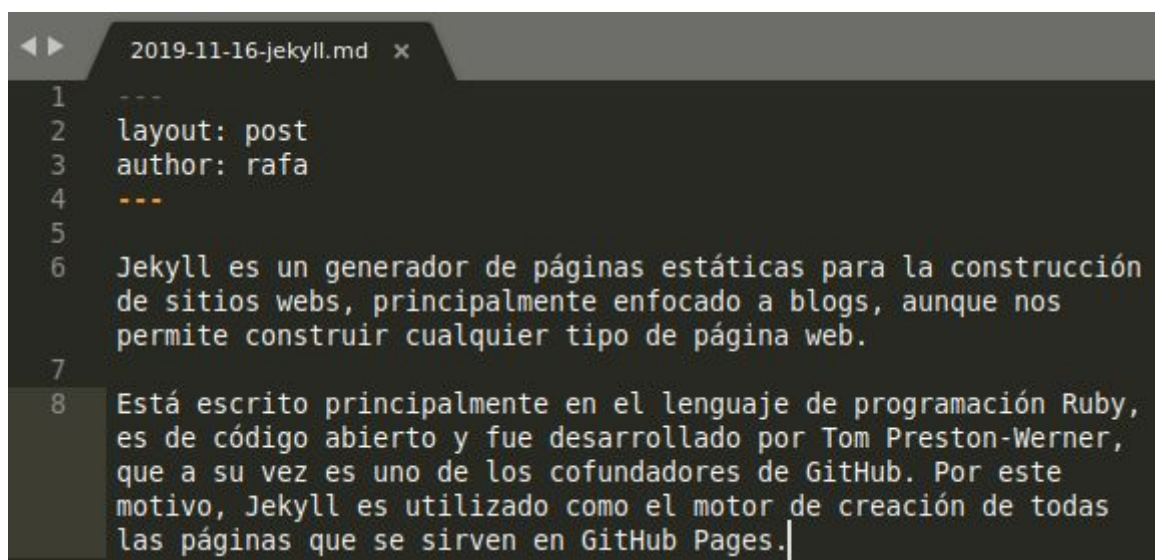
Para hacer uso de plantillas de internet, basta con ejecutar *bundle install* desde la terminal, dentro de la carpeta que hayamos descargado y modificar los parámetros del archivo `_config.yml` a nuestra necesidad.

8. POSTS

A partir de la creación de nuestras plantillas, publicar nuevos artículos es muy rápido, basta con escribir la parte de código que hemos dejado en blanco (etiqueta content ya explicada anteriormente) en la plantilla, en un archivo a parte, en formato html o markdown.

Los requisitos para que nuestro post se muestre correctamente son guardar el fichero bajo un nombre con el siguiente formato YYYY-MM-DD-Nombre-del-post en la carpeta _posts, donde YYYY-MM-DD hace referencia a la fecha de publicación del post, que nos servirá también para llevar un control de cuando son publicados.

Por ejemplo, este sería uno de los primeros post de nuestro sitio web:



```
1 ---
2 layout: post
3 author: rafa
4 ---
5
6 Jekyll es un generador de páginas estáticas para la construcción
7 de sitios webs, principalmente enfocado a blogs, aunque nos
8 permite construir cualquier tipo de página web.
9
10 Está escrito principalmente en el lenguaje de programación Ruby,
11 es de código abierto y fue desarrollado por Tom Preston-Werner,
12 que a su vez es uno de los cofundadores de GitHub. Por este
13 motivo, Jekyll es utilizado como el motor de creación de todas
14 las páginas que se sirven en GitHub Pages.
```

Además de la plantilla utilizada, que en este caso será la de post, hemos añadido también en el front matter otro par atributo: valor, que en este caso hace referencia al autor del post. Sin embargo, author es una variable personalizada por lo que cada desarrollador le podrá asignar el nombre que quiera, como por ejemplo “creator” o “escritor”.

Cada desarrollador podrá añadir al front matter todos los pares atributo valor que crea necesarios.

9. ASSETS

Otro recurso que nos servirá de utilidad para la elaboración de nuestro sitio web son los assets, los cuales nos permitirán añadir fotos o archivos PDF a nuestros posts. Para ello crearemos una carpeta que con el nombre de assets y en la que incluiremos todas las imágenes o archivos que queremos que aparezcan en nuestra página. Además, también podemos añadir archivos de estilo css y nuestros scripts en javascript de esta forma, organizando la carpeta en subcarpetas tendremos todos los recursos bien organizados.

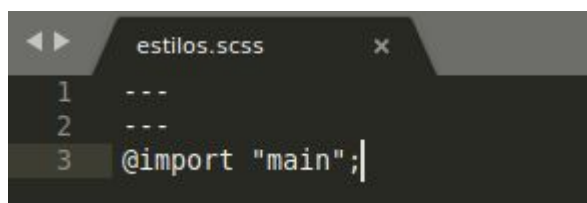
Después, para utilizarlos simplemente tendremos que enlazarlos añadiendo la ruta del archivo que queremos añadir en el código de nuestro post.

Por ejemplo, si queremos añadir la imagen de nuestro logotipo en lenguaje Markdown haríamos lo siguiente:

```
![Logotipo](/assets/images/logotipo.png)
```

La principal utilidad que tiene el uso de assets es a la hora de dar estilo a nuestra página, para ello podemos utilizar Sass que es un preprocesador de lenguaje para hojas de estilo y que podemos usar conjuntamente con CSS.

Primero crearemos un archivo Saas con extensión .scss, por ejemplo estilos.scss y lo añadiremos a nuestra carpeta de css dentro del directorio de assets. En este archivo dejaremos el front matter vacío, **lo cual le indica a jekyll que tiene que procesar** y añadiremos también la siguiente línea de código para importar el archivo de estilos, quedando nuestro archivos scss de la siguiente forma:



```
1 ---
2 ---
3 @import "main";|
```

Con esto indicamos a Sass que busque un archivo llamado main.scss (que en nuestro caso contendrá el estilo de nuestra página principal) en el directorio correspondiente a los archivos Sass (/_sass).

Por último, para usar esta hoja de estilos solamente tendremos que incluir en la cabecera del archivo html al que queramos dar forma la referencia a nuestro archivo scss creado al principio:

```
<head>
  <meta charset="utf-8">
  <title>{{ page.title }}</title>
  <link rel="stylesheet" href="{{ "/assets/css/styles.css" | prepend: site.baseurl }}">
</head>
```

Para proyectos grandes en los que tenemos varias hojas de estilo esta es la mejor forma de tener organizado nuestro código.

HABLANDO DE SASS:

Sass es un preprocesador CSS que nos permite traducir un código de hojas de estilo no estándar, específico del preprocesador en cuestión; en este caso con la extensión .scss, a un código CSS estándar y como ya hemos comentado antes nos permite trabajar mucho más rápido en la creación de código.

Sass dispone de dos formatos diferentes para su sintaxis: archivos .sass y archivos .scss, aunque el más utilizado actualmente es .scss y es el formato utilizado en el desarrollo de nuestro trabajo.

PECULIARIDADES DE SU SINTAXIS:

Sass permite el uso de variables para definir cualquier valor. Para ello el nombre de la variable irá precedido del símbolo \$. El siguiente ejemplo ilustra su uso:

```
3 $color_botones: #F00;
4 .boton{
5     background-color: $color_botones;
6 }
```

El cuál sería equivalente al siguiente archivo en css:

```
3 .boton{
4     background-color: #F00;
5 }
```

Otra opción interesante que nos permite Sass es la anidación, ya que es muy común en el desarrollo con html tener una estructura donde unos elementos estén dentro de otros, como por ejemplo un menú donde para dar estilo al último elemento anidado tendríamos que poner todos los elementos padre en orden:

```
3 nav ul li a{
4     // ...
5 }
```

Y si quisiéramos dar estilo a alguno de los elementos padre haríamos lo siguiente:

```

3   nav{
4     // ...
5   }
6
7   nav ul{
8     // ...
9   }
10  |
11  nav ul li{
12    // ...
13  }

```

Esto acaba generando páginas CSS complicadas de mantener, y en este punto es donde SASS nos ofrece una forma más clara de trabajar ya que nos permite al igual que en html anidar nuestros estilos de una forma más visual. Por ejemplo, para dar estilos a una barra de navegación:

```

3   nav{
4     ul{
5       margin: 0;
6       padding: 0;
7       li{
8         a{
9           color: blue;
10        }
11      }
12    }
13  }

```

Y su equivalente en lenguaje CSS sería:

```

3   nav ul {
4     margin: 0;
5     padding: 0;
6   }
7
8   nav ul li a{
9     color: blue;
10  }

```

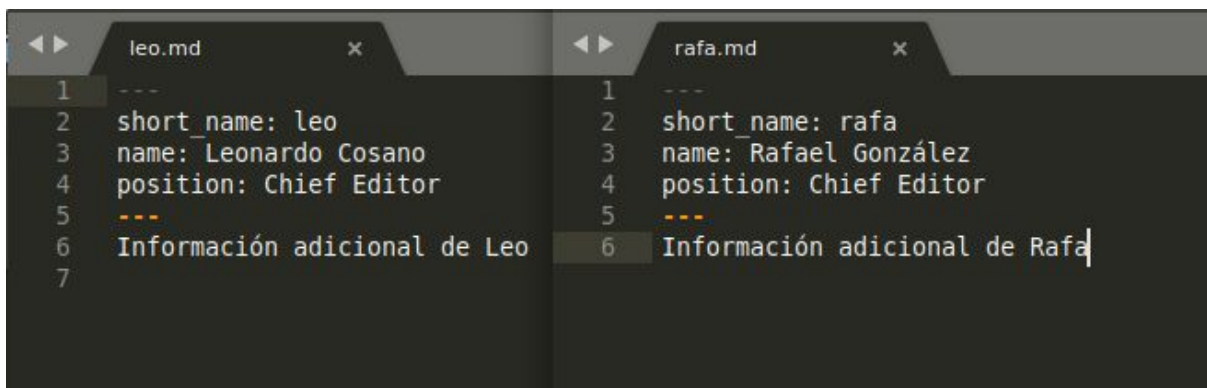
10. COLLECTIONS

Es común que la administración de sitios web, sobre todo los dedicados a la publicación de posts y noticias sea llevada a cabo por más de una persona. Por tanto, si tenemos algún colaborador usaremos esta opción para identificar a qué autor pertenece cada post.

Para usar esta opción tendremos que editar el archivo de configuración `_config.yml` y añadiendo lo siguiente advertimos a jekyll de que tenemos una colección de autores:

```
36
37 #Collections
38 collections:
39   authors:
40
```

Para añadir nuevos autores crearemos una carpeta en nuestro directorio raíz con el nombre de `_authors` y en la que añadiremos un archivo en formato markdown (o html=) para cada uno con su información personal. Por ejemplo en nuestro caso añadiremos dos archivos:



```
leo.md
1 ---
2 short_name: leo
3 name: Leonardo Cosano
4 position: Chief Editor
5 ---
6 Información adicional de Leo
7

rafa.md
1 ---
2 short_name: rafa
3 name: Rafael González
4 position: Chief Editor
5 ---
6 Información adicional de Rafa
```

Como ya hemos explicado anteriormente, los pares atributo: valor añadidos al front matter han sido a elección personal y cada desarrollador puede añadir los suyos propios.

También podemos añadir a nuestro sitio web una lista con los colaboradores fácilmente utilizando la herramienta collections ya que los autores estarán disponibles en la variable **site.authors** por lo que simplemente iterando esta variable podemos mostrar la información que queramos de nuestros autores. Por ejemplo, a continuación mostraremos una lista con el nombre de nuestros colaboradores, su función e información adicional:

```

colaboradores.html x
1 ---
2 layout: default
3 title: Colaboradores
4 ---
5 <h1>COLABORADORES</h1>
6
7 <ul>
8   {% for author in site.authors %}
9     <li>
10      <h2>{{ author.name }}</h2>
11      <h3>{{ author.position }}</h3>
12      <p>{{ author.content | markdownify }}</p>
13    </li>
14  {% endfor %}
15 </ul>

```

Si queremos mostrar la información adicional que no se encuentra dentro del front matter tendremos que utilizar el filtro `markdownify` para que se muestre en nuestro sitio web correctamente.

Por otro lado, si queremos que cada autor o colaborador tenga su propia sección tendremos que indicarlo en el archivo de configuración `_config.yml` añadiendo lo siguiente:

```

37 #Collections
38 collections:
39   authors:
40     output: true
41

```

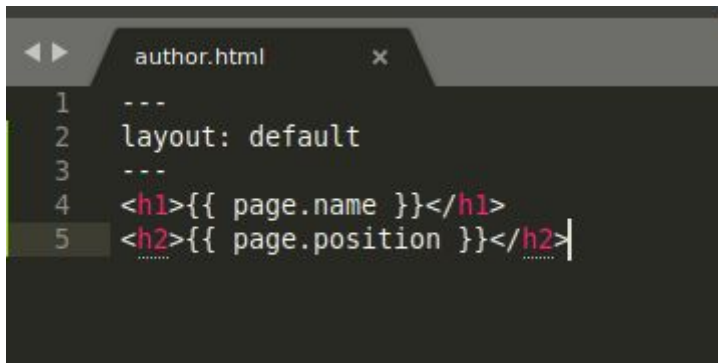
De esta forma si queremos que al mostrar la información de los colaboradores el usuario pueda ir a la sección concreta de un autor solo tendremos que utilizar la variable `author.url`:

```

colaboradores.html x
1 ---
2 layout: default
3 title: Colaboradores
4 ---
5 <h1>COLABORADORES</h1>
6
7 <ul>
8   {% for author in site.authors %}
9     <li>
10      <h2><a href="{{ author.url }}">{{ author.name }}</h2>
11      <h3>{{ author.position }}</h3>
12      <p>{{ author.content | markdownify }}</p>
13    </li>
14  {% endfor %}
15 </ul>

```

Como ya hicimos con los posts tendremos que añadir la plantilla correspondiente para los autores en la carpeta `_layouts` con el siguiente contenido:

A screenshot of a code editor window titled 'author.html'. The editor shows five lines of code. Line 1 is '---'. Line 2 is 'layout: default'. Line 3 is '---'. Line 4 is '<h1>{{ page.name }}</h1>'. Line 5 is '<h2>{{ page.position }}</h2>'.

```
1 ---
2 layout: default
3 ---
4 <h1>{{ page.name }}</h1>
5 <h2>{{ page.position }}</h2>
```

Ahora lo que tenemos que hacer es configurar los archivos de cada autor para que usen el archivo añadido a la carpeta de `_layouts` y esto lo podemos hacer fácilmente editando el archivo de configuración `_config.yml`. Esto nos servirá siempre que queramos añadir valores predeterminados para cualquier diseño en nuestra página y nos permitirá eliminar el atributo `layout` que se encuentra en el front matter de cada post, autor o sección. Así estamos evitando repetir líneas de código, para hacer nuestro sitio web más eficiente.

A screenshot of a code editor window titled '_config.yml'. The editor shows lines 41 to 58. Line 41 is empty. Line 42 is '#Valores predeterminados'. Line 43 is 'defaults:'. Line 44 is '- scope:'. Line 45 is 'path: ""'. Line 46 is 'type: "authors"'. Line 47 is 'values:'. Line 48 is 'layout: "author"'. Line 49 is '- scope:'. Line 50 is 'path: ""'. Line 51 is 'type: "posts"'. Line 52 is 'values:'. Line 53 is 'layout: "post"'. Line 54 is '- scope:'. Line 55 is 'path: ""'. Line 56 is 'values:'. Line 57 is 'layout: "default"'. Line 58 is empty.

```
41
42 #Valores predeterminados
43 defaults:
44   - scope:
45       path: ""
46       type: "authors"
47     values:
48       layout: "author"
49   - scope:
50       path: ""
51       type: "posts"
52     values:
53       layout: "post"
54   - scope:
55       path: ""
56     values:
57       layout: "default"
58
```

Otra característica interesante que podemos añadir a nuestro sitio web consiste en filtrar las publicaciones por autor para mostrar, por ejemplo, solo las publicaciones de un autor concreto. Para ello deberemos hacer coincidir el atributo `short_name` (llamado así en nuestro caso) de cada autor con el atributo `author` de cada post. Guardaremos en una lista todos los post en los que ambos atributos coincidan y después solo tendremos que recorrer esa lista mediante un bucle `for` para mostrarlos:

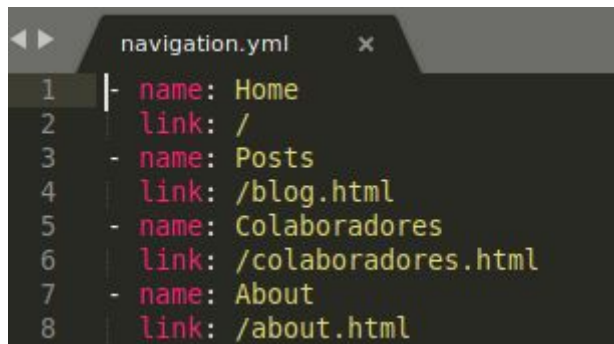
```
author.html x
7  {{ content }}
8
9  <h1>POSTS</h1>
10 <ul>
11   <!--Guardamos en la lista post_filtrados los post cuyo autor coincida con el nombre de autor
12   buscado-->
13   {% assign post_filtrados = site.posts | where: 'author', page.shortname %}
14
15   <!--Recorremos la lista mostrando el titulo de todos los post filtrados-->
16   {%for post in filtered_posts %}
17     <li><a href="{{ post.url }}">{{ post.title }}</a></li>
18   {% endfor %}
19 </ul>
```

Por último también podemos vincular cada post con la página de su autor ya que cada post tiene una referencia al autor. Para ello en el archivo `post.html` del directorio `_layouts` añadimos el siguiente código, utilizando como enlace la variable `author.url`:

```
post.html x
1  ---
2  layout: default
3  ---
4  <h1>{{ page.title }}</h1>
5
6  <p>
7    {{ page.date | date_to_string }}
8
9    <!--Guardamos en la variable author el autor del post-->
10    {% assign author = site.authors | where: 'short_name', page.author | first %}
11    {% if author %}
12      <!--Añadimos la referencia a la página de dicho autor-->
13      - <a href="{{ author.url }}">{{ author.name }}</a>
14    {% endif %}
15  </p>
16
17  {{ content }}
```

11. NAVEGABILIDAD DE NUESTRO SITIO WEB


Si nuestro sitio web está formado por distintas secciones podemos añadir un apartado para cada sección el cuál será accesible desde la barra de navegación. Para ello, creamos un nuevo archivo en formato YAML que llamaremos `navigation.yml` y escribiremos una nueva entrada por cada sección, con el nombre y el enlace de cada una.



```
1 | name: Home
2 | link: /
3 | - name: Posts
4 |   link: /blog.html
5 | - name: Colaboradores
6 |   link: /colaboradores.html
7 | - name: About
8 |   link: /about.html
```

Podremos acceder a cada una de las secciones, iterando en un bucle `for` `{% for Seccion in site.data.navigation %}` y a cada iteración, el objeto sección tomará un valor distinto, pudiendo acceder a su `name`, `link`, y cualquier otro atributo que le añadamos.

Por ejemplo, para el desarrollo de nuestro sitio web hemos realizado una barra de navegación, mediante la iteración por secciones explicada:



```
<nav class="navegacion">
  <h1 id="titulo">SITIOS SERVERLESS CON JEKYL</h1>
  {% for item in site.data.navigation %}
    <a href="{{ item.link }}" {% if page.url == item.link %}class="current"{% endif %}>
      {{ item.name }}
    </a>
  {% endfor %}
</nav>
```

En este código, además comprobamos mediante una sentencia `if`, si el `link` del elemento iterado se corresponde con el `link` de la página actual y si es así aplicamos los estilos correspondientes de la clase `"current"` que podemos encontrar en el archivo `main.css` y mostrará al usuario de un color diferente la pestaña de la sección en la que se encuentre.

Al igual que hemos hecho este archivo YAML para crear una barra de navegación, podemos crear cualquier otro listado de items sobre el que queramos iterar fácilmente con frecuencia siguiendo la misma estructura y guardado el archivo **listado.yml** en la carpeta `_data`, de esta forma accederemos a dicho listado mediante el objeto **site.data.listado**

12. DESPLIEGUE EN GITHUB PAGES

Github Pages nos permitirá alojar nuestro sitio web de forma gratuita haciendo uso de un repositorio en el que se cargará todo el contenido.

Para empezar con el despliegue, crearemos un nuevo repositorio. Como recomendación personal aconsejamos que ese nombre sea significativo con respecto al sitio web ya que formará parte de la url para acceder al sitio.

En nuestro caso hemos creado un repositorio con el nombre **Jekyll-GithubPages**, este nombre es el mismo del proyecto generado al principio del desarrollo de este trabajo con el comando ***jekyll new***.

Tenemos que modificar el archivo de configuración de nuestro repositorio local `_config.yml` donde añadiremos la dirección url y baseurl. En nuestro caso quedaría de la siguiente forma:

```
27 baseurl: "/Jekyll-GithubPages" # the subpath of your site, e.g. /blog
28 url: "https://i72gogur.github.io/Jekyll-GithubPages" # the base hostname
```

A continuación, desde la terminal accedemos a nuestro directorio e iniciamos git con el comando ***git init***.

El siguiente paso será enlazar nuestro repositorio de Github creado con el directorio local mediante el siguiente comando:

```
git remote add origin https://github.com/i72gogur/NombreDelRepositorio.git
```

Después creamos una nueva rama en el repositorio donde añadiremos el contenido de nuestro sitio web, aunque también podríamos hacer esto en la rama master, recomendamos la creación de una nueva rama para este fin. Para crear la nueva rama utilizamos el comando, donde gh-pages es el nombre que le hemos asignado a la nueva rama:

```
git checkout -b gh-pages
```

Una vez creada la rama estaremos dentro de ella automáticamente y tendremos que añadir el contenido de nuestro directorio local con el siguiente comando:

```
git add .
```

Con el `.` indicamos a git que añada todos los archivos, pero si durante el desarrollo de nuestro proyecto editamos un archivo y queremos subir solo ese archivo modificado lo podemos hacer con el comando:

```
git add NombreDelArchivo
```

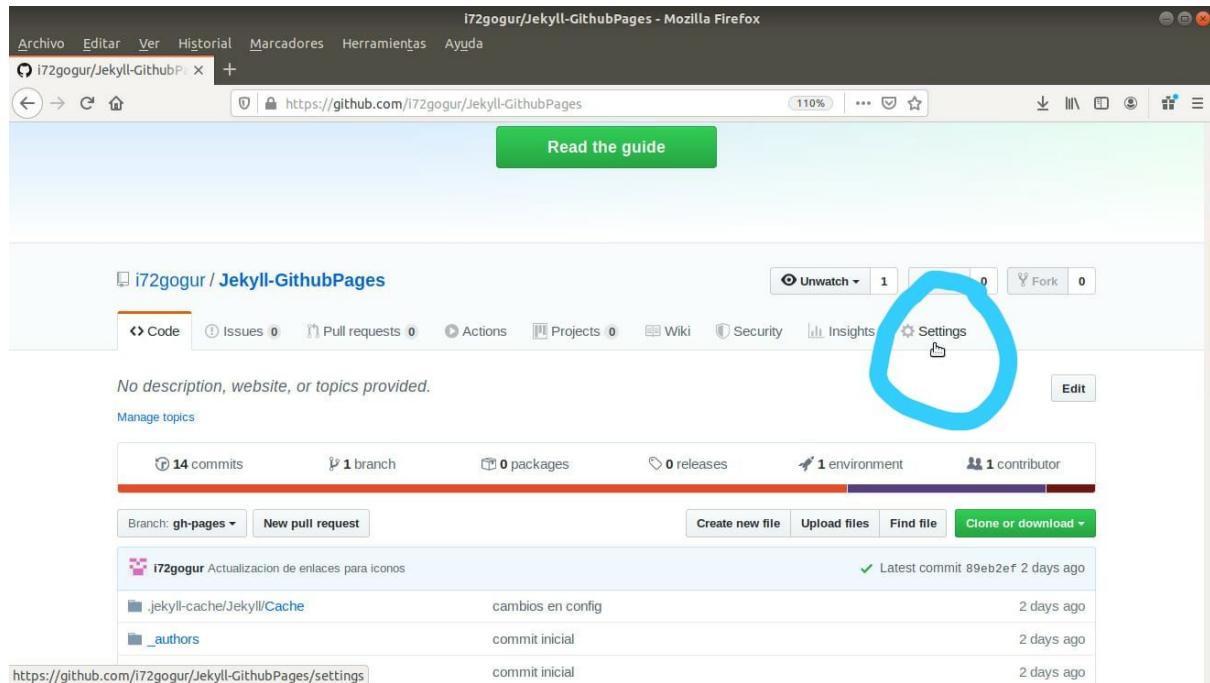
Una vez añadido el contenido haremos un commit para llevar el control de los cambios en el repositorio con el comando:

```
git commit -m "Commit inicial"
```

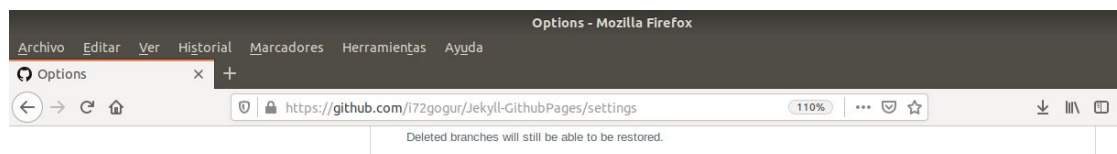

Y por último para que los cambios se hagan efectivos en el repositorio de github tendremos que hacer push en la rama creada:

git push origin gh-pages

También tendremos que configurar nuestro repositorio de github para que muestre la rama de gh-pages creada con anterioridad. Para ello nos vamos a github y accedemos al apartado de configuración dentro de nuestro repositorio:

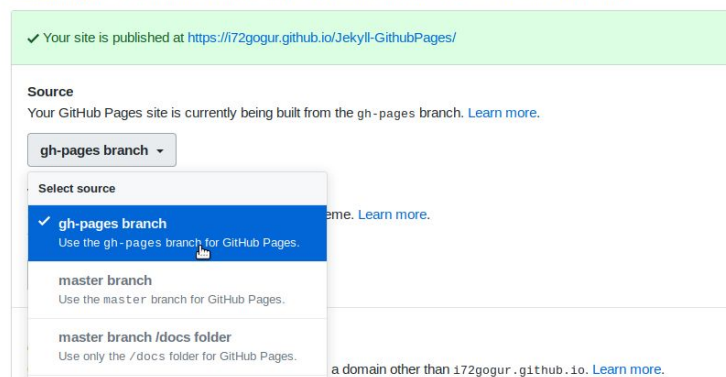


Dentro de ajustes, nos vamos al apartado de Github Pages y seleccionamos la rama creada desde la que se construirá nuestro sitio web.



GitHub Pages

GitHub Pages is designed to host your personal, organization, or project pages from a GitHub repository.



BIBLIOGRAFÍA

ENLACES PARA APRENDER JEKYL

-Documentación oficial de jekyll:

<https://jekyllrb.com/docs/usage/>

-Tutorial de instalación más detallado

<https://jekyllrb.com/docs/>

-Tutorial español sobre configuración básica y estructura de carpetas

<https://jorgeatgu.com/blog/una-guia-sobre-jekyll-configuracion/>

-Chuletario de jekyll.

<https://learn.cloudcannon.com/jekyll-cheat-sheet/>

-Chuletario YAML

<https://learn-the-web.algonquindesign.ca/topics/markdown-yaml-cheat-sheet/#yaml>

-Filtros y etiquetas liquid:

<https://shopify.github.io/liquid/>

-SITIOS SERVERLESS:

<https://www.campusmvp.es/recursos/post/que-son-las-arquitecturas-sin-servidor-serverless-computing-en-la-nube-y-por-que-deberian-interesarte.aspx>

<https://aws.amazon.com/es/lambda/serverless-architectures-learn-more/>

<https://www.bbva.com/es/serverless/>

-OTROS ENLACES:

<https://help.github.com/es/github/working-with-github-pages/about-github-pages-and-jekyll>