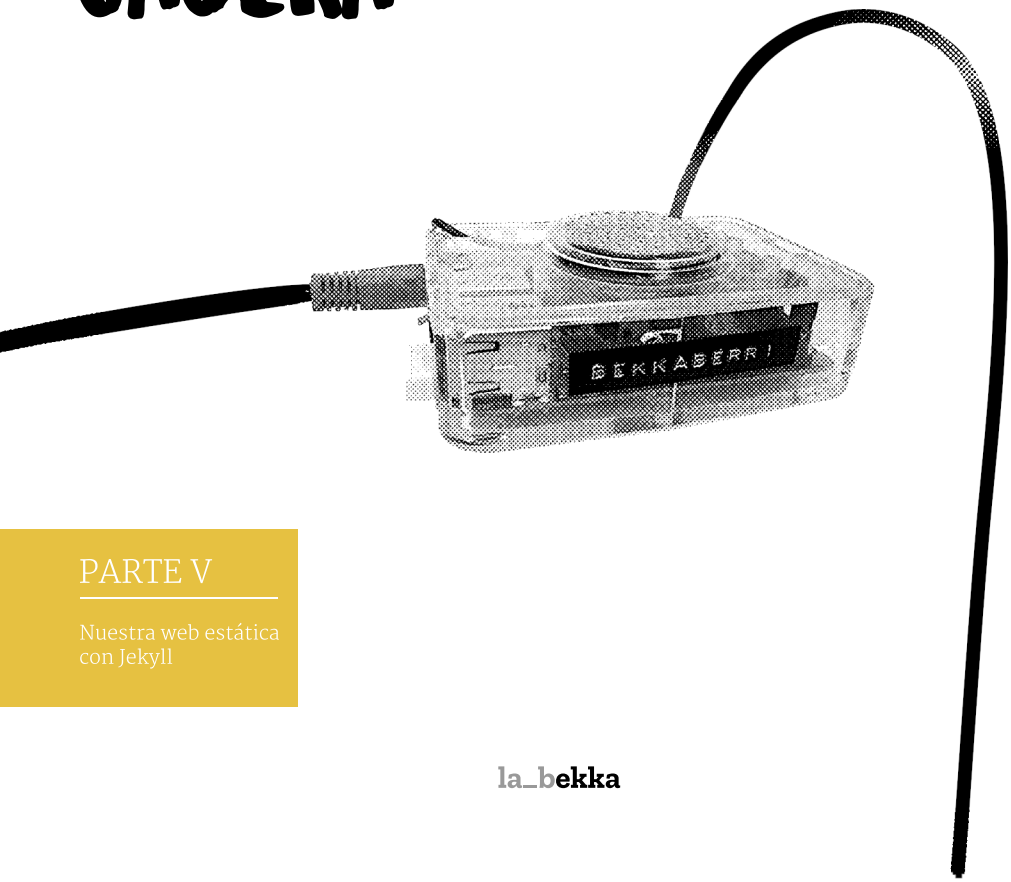
CÓMO MONTAR UNA SERVIDORA FEMINISTA CON UNA CONEXIÓN CASERA



PARTE V

Nuestra web estática con Jekyll

la_bekka

la_bekka <espacio hackfeminista>
https://labekka.red
hacklabfeminista@riseup.net
GPG 0xD47C2A1A064B0D97

Proceso de aprendizaje:
Irene, m4rtu, Inés Binder, Andrea, Marta S.

Redacción y contenidos:
Inés Binder y m4rtu.

Diseño y diagramación:
Inés Binder y m4rtu.

Impreso en Canarias.

PARTE V: NUESTRA WEB ESTÁTICA CON JEKYLL

Contenidos

17. Creemos un nievo sitio web estático con Jekyll

17.2 Instalemos Jekyll

Como vimos en el apartado 9, una página web puede ser tan sólo un archivo HTML. A la colección de páginas web HTML relacionadas entre sí y vinculadas a un dominio se las suele llamar "sitio web". El formato -tamaño y fuentes del texto, colores, etc.- de las páginas

web se suele definir a través de otros archivos con la extensión CSS. Así que básicamente lo que hace un navegador web es leer archivos HTML y CSS consultado el servidor web que los está "sirviendo".

Sin embargo, pocas personas crean sitios webs escribiendo estos archivos HTML y CSS desde cero. Normalmente utilizan otros programas que las ayudan a hacerlo más rápidamente aún cuando no tengan conocimientos de lenguaje HTML y CSS. Existen unos programas llamados "sistemas de gestión de contenidos" (CMS, por sus siglas en inglés) y que no son más que programas que nos ayudan a administrar nuestras publicaciones web. Quizás te suenen Wordpress, Joomla o Drupal. Estos sistemas requieren del uso de bases de datos, PHP y otros recursos más complejos para generar sitios webs "dinámicos" que se actualizan permanentemente. Aunque es la manera más difundida de montar sitios web, no es la única.

Otra opción es utilizar otro tipo de programas llamados "generadores de sitios estáticos", como Jekyll o Hugo, que no requieren de bases de datos ni otras tecnologías más complejas. Estos programas, en vez de instalarse en la propia servidora, se instalan en la computadora/ordenador de la persona que lo actualiza, generando archivos HTML y CSS. Una vez que generamos los archivos, los subimos a la servidora y tenemos nuestros sitios web "estáticos", que no necesitan de bases de datos para funcionar.

La característica fundamental de los generadores de sitios estáticos es que "fusionan" el contenido del sitio con la plantilla

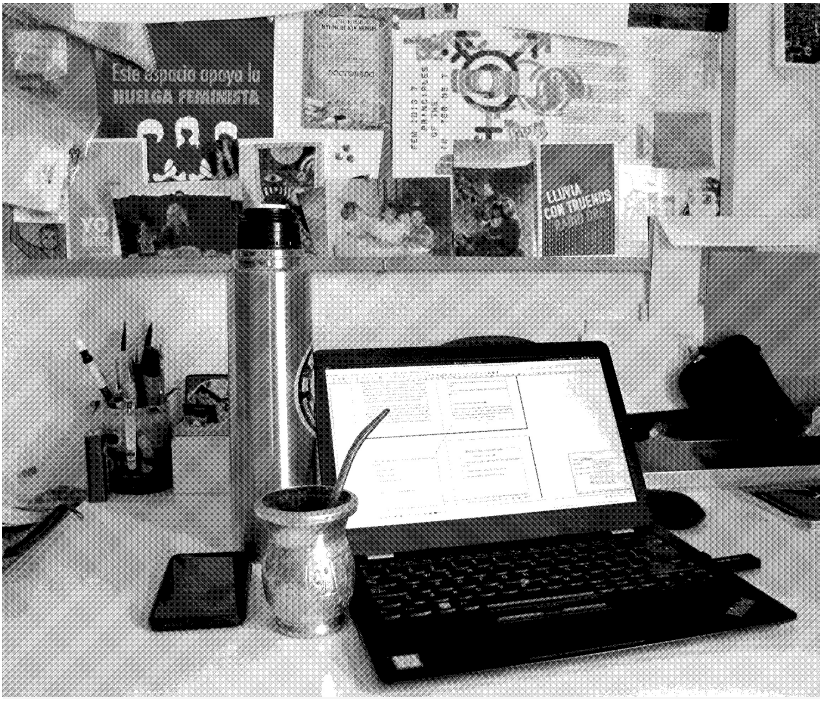
(*template*, eso que le da estructura y formato al contenido) cada vez que actualizamos el sitio en nuestra computadora y no cada vez que se visita, como sí ocurre en los sitios webs dinámicos. Es decir, esta "fusión" de todos lo elementos del sitio web se hace de manera local. Esto no significa que tengamos que tener sitios web aburridos o extremadamente básicos. ¡Nada más lejos! Podemos subir fotos, vídeos, y diseñarla como queramos. Y aunque "tocar" los templates sea más avanzado podemos elegir entre cientos de plantillas gratuitas y libres predeterminadas.

Los sitios web estáticos corren con algunas ventajas: son mucho más ligeros y por lo tanto más rápidos, más seguros y, al no tener comunicación con ninguna base de datos, reducen el consumo de energía eléctrica y la posibilidad de ser atacadas a través de ellas. Otra de sus ventajas es que son muy portables (se pueden distribuir de maneras alternativas, por USB, por ejemplo, o a través de redes p2p), son fáciles de respaldar y restaurar en caso de ataque.

Es verdad, también, que la gestión de este tipo de sitios web puede llegar a ser compleja. Primero, porque hay que tener algunos conocimientos previos. Al no tener una interfaz de administración la usuaria que quiera actualizar la web puede sentirse desorientada. Además, si un equipo es quien se encarga de administrarla tenemos que aprender a usar Git para compartir repositorios y controlar las versiones. Por eso, si no estamos cómodas, podemos usar plantillas HTML sencillas. Pero si nos animamos a aprender, aunque exista la posibilidad de que rompamos todo, esta puede ser una buena oportunidad.

17.2 Instalemos Jekyll

Existen un gran número de generadores de sitios estáticos. Pueden visitar https://www.staticgen.com/ para ver en detalle las características de cada uno. Luego de probar un par elegimos Jekyll por varios motivos: porque es el más utilizado y el que tiene una comunidad más grande y activa que redunda en mayor documentación; porque es el que nos recomendaron más personas; y, finalmente, porque habíamos asistido a una sesión de Jekyll para aprender los básicos. Así que vamos a ello.



Con feminismo, amigas, calma y mate ¡todo sale!

Recuerden que la instalación de Jekyll la haremos en la computadora/ordenador que utilizaremos para actualizar la web. Es decir, no hay que instalar Jekyll en la servidora. Dependiendo del sistema operativo que usen en su computadora necesitarán seguir unos pasos u otros. En la web de Jekyll se indican: https://jekyllrb.com/docs/installation/. Nosotras, como saben, daremos las instrucciones para instalarlo en un Debian.

Jekyll está desarrollado en un lenguaje de programación que se llama Ruby. Tranquilas que no tenemos que aprender Ruby para usarlo. Pero sí tenemos que instalar el paquete **ruby** y algunas herramientas de compilación para poder usarlo. Ejecutamos en la terminal:

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

Además debemos ejecutar lo siguientes comandos en la terminal para que podamos usar Ruby sin ser *root* (¡atención! es importante ejecutarlos sin ser *root* y uno a uno):

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

Para ahora instalar Jekyll ejecutamos:

```
gem install jekyll bundler
```

¡Bravo! ¡ya tenemos Jekyll instalado!

17.3 Nuevo sitio web estático con Jekyll

Para comenzar a crear su primera web estática tenemos que ir a la terminal y estando en el directorio donde queramos que se guarden los archivos de nuestra web ejecutaremos:

```
jekyll new nuestra_web
```

Donde dice "nuestra web" pueden poner el nombre que quieran. Es sólo identificativo no afectará a nuestra página en ningún sentido. Con este comando se creará un directorio que contendrá diferentes archivos y directorios con las configuraciones por defecto del tema Mínima https://github.com/jekyll/minima, que es el tema que trae Jekyll por defecto. Pueden ir al directorio y ver qué hay dentro. ¿Qué archivos ven? ¿Les suena alguno? ¿Al menos **index.html**? ¿Tienen algo así?

```
├── Gemfile
├── Gemfile.lock
├── _config.yml
├── _posts
│   └── 2016-12-04-welcome-to-jekyll.md
├── about.md
└── index.md
```

Editar y crear páginas en Jekyll

Lo primero que pueden hacer es editar el archivo **config.yml** para ajustar algunos parámetros generales. Lo pueden hacer desde

la consola con **nano**, como venimos haciendo, o con cualquier otro editor de texto que les guste:

```
title: el título de la web
email: el correo de contacto
description: de su web o su colectiva
baseurl: pueden dejarlo vacío (explicaremos luego)
url: pueden dejarlo vacío
twitter_username: usuaria de twitter
github_username: usuaria de github
```

Si no tienen Github o Twitter, por ejemplo, pueden borrar las líneas y no les aparecerá el iconito vacío. También se pueden agregar otras plataformas utilizando el kit de íconos Fontawesome https://fontawesome.com/ (pero no es prioritario ya que es un poco más avanzado).

Luego pueden editar el archivo **about.md**, que es la página del "Quiénes somos" de su web. Cuando la abran lo primero que verán será la cabecera. Esta nos da alguna información sobre la página que editaremos. En este caso el layout, el título y el link:

```
---
layout: page
title: About
permalink: /about/
---
```

Pueden cambiar el título para que esté en castellano y el permalink también:

```
---
layout: page
title: nosotras
permalink: /nosotras/
---
```

En esta página, por ejemplo, pueden escribir sobre ustedes y su proyecto. El *layout* no lo cambien porque es la que asigna el formato a la publicación. Guardamos y cerramos. Por ahora lo dejamos así. Las páginas automáticamente forman parte del menú de nuestro sitio web. Así que el título de la página ("Nosotras") aparecerá en el menú principal.

Pueden crear otras páginas en el mismo directorio (recuerden que tendrán que tener el **`layout: page`** en la cabecera).

Editar y crear posts en Jekyll

Probemos ver también cómo crear una publicación, eso que cotidianamente llamamos *post* y que se lista cronológicamente a partir de su fecha de publicación. Cada post será un archivo de markdown (**`extensión.md`**) por separado que estará guardado en el directorio **`_post.`** Si entramos a ese directorio veremos que hay uno de ejemplo. Podemos abrirlo para reconocer sus partes e intentar deducir qué hacen. Acá hay algo de documentación en inglés: https://jekyllrb.com/docs/posts/, pero seguramente hay mucha más en español en Internet.

Imagen de Silvana Riggio

Lo primero que pueden observar es el nombre del archivo que está compuesto por la fecha en formato año, mes, día, separado por guiones; y, luego, el título del post también separado por guiones. La parte del título en el nombre del archivo se convertirá en el slug, que es la parte final de la URL donde podemos identificar el nombre del recurso al que estamos accediendo. No es necesario que sea exactamente el título completo, pueden ser sus palabras clave para que no sea tan largo. Así que cada vez que queramos crear un nuevo post simplemente podemos hacer una copia de ese primer archivo, cambiarle el nombre y listo.

Abran el archivo **`20XX-XX-XX-welcome-to-jekyll.md`** de prueba para ver los datos que trae la cabecera:

```
---
layout: post
title: "Welcome to Jekyll!"
date: 2019-04-25 21:19:02 +0100
categories: jekyll update
---
```

De nuevo, el layout no lo vamos a tocar. Pero podemos editar las otras partes. El campo título (**`title`**) tiene el título de nuestro post y que deberá ir entre comillas (que luego no saldrán). En el campo de la fecha (**`date`**) ponemos la fecha y hora de publicación más el huso horario. Recuerden que los posts se listan cronológicamente así que la fecha es bien importante. Y, finalmente, en categorías (**`categories`**) podemos escribir categorías, separadas por espacios vacíos, que nos ayudarán a organizar el contenido. Debemos tener cuidado de escribir las categorías siempre igual, si usamos una mayúscula o una tilde las interpreta como categorías distintas. Este campo puede quedar en blanco también.

Existen otros parámetros que podemos agregar a la cabecera. Por ejemplo, Jekyll tiene la funcionalidad de guardar posts en borrador. Para ello tendríamos que agregar **`Published: false`** y luego usar un comando para publicarlo. También podemos "programar" posts para el futuro, publicando el post con una fecha posterior. Estas son funcionalidades avanzadas que no veremos ahora. Pero si les pica el gusanillo pueden averiguar.

Volvamos a nuestro archivo de markdown que se convertirá en un post. Debajo de la cabecera (identificada con "`---`") pueden es-

cribir el cuerpo de su publicación. Aunque parezca muy básico podemos hacer muchas cosas si conocemos la sintaxis de markdown: podemos añadir enlaces, imágenes, títulos de distinta jerarquía, listas ordenadas y sin ordenar, etc. Es bastante sencilla una vez que la aprendemos (de hecho esta guía la escribimos en markdown). Si no están familiarizadas pueden el artículo de Wikipedia sobre Ejemplos de sintaxis de Markdown.

Para insertar una imagen:

```
![Texto alternativo](URL de la imagen)
```

Para insertar un enlace:

```
[Texto del enlace aquí](URL de destino)
```

17.4 De markdown a HTML

Ahora toca ver cómo transformar esto en un sitio web. Para ello hay que ejecutar un comando que "construya" la web (que se haga esa fusión entre plantilla y contenido de la que hablamos) y podamos ver como queda.

Primero tenemos que situarnos en el directorio **`nuestra_web`**:

```
cd nuestra_web
```

Y luego ejecutar:

```
bundle exec jekyll serve
```

La terminal les devolverá algo así:

```
Configuration file: /home/usuaria/nuestra_web/_config.yml
Source: /home/usuaria/nuestra_web
Destination: /home/usuaria/nuestra_web/_site
Incremental build: disabled. Enable with --incremental
Generating
Jekyll Feed: Generating feed for posts
done in 0.292 seconds.
Auto-regeneration: enabled for '/home/usuaria/nuestra_web'
Server address: http://127.0.0.1:4000/
Server running... press ctrl-c to stop.
```

Lo que les está diciendo es que la web se generó en el directorio **`_site`** (que se acaba de crear con el comando anterior). Éste será un directorio importante donde encontraremos los archivos y directorios de nuestra web fresca para pasar a la servidora. Y también les está diciendo que pueden ver una "vista previa" de como quedó la web en http://127.0.0.1:4000/. Es decir tienen que ir al navegador de la computadora en donde están trabajando, copiar en en la barra de navegación http://127.0.0.1:4000/. ¿Les cargó la página? ¡Ya pueden ver como quedó la web con sus cambios! ¿Tiene todos los datos que añadimos? ¡Bien!

Mientras están visitando la web en su navegador pueden ver que la terminal se queda como esperando algo. Está así porque está como "sirviendo" la web. Así, si editan algo en **`about.md`** o en al-

gún post, automáticamente podrán ver los cambios en el navegador en http://127.0.0.1:4000/. Esto es una facilidad de Jekyll que nos ayuda a trabajar en tiempo real y ver cómo se van haciendo los cambios.

> **Importante:** como pueden observar, lo que hace Jekyll es tomar los datos de los diferentes archivos (**`_config.yml`**, **`about.md`**, los posts, etc) para generar todos los archivos que finalmente compone la web y situarlos en **`_site`**. Por lo tanto, lo que se genera dentro de **`_site`** es mejor no tocarlo, solo reservarlo para cuando queramos pasar la web a la servidora.

Cuando ya queramos que Jekyll termine de "servir" esta "vista previa" de la web podemos ejecutar **`Ctrl + C`** como nos dice en el diálogo de la terminal y así se terminará el proceso. Es importante terminar el proceso con **`Ctrl + C`** porque si no se quedará corriendo en segundo plano. Si no lo cerraron y luego les da problemas pueden terminar el proceso manualmente con el comando **`sudo kill <id del proceso>`** Para buscar el id del proceso pueden ejecutar el comando **`sudo ps a | grep "jekyll serve"`**.

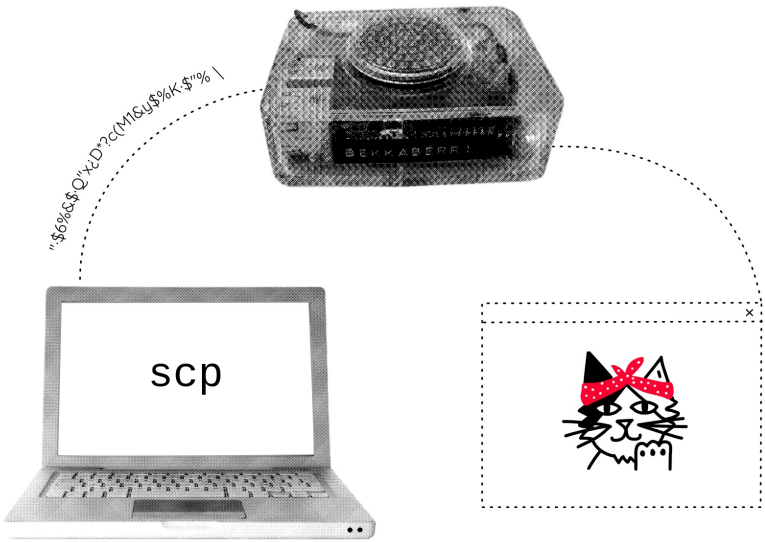
Si no necesitan ver vistas previas y solo quieren que reconstruya **`_site`** con los cambios nuevos pueden ejecutar sencillamente el comando:

```
bundle exec jekyll build
```

Recuerden ejecutarlo estando en el directorio **`nuestra_web`**. ¡Sigan probando hasta que les quede algo que les guste para subir a su servidora!

Jekyll puede parecer un poco raro al principio. Poco a poco se irán familiarizando y ampliando las posibilidades de lo que pueden hacer y cambiar. Sobre todo agarrarle la mano a los temas de formato y los estilos de la web.

Esta guía se nos queda corta para explicar más, aquí les dejamos documentación que nosotras seguimos: https://www.taniarascia.com/make-a-static-website-with-jekyll/ y https://jekyllrb.com/docs/



18. Pasemos la web a la servidora

Ahora nuestro objetivo será subir nuestra web a la servidora y que todo Internet pueda verla. Para eso habrá que pasar el contenido de **`/home/usuaria/nuestra_web/_site`** en nuestra computadora al directorio **`/var/www/html/`** en la servidora. Lo haremos copiando los archivos desde la terminal con el comando **`scp`** que sirve para transferir archivos a través de una conexión SSH (¡sí!, la misma que usamos para conectarnos remotamente a la servidorcita). Pero antes necesitamos solucionar algunos temas de permisos que nos permitan hacer esta transferencia.

Para que la usuaria **`pi`** (la usuaria de la servidora) pueda editar el directorio root de Apache (**`/var/www/html`**) debemos incorporar a **`pi`** en el grupo **`www-data`**. Para ello hay que ejecutar en la servidora:

```
sudo usermod -a -G www-data pi
```

Y hay que editar a quién pertenece el directorio *root* de Apache. Ejecutamos:

```
sudo chown -R pi:www-data /var/www/html/
```

Ahora sí podemos ir a nuestra computadora y pasar los archivos a la servidora ejecutando el siguiente comando:

```
scp -P 2251 -r /home/usuaria/nuestra_web/_site/*
pi@nuestrodominio.net:/var/www/html/
```

> **¡Atención!** Como cambiamos el puerto SSH debemos especificar aquí también que la transferencia la haga en el puerto que configuramos en el apartado 12.1 de la parte IV. La **P** en el modificador **-P** debe ir en mayúscula. Y el modificador **-r** nos copiará todos los archivos de manera recursiva.

Si todo sale sobre ruedas nos pedirá la contraseña de nuestra llave SSH. Si nuestra llave SSH está bien configurada no debería haber problemas. En caso de que nos muestre un error no nos preocupemos. Siempre tendremos que leer qué nos devuelve el sistema para tratar de corregirlo. Si nos devuelve un error de permisos volvamos a cerciorarnos que pusimos los permisos bien en el paso anterior.

¡Viva, viva, nuestra web ya está en la servidora!

Permisos de los archivos y directorio de la web

Según hemos comprobado Jekyll otorga permisos al directorio **_site** ya óptimos para pasarlos a la servidora. Es decir, **755** para directorios y **644** para los archivos. Pero, si por cualquier cosa se lían, pueden ir al directorio *root* de Apache con el comando:

```
cd /var/www/html
```

y ejecutar los siguientes comandos que darán permisos **755** para

directorios y **644** para los archivos:

```
sudo find . -type d -exec chmod 755 {} \;
sudo find . -type f -exec chmod 644 {} \;
```

Mucho cuidado al ejecutar estos comandos. Corroboren que están en la ruta **/var/www/html**.

El comando para revisar los permisos de un directorio es **ls -l.** Debería devolverles algo así:

```
-rw-r--r-- 1 pi www-data 3495 mar 26 13:21 404.html
drwxr-xr-x 2 pi www-data 4096 mar 26 13:21 about
drwxr-xr-x 2 pi www-data 4096 mar 26 13:21 assets
-rw-r--r-- 1 pi www-data 3036 mar 26 13:21 feed.xml
-rw-r--r-- 1 pi www-data 3563 mar 26 13:21 index.html
drwxr-xr-x 3 pi www-data 4096 mar 26 13:21 categoria1
```

Les recomendamos que revisen documentación en Internet para entender bien cómo funcionan los permisos en GNU/Linux. Pero les adelantamos algunas cosas. Para entender los permisos de cada directorio o archivo hay que saber que lo primero que nos dice es si es un directorio (**d**) o un archivo (**-**). Luego le siguen tres grupos de tres caracteres: **r**, **w**, y **x** que significan *read* (lectura), *write* (escritura) y *execute* (ejecución) o un guión si no tiene permiso. Cada uno de estos tres grupos de caracteres se va a referir a los permisos de la propietaria del directorio o archivo (si pone **rwx** significa que tiene todos los permisos). Luego le siguen otras tres letras que se van a referir a los permisos del grupo al que pertenece el directorio o

archivo (si pone **r--** quiere decir que sólo tiene permisos de lectura). Luego le siguen las últimas tres letras que se refieren a los permisos que tiene cualquier otra usuaria (si pone que **r-x** quiere decir que tiene permisos de lectura y ejecución).

Hay una correlación entre los números de los comandos y los permisos (quitando el primer campo de si es **d** ó **-**):

Las r valen 4

Las w valen 2

Las x valen 1

Para llegar al número de tres cifras que representa los permisos se suman las tres primeras (que son los permisos del *owner* o propietaria) para el primer campo; se suman las tres del medio (que son los permisos del grupo) para el segundo; y, para el tercero, se suman las tres ultimas (que son los permisos del cualquier otra usuaria).

Por lo tanto:

rw-r--r-- equivale a 644

rwxr-xr-x equivale a 755

Configurar `.htaccess`

El archivo **.htaccess** sirve para configurar muchas cosas de Apache pero en este caso sólo vamos a usarlo para indicar cuál es la página de error que queremos que se muestre cuando no se encuentre un contenido en nuestra web (el famoso error 404 o error

400) o para cuando el acceso no esté autorizado (error 401 o error 403). Jekyll crea una página de error **404.html** por defecto que también estaba en *_site*. Es la que usaremos para todos estos errores. Primero debemos habilitar el uso de este **.htaccess** editando el archivo de configuración de Apache **/etc/apache2/apache2.conf** y en la siguiente sección poner **All** después de **AllowOverride**:

```
<Directory /var/www/>
        Options FollowSymLinks
        AllowOverride All
        Require all granted
</Directory>
```

Después hay que crear el archivo .**htaccess**:

```
sudo nano /var/www/html/.htaccess
```

Le añadimos el siguiente código que denegará el acceso al propio archivo:

```
# STRONG HTACCESS PROTECTION`
<Files ~ "^.*\.([Hh][Tt][Aa])">
    Order allow,deny
    Deny from all
    Satisfy all
</Files>
```

Luego, más abajo, le indicaremos el archivo HTML al que debe redirigir. Para ello le añadimos las líneas:

```
ErrorDocument 400 /404.html
ErrorDocument 401 /404.html
ErrorDocument 403 /404.html
ErrorDocument 404 /404.html
```

Recuerden grabar y salir. Nosotras usamos la misma página para todos los errores. Lo propio sería hacer un archivo html para cada error.

Después, para proteger mejor el **.htaccess** le daremos los permisos **640**:

```
sudo chmod 640 /var/www/html/.htaccess
```

En GNU/Linux los archivos que tienen un punto "**.**" delante del nombre, como es el caso de **.htaccess**, se hacen invisibles, ocultos. Así que para listar los archivos de un directorio y que se vean también los archivos ocultos ejecutamos:

```
ls -la
```

Les debería salir algo así:

```
drwxr-xr-x 5 pi   www-data 4096 abr 25 14:04 .
drwxr-xr-x 3 root root     4096 mar 26 13:18 ..
-rw-r--r-- 1 pi   www-data 3495 mar 26 13:21 404.html
drwxr-xr-x 2 pi   www-data 4096 mar 26 13:21 about
drwxr-xr-x 2 pi   www-data 4096 mar 26 13:21 assets
-rw-r--r-- 1 pi   www-data 3036 mar 26 13:21 feed.xml
-rw-r----- 1 pi   www-data  235 abr 25 14:04 .htaccess
-rw-r--r-- 1 pi   www-data 3563 mar 26 13:21 index.html
drwxr-xr-x 3 pi   www-data 4096 mar 26 13:21 categoria1
```

Finalmente debemos reiniciar el servicio apache2 para que los cambios tengan efecto:

```
sudo service apache2 restart
```

Si en algún momento vuelven a usar el comando **`sudo find . -type f -exec chmod 644 {} \`**; tengan cuidado porque esto también cambiará los permisos de **`.htaccess`**, así que tendrán que volver a hacer **`sudo chmod 640 /var/www/html/.htaccess`** para mantenerlo como lo teníamos (¡bien seguro!).

Si quieren profundizar en las configuraciones del **`.htaccess`** pueden chequear este recurso: https://ayudawp.com/todo-sobre-htaccess/#2_Notas_importantes_para_novatos_con_htaccess)

19. Pánico: ¡el sitio web no carga!

Esto nos ha pasado varias veces: funciona todo hasta que deja de funcionar. Por eso armamos una pequeña lista de algunas cosas que podemos revisar cuando esto nos ocurra para identificar el problema y, a partir de allí, pensar en una solución. Lo más importante en estos casos es no desesperar y leer con calma las pistas que nos da el sistema.

- Prueben hacer un **`ping`** en la terminal (de cualquier compu conectada a Internet): **`ping nuestrodominio.net`** y ver cómo responde. Si no devuelve datos, ¡malas noticias!, toca revisar las configuraciones.

NUESTRAS WEBS, NUESTRAS REGLAS

- Con el comando **`sudo service apache2 status`** pueden saber el estatus de la servidora y ver hace cuánto tiempo está corriendo. De ese modo sabrán si se detuvo en algún momento.
- Revisen lo que hay en el *root* de Apache en términos de contenidos y permisos (ver apartado 18).
- Revisen el estado del *firewall* (ver apartado 13.5).
- Acérquense físicamente al lugar donde está conectada nuestra servidora y vean si la conexión de Internet está arriba o si alguien desenchufó la máquina.
- Pueden revisar también si es que el Registro A del servicio de DNS dinámico sigue configurado o si cambió algo en el panel de administración de nuestro dominio. En tal caso volvemos a seguir los pasos del apartado 10.4.
- Por último pueden revisar la configuración del router y ver si la DMZ está bien (ver apartado 10.3).

Este es un buen momento para recordarnos que este es un proceso colectivo y que no tenemos por qué saber de todo. Preguntemos a la comunidad y leamos documentación para tratar de entender qué es lo que pasa. A veces nos puede ayudar hacer un diagrama para ir descartando errores. Y si nada de nada anda, ¡empecemos de nuevo! Que nadie se ha muerto de esto y de paso aprendemos más mientas nos divertimos con las amigas.

20. Ánimos y despedida

Amigas, ¡terminamos! Si llegaron hasta aquí significa que tienen su web montada en una servidora autónoma y feminista. Hay muchas cosas que se pueden hacer con ella. Es un paso importante en la construcción de autonomía feminista.

En esta ocasión propusimos la instalación de una servidora web pero perfectamente pueden instalar otros servicios: Nextcloud, Etherpad, hasta un servidor Icecast para tener un *streaming* de radio. ¿Qué necesitan? ¿Qué quieren? Podemos pensar juntas en cómo ampliar esta guía.

Probablemente hayan muchas cosas que no salieran a la primera o a la segunda. Lo sabemos, lo vivimos. A nosotras nos sirvieron la paciencia, la lectura sosegada y, sobre todo, compartir las frustraciones. No hace falta tampoco que hagan todo de un tirón. Pueden ir paso a paso, asegurándose de comprender el proceso y dejar que los conocimientos se asienten.

Queremos conocerlas, queremos ver y enlazar nuestras servidoras webs para armar comunidades transhackfeministas en Internet. ¿Nos escriben? Queremos saber cómo les fue para seguir ajustando los pasos de esta guía.

¡Un abrazo transhackfeminista!

la_bekka, octubre 2019.

labekka.red