# Tutorial para configurar mod_rewrite en Apache sobre Ubuntu

Para los que "montamos" páginas web siempre es útil ocultar las extensiones de nuestros archivos a ser servidos y así despistar a posibles atacantes *pero es mejor aún para que nuestros usuarios les sea más fácil recordar nuestras secciones e incluso promociones publicitarias, ¿ya tenemos vuestra atención?*

## Introducción.

En un mundo donde ya prácticamente todo el mundo introduce una búsqueda en Google o DuckDuckGo para llegar a nuestra página web -exponiéndose a redirecciones hacia páginas falsas por los buscadores web- **pues nosotros nos empeñamos en utilizar la barra de direcciones de nuestro navegador web recomendado,** Mozilla Firefox**.**

Es así entonces que podemos publicar alguna promoción en algún medio de comunicación como televisión o prensa y queremos que se pongan en contacto con nosotros por medio de una página web que a tal efecto hayamos programado para ello. Dicha dirección podría ser:

```
nuestrodominio.com/contacto
```

Como véis no incluimos "www" ni tampoco las dobles barras (de hecho Tim Berners-Lee se disculpó públicamente por haber hecho la redundancia y hacernos escribir billones de veces ese caracter adicional) ni "http:" -la mayoría de los navegadores web completan automáticamente la dirección- *pero mejor aún hemos obviado la extensión del archivo .html, algo que consideramos podría confundir a cualquier usuario común* (sí, que hasta los "nativos digitales" también se pueden equivocar).

La intención de este tutorial es CONFIGURAR nuestro servidor web para que acepte dicha dirección y redirija a nuestros posibles clientes de manera transparente y así, a futuro, trabajemos y aumentemos nuestros ingresos ;-) .

## Prefacio.

Aunque no estamos escribiendo un libro, estas páginas web algún día las podríamos recopilar en un ejemplar, por eso el uso de la palabra "prefacio". Lo que necesitaremos para nuestra práctica didáctica es un ordenador con Ubuntu 16 instalado (que también vale este tutorial para Ubuntu 14) al cual le instalaremos **Apache2**. **Para nuestro caso nuestra máquina tiene una dirección IP fija en nuestra red área de local 192.168.1.47 CAMBIAD lo que veaís aquí por vuestra propia**

**dirección IP local.** Lo que acá escribimos es válido también para cuando queráis montar vuestro servidor de producción en la internet y solo está limitado por las restricciones que os imponga vuestro proveedor de hospedaje. *Este tutorial es completo y el escenario sería o bien una máquina virtual o real que alquileís a un proveedor de alojamiento (es más costoso que alojamiento compartido) o vuestro ordenador con una dirección IP fija que os venda vuestro ISP y que os conécteis bien sea por "ADSL" o "frame relay".*

## Instalando Apache Web Server:

Abrimos una ventana de comandos (si no sabéis cómo, leed nuestro tutorial) e introducimos el siguiente comando:

```
sudo apt-get update  sudo apt-get install apache2
```

La primera línea actualiza los paquetes disponibles para nuestra versión GNU/Linux que tengamos instalado (Ubuntu 16 en este ejemplo) y la segunda línea instala el servidor web Apache2, software que ha evolucionado mucho desde los años 90. Nótese que usamos "sudo" para tener derechos de administrador o "root" y deberemos colocar la contraseña respectiva (luego el sistema obvia esta solicitud si corremos varios comandos seguidos en corto período de tiempo). Podremos ver algo muy parecido a esto, si todo sale bien:

Para probar que tenemos nuestro servidor instalado podemos navegar hasta nuestra propia dirección IP *o en su defecto podemos utilizar la palabra clave "localhost"* ***y eso sí, en la barra de direcciones de nuestro navegador (los buscadores web NO mostrarán enlace hacia nuestro servidor).***

## Habilitando mod_rewrite.

Antes de meternos de lleno con mod_rewrite cumplimos con avisaros que también existe la función mod_alias que es muchísimo más sencilla para resolver rápidamente ciertas situaciones -e incluso cuenta con los comando más avanzados llamados "" y ""- .

Pero si queremos evolucionar y disponer de una herramienta poderosa (y compleja) estudiaremos mod_rewrite. Para activar mod_rewrite debemos ejecutar las siguiente líneas:

```
sudo a2enmod rewrite  sudo service apache2 restart
```

La segunda línea sirve para que el servicio o "daemon" reinicie con la configuración deseada. En el ambiente GNU/Linux son muy parcos y estoicos, si todo va bien simplemente veréis de nuevo el indicador de la línea de comando "prompt" -pero al finalizar el primer comando nos indica que falta alguna instrucción que sea importante para finalizar el comando-. ES DECIR no espéreis a cada rato mensaje de confirmación de comando ejecutado -otra cosa, en raras y contadas ocasiones deberemos hacer un reinicio completo de nuestro ordenador, con reiniciar los servicios respectivos tiene para seguir funcionando luego de nuestros cambios-.

Como ya sabéis los comandos en GNU/Linux diferencian letras mayúsculas de minúsculas y por supuesto debemos escribir muy bien los comandos, *no como nosotros que nos equivocamos pero luego correjimos, mirad:*

## Configurando .htaccess

Ahora debemos crear un archivo (oculto, mirad el puntito que precede el nombre del fichero) muy importante en el funcionamiento de Apache. Si lo tenemos ubicado en el directorio raíz de nuestro servidor web se aplicará a todas las subcarpetas que tengamos pero cada subcarpeta podrá tener su propio ".htaccess" que regirá para la carpeta donde esté ubicado (y a su vez las subcarpetas que contenga) *obviando las instrucciones que hayamos colocado en nuestro ".htacces raíz".*

Pero antes, por razones de seguridad, debemos tener acceso (desde Apache) para crear y escribir dicho fichero, usamos **nano** -o vuestro editor de texto favorito- para modificar el siguiente archivo:

```
sudo nano /etc/apache2/sites-enabled/000-default.conf
```

Tras dentro del cual debemos insertar lo siguiente (notad los comentarios que insertamos en castellano en la imagen más abajo:)

```
        Options Indexes FollowSymLinks MultiViews       AllowOverride Al
l       Order allow,deny       allow from all
```

Es sumamente importante indentar las líneas (en nano lo podéis hacer rápidamente pulsando la tecla TAB para cada sangría) y respetar mayúsculas y minúsculas. Luego debemos reiniciar el servicio web Apache con el consabido comando:

```
sudo service apache2 restart
```

Como información adicional os contamos que el archivo "000-default.conf" es el archivo por defecto del primer dominio que sea servido por nuestro ordenador, *es decir, podremos a futuro alojar varios dominios en una sola máquina. De este modo* el segundo dominio utilizará el archivo "001-default.conf" y así sucesivamente, pero ¿cómo reconoce nuestro servidor web cual dominio entregar? Estad atentos más adelante en este tutorial os daremos noción de ello -y a futuro publicaremos una entrada totalmente dedicada a configurar un servidor Apache con varios dominios virtuales-.

### Creando .htaccess

Ya que tenemos configurado a Apache para que acepte nuestro .htaccess procedemos a crearlo. Para ello debemos utilizar nuestro editor de archivos de texto favorito (usaremos **nano** en este caso) y escribiremos lo siguiente:

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

Y le agregamos una sola línea:

```
RewriteEngine on
```

Guardamos y luego debemos garantizar su acceso para los demás usuarios, nosotros o cualquiera otro usuario que edite nuestra página web (recordad que lo creamos con la credencial de administrador o "root"):

```
sudo chmod 644 /var/www/html/.htaccess
```

¡Y listo! Ya tenemos configurado nuestro .htaccess pero como decía Santo Tomás "hasta no ver, no creer" así que vamos a probar si es verdad que funciona.

## Creando nuestra pequeña página web.

Como dijimos al principio haremos una página de contacto ***sin mayores pretensiones***, en este ejemplo indicando nuestro correo electrónico de manera tal que no sea capturada por los robots

que esparcen "spam" a diestra y siniestra (para una explicación más a fondo visitad nuestro tutorial sobre HTML5):

## Correo electr&oacutenico:

```
contacto EN 192.168.1.47
```

Como veís con el lenguaje PHP buscamos la palabra clave "contacto" para notificar que el cliente vio la promoción por televisión, *el otro único valor posible es "correoe" ya que la regla mod_rewrite filtra lo que escribe el usuario.*

- Con el comando $_GET['origen'] obtenemos la cadena $1 del comando RewriteRule.
- Con el comando strtolower() lo convertimos todo a minúsculas.
- Con el comando strpos() obtenemos FALSO si no se haya la palabra clave, de lo contrario devuelve un valor mayor o igual a cero.
- Teniendo en cuenta lo anterior se debe utilizar la comparación ($contacto !== false) por negación para obtener el resultado correcto.

Hasta acá hemos considerado que el usuario o visitante escriba ya sea "correoe" o "contacto" -con todas sus variantes, con o sin extensión (.html o .htm) pero **¿Qué ocurriría si, dado el caso, el navegante escribiera "correo", es decir, obviara la última letra?** *En este caso nuestro servidor simplemente devolvería un mensaje de error 404 como este:*

Podemos, con propósitos didácticos, ampliar nuestro estudio de Rewrite con un comando adicional: **RewriteCond**.

## Reglas de "RewriteCond".

El comando "RewriteCond" nos permite evaluar una condición y de ser cierta ejecutará la siguiente e inmediata línea "Rewrite" de lo contrario la saltará y seguira evaluando el resto de las líneas de nuestro fichero .htaccess (si es que hubieran más líneas). Otro punto a tener en cuenta es que se pueden tener varias líneas RewriteCond contínuas una después de la otra y si alguna se cumple se ejecutará la siguiente e inmediata línea "Rewrite".

Veamos la sintaxis de RewriteCond:

`RewriteCond` *`cadena condicion banderas`*

- *RewriteCond*: el comando en sí mismo que debe ser sucedido en la siguiente línea con un comando "RewriteRule".
- *Cadena*: lo que vamos a comprobar, la entrada de datos.
- *Condicion*: lo que le vamos a aplicar a la cadena, con la que vamos a comparar y que nos devolverá un valor verdadero o falso.
- *Banderas*: modifican el comportamiento de la condicion y va entre corchetes y separadas por comas, si son varias y tal como lo vimos con el comando "RewriteRule".

Como vemos ya tenemos la teoría, ahora vayamos a la práctica. Seguiremos con nuestro ejemplo anterior: ¿qué sucede si el visitante de nuestra página web escribiera mal la dirección web (URI) y obtuviera un mensaje con error 404?

Podríamos "redirigir" la dirección solicitada hacia la página principal de nuestro servidor web de manera transparente con nuestro archivo .htaccess pero primero una advertencia:

---

***Esta NO es la manera correcta de tratar una visita a nuestra servidor web que no contenga, por alguna razón (página borrada, error del usuario, etc) una URI solicitada. Lo "legal" es manejar el error 404 con una página web personalizada (de hecho la que vemos es la que trae Apache por defecto) y recomendamos que transcurrido un tiempo necesario para su lectura sea redirigida automáticamente hacia nuestra página web principal (esto escapa a este tutorial pero podéis hacerlo con JavaScript o incluso PHP).***

Para configurar nuestra propia página web personalizada para el error 404 podemos hacerla a nuestro gusto/diseño y nombrarla, por ejemplo, "404_personalizado.html" y ubicarla en el directorio raíz para luego agregar a nuestro archivo .htaccess la siguiente línea:

```
ErrorDocument 404 /404_personalizado.html
```

---

Retomando nuestro ejemplo podemos agregar a nuestro .htaccess las siguientes líneas que

evaluarán y tratarán la entrada de la URL recibida **y la ubicaremos al final del fichero para que sea ejecutada en último lugar** (si la colocamos antes al llegar una consulta por "/correoe" automáticamente será redirigida a la página principal):

```
AddType application/x-httpd-php .html .htm  RewriteEngine on  RewriteRule
 ^(contacto|contacto.html|correoe)$ contacto.html?origen=$1 [NC,QSA]



RewriteCond %{REQUEST_FILENAME} !-f  RewriteRule ^(.*)$ %1
```

Las 3 primeras líneas ya las hemos explicado y aprendido ahora expliquemos las dos líneas agregadas:

- **"RewriteCond":** El comando que nos permite evaluar una condición.
- **"%{REQUEST_FILENAME}":** El archivo solicitado, luego de haber recibido la URI (con confundir con {REQUEST_URI} ).
- **"!":** es un operador lógico que invierte un valor, por ejemplo de falso a verdadero.
- **"-f":** Le indica a nuestro servidor que compruebe si es un archivo es "regular", esto es, que esté bien escrito y que exista en nuestro ordenador -y devuelve verdadero si esto se cumple-.
- *Si no se cumple el punto anterior y devuelve falso con el operador "!" lo convertimos en verdadero y tenga la respuesta positiva para ejecutar la siguiente línea "RewriteRule".*
- *"RewriteRule":* con la respuesta positiva recibida comienza a trabajar comparando con el patrón siguiente.
- **"^(.*)$"**: los metacaracteres "^" y "$" ya los conocemos, los nuevos son "." y "*" que indican que compare -y acepte- cualquier caracter y cualquier cantidad de los mismos, es decir, acepte toda entrada posible (el punto indica que puede ser *cualquier caracter* y el asterisco indica *que dicho caracter se puede repetir cualquier cantidad de veces*). Como cumple con el patrón procede a aplicar la regla.
- ¿Recuerdan la variable "$1" que nos permitía elegir que opción de un patrón recibíamos? **Ahora con la variable "%1" nos permite pasar intacta la referencia recibida de la "RewriteCond" (la parte que corresponde a nuestro dominio) que desencadenó la ejecución de la "RewriteRule".**

En este punto consideramos prudente mostrar de forma un tanto gráfica la nomenclatura de

## Un último ejemplo de "RewriteCond".

Para finalizar este tutorial vamos a proponernos el bloquear el acceso a una dirección IP4 en

particular. Para nuestros propósitos didácticos tenemos una red de área local con nuestro servidor web en 192.168.1.47 (la misma a lo largo de toda esta entrada) y una máquina virtual con la dirección IP4 192.168.1.78 la cual queremos bloquear y para ello agregaremos lo siguiente:

```
RewriteCond %{REMOTE_ADDR} ^(192\.168\.1\.78)$  RewriteRule (.*) - [F,L]
```

Pasamos a explicar línea por línea y comando por comando:

- **"RewriteCond":** comando para evaluar una condición.
- **"%{REMOTE_ADDR}":** una variable que contiene la dirección IP del visitante.
- **"^" y "$":** delimitadores izquierdo y derecho de la dirección IP recibida.
- **"(" y ")":** cadena a evaluar con expresion regular.
- **"\":** la barra invertida sirve para indicarle a nuestro servidor web que el punto a la derecha lo considere como símbolo y NO como metacaracter (ver ejemplo anterior de error 404).
- Los números pues son la dirección IP a bloquear.
- "RewriteRule": es la línea que sirve a la(s) última(s) "RewriteCond".
- **"(.*)":** indica que acepte cualquier cadena de texto -ver ejemplo arriba-.
- **"-":** un guión para que NO sustituya nada ya que la bandera hará el trabajo por nosotros -ver siguiente punto-.
- **"[F,L]":** son dos banderas con dos instrucciones diferentes, la letra "F" le indica que envie el "mensaje" de prohibido (en realidad devuelve un "error 403" el cual también podemos personalizar con un fichero .html a nuestro gusto y conveniencia tal y como lo hicimos con el "No encontrado 404"). **La letra "L" es muy importante e indica que es la última ("last") regla a aplicar y que no revise las siguientes (si las hubiere) líneas en .htaccess . Esto es así porque si de plano le prohibimos el acceso ¿para qué vamos a revisar reglas adicionales?**

La probamos y veremos algo muy similar a esto:

*Si por el contrario queremos que solamente esa dirección IP tenga acceso al servidor web (por ejemplo queremos"administrar" nuestro servidor desde esa única y exclusiva dirección IP, sin que nadie nos moleste) simplemente antecedemos el caracter lógico "!" que inverte la respuesta de la comparación.*

## Fuentes consultadas:

### En idioma castellano:

- «Teoría sobre Apache" por Sapoclay.
- «Directivas básicas de configuración de Apache 2».

### En idioma inglés:

- "Apache on Ubuntu Linux For Beginners" by Carla Schroder.
- "How To Set Up mod_rewrite for Apache on Ubuntu 14.04" by Alvin Wan and published at Digital Ocean.
- "RewriteRule Flags" by Apache.org
- "Apache Module mod_rewrite" by Apache.org
- "strpos" at PHP.org
- "Using .htaccess to make all .html pages to run as .php files?" at StackoverFlow.
- "Example .htaccess Code Snippets"
- "Apache mod_rewrite Introduction" by Apache.org
- "Apache Module mod_alias" by Apache.org
- "mod_rewrite Cheat Sheet" by DaveChild.
- "How To Set Up Mod_Rewrite" by Etel Sverdlov.
- "How To Set Up Mod_Rewrite (2)" by Etel Sverdlov.
- "An Introduction To Regular Expressions" at DigitalOcean.
- "Useful .htaccess Snippets" by WPAleks.

#### Enlaces hacia expresiones regulares:

- "Regex" at ComputerHope.
- "Regular expressions quick reference" at ComputerHope.

## En idioma japonés:

- "Mod_Rewrite??????????????????????"