

Instalación y administración de un servidor linux

Guillermo Vicente González

Jorge Prieto de la Cruz



**VNiVERSiDAD
D SALAMANCA**

CAMPUS DE EXCELENCIA INTERNACIONAL

Introducción	4
Configuración inicial	4
Usuario administrador	4
Hostname y motd	4
Configuración de red	5
perl	5
ssh	6
Usuarios.	7
Openldap	7
Base de datos.	10
Apache2	11
Modulo CGI	11
Conexiones web seguras	12
Interfaz principal.	13
landing page (index.html)	13
Sign Up	14
Control de usuarios no registrados	16
Correo.html	16
Login	17
Landing.cgi	18
Modificar datos	19
Cambiar contraseña	20
dar de baja	21
Cerrar sesion.	21
Apuntes	21
Correo	22
Solicitar web	24
forgottenPass.cgi	24
Monitor.cgi	24
Pantallas de error.	25
Cuotas	25
Backups.	25

Monitorización	26
Wordpress	27
Otras funcionalidades.	31
Mensaje de login al root.	31
Crontab	32
Conclusiones	33

Introducción

Para la realización de este trabajo hemos utilizado una instalación mínima de Debian virtualizada en una torre mediante el programa vmware. Por comodidad, los scripts se han escrito en otros ordenadores y se han subido al servidor por medio de sftp. El resto de configuraciones se han realizado mediante conexión ssh como usuario root.

Configuración inicial

Usuario administrador

Aunque la mayor parte del trabajo se ha realizado como root, se ha creado un usuario admin para que una vez el servidor esté listo, se deshabilite el login como root y el admin sea un usuario con capacidades similares. Para crear el usuario administrador hemos seguido los siguientes pasos.

1. creamos el usuario

```
adduser admin
```

2. cambiamos la contraseña del root

```
passwd
```

3. Instalamos la herramienta sudo

```
apt install sudo
```

4. añadimos al usuario admin a sudoers

```
usermod -aG sudo admin
```

Hostname y motd

A parte de esto, en el trabajo se indica que tenemos que crear una empresa hipotética, es por esto que hemos modificado el fichero hostname para que muestre el nombre de nuestra empresa. El nombre que hemos escogido para nuestra empresa hipotética es pecera, ya que los autores de la práctica hicimos un primer año de carrera en Zamora, donde existía un aula que frecuentamos llamada “la pecera”. No se nos ocurría nada mucho mejor

```

root@pecera:~# cat /etc/hostname
pecera.local
root@pecera:~# hostname
pecera.local
root@pecera:~# █

```

Por último, hemos cambiado el contenido del fichero `/etc/motd` para que muestre un mensaje más apropiado para nuestra empresa.

```

root@pecera:~# cat /etc/motd
Bienvenido a Pecera Dev. No se que poner aqui pero me piden que modifique este f
ichero así que modificado queda
No seas gañan y respeta las normas, no intentes hackearme que me enfado
Un saludo y pasalo bien, nos vemos en mc donalds
root@pecera:~# █

```

Configuración de red

A parte de esto también intentamos modificar el fichero `/etc/network/interfaces` para obtener una ip estatica, ya que esto facilita la realización de otros apartados de la práctica como la instalación de wordpress, o de bind9, pero no logramos dar con la configuración adecuada y decidimos dejar la que venía por defecto. Se adjunta el contenido del fichero con las líneas para la dirección ip estática comentadas.

```

GNU nano 5.4 /etc/network/interfaces
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

source /etc/network/interfaces.d/*

# The loopback network interface
auto lo
iface lo inet loopback

auto ens33
iface ens33 inet dhcp
# The primary network interface

#allow-hotplug ens33
#iface ens33 inet static
#    address 192.168.0.147
#    netmask 255.255.255.0
#    network 192.168.1.0
#    broadcast 192.168.1.255

```

perl

para poder utilizar scripts con módulos de perl instalamos `apt install cpanminus`.

ssh

El servicio ssh es fundamental para la práctica ya que a parte de ser requisito fundamental de la misma nos permite conectarnos de forma remota a nuestro servidor mediante consola de comandos. En la iso de debian que descargamos venía ya descargada, pero hemos hecho algunos ajustes en los ficheros de configuración.

En primer lugar, se indica que los usuarios de nuestro servicio han de ser capaces de usar un protocolo seguro para modificar sus archivos y gestionar aspectos de su usuario, pero que han de estar enjaulados. Esto se logra por medio del archivo de configuración `/etc/ssh/sshd_config`, en el que hemos especificado que los usuarios que pertenezcan al grupo alumnos (se detallará esta decisión mas adelante) no pueden salir del directorio `/home` y ademas solo pueden logearse por medio de sftp, no con ssh. Hemos decidido que los profesores si puedan de manera similar a como funcionan los propios servidores del departamento de informática como nogal o encina.

Las líneas que hemos añadido al fichero de configuración son las siguiente

```
Port 2222
LoginGraceTime 2m
PermitRootLogin no
Subsystem sftp internal-sftp
Match group alumnos
    ChrootDirectory /home
    AllowTcpForwarding no
```

Port 2222 nos permite especificar qué puerto escuchará las peticiones . El puerto por defecto es el 22 pero al ser esto un “estándar” también facilita la labor a los atacantes, así que lo hemos cambiado por seguridad.

El login grace time especifica el tiempo de gracia, que permite configurar el tiempo que tendrá el usuario para loguearse antes de que ssh se desconecte. Nosotros lo hemos establecido a 2 minutos, que es el que viene comentado por defecto.

Por último, tras terminar la práctica hemos desactivado la opción `PermitRootLogin` para que no se pueda hacer login por ssh como usuario root.

Las líneas restantes del fichero restringen a los usuarios del grupo alumnos al directorio home. Para poder enjaular a los usuarios, la carpeta `chrootDirectory` ha de ser del root. Existen soluciones más complejas para crear un directorio chroot, pero hemos decidido dejarlo aquí por incompatibilidades con otros apartados de la práctica.

Usuarios.

Existen dos tipos de usuarios, profesores y alumnos. La traducción directa de este enunciado a su implementación es la creación de dos grupos, usuarios y alumnos. Se han otorgado permisos distintos a los dos grupos para cumplir con los requisitos de la práctica y atendiendo a nuestro propio criterio en cuanto a seguridad, o sencillamente lógica. El ejemplo más claro de esto es el directorio apuntes, que se especifica que puede ser accedido por todo el mundo pero solo pueden subir apuntes los profesores. La manera de lograr esto es cambiando el grupo propietario del directorio apuntes y otorgando todos los permisos de grupo pero solo de lectura y ejecución a todos los usuarios. Los comandos utilizados son los siguientes:

```
groupadd profesores
groupadd alumnos
chgrp profesores /home/apuntes
chmod 775 apuntes
```

El directorio apuntes se encuentra /home ya que ha de ser accesible por todos los usuarios para que suban apuntes, y al estar enjaulados si estuviera fuera de home no podrían acceder. Los apuntes han de ser accesibles desde el exterior también así que se ha creado un enlace simbólico en el directorio /var/www/html para que sea visible como una web. Esto se logra con el siguiente comando.

```
ln -s /home/apuntes /var/www/html/apuntes.
```

El enlace simbólico si es accesible desde el exterior.

Openldap

Aunque contamos con nuestra base de datos personal (se describe en el siguiente apartado) donde almacenar datos de los usuarios, hemos optado por instalar también ldap ya que facilita ciertas de las tareas de gestión de manera más segura que otras alternativas como es por ejemplo la creación de directorios personales o la instalación automática de cuotas. La instalamos con los siguientes comandos

```
apt install slapd ldap-utils ldapscripts
dpkg-reconfigure slapd
#introducimos los nombres de dominio que queremos, en nuestro caso
pecera.local
apt install libpam-ldap
```

Una de las cosas que permite ldap es la creación de los directorios /home/usuarios automáticamente la primera vez que se loguean en el sistema. Esto es posible gracias al fichero /etc/pam.d/common-account, en el que añadiremos las siguientes líneas.

```
session required pam_mkhomedir.so skel=/etc/skel/ umask=0022
session required pam_exec.so /usr/bin/perl /var/pecera/crearCuotas2.pl
```

con la primera línea obtenemos la creación de directorios /home automática, y con la segunda conseguimos que en el momento del login se ejecute el script crearCuotas2.pl que es un programa escrito en perl que prepara las cuotas. El contenido del programa es el siguiente:

```
use DBI;
use Quota;
use Linux::usermod;

my $hlimit = 83886080;
my $slimit = 62914560;
my $dev = Quota::getqcfg('/');
my $user;
my $nombre;
my $uid;
my @usuarios;
#inicio sesion en la base de datos
my $dbh = DBI->connect('DBI:MariaDB:database=global;host=localhost',
                      'gestor','Yg0yubme7jyHzCOF',
                      { RaiseError => 1, PrintError => 0 });
$sth = $dbh->prepare("select uid, nombre from usuarios where quota=0") or
die;
$sth->execute();

while(@row = $sth->fetchrow_array){
    $uid = $row[0];
    $nombre = $row[1];
    if($uid){
        Quota::setqlim($dev,$uid,$slimit,$hlimit,0,0,0,0);
        print "las cuotas se han instalado correctamente para
$nombre\n";
        push(@usuarios,$uid);
    }
}

foreach my $i (@usuarios){
    $sth = $dbh->prepare("update usuarios set quota=1 where uid=?") or
die;
```



```
$sth->execute($i);  
}  
$dbh->disconnect;
```

El programa se conecta a la base de datos mediante el módulo DBI y el usuario gestor previamente configurado y busca usuarios que tengan el campo cuota a falso (de esta manera aplicamos cuotas de golpe a otros usuarios también cubriendo errores y solo se aplica a los usuarios del sistema en la base de datos, es decir, “nativos de nuestra práctica”, no a usuarios del sistema o creados con otro propósito como admin por ejemplo.

Mediante el módulo quota establecemos un límite duro de 80 megas y uno blando de 60 megas. Por último actualizamos el campo cuota de la base de datos para no volver a aplicar las cuotas a este usuario cuando se registre el siguiente. Se podría implementar más control de errores y eliminar código superfluo, pero como en el resto de scripts de la práctica, eso llevaría demasiado tiempo y nunca se llegaría a un resultado verdaderamente satisfactorio, por lo que nos conformamos con el resultado.

Fichero /etc/skel

Para que los usuarios tengan ficheros en su directorio nada mas este sea creado existe el directorio /etc/skel que replica sus contenidos en los directorios de los usuarios. En este directorio hemos incluido un fichero condiciones.txt, que es un fichero de texto que detalla las normas de uso del servidor

```
Bienvenido a Pecera.  
Esperamos que tu uso de la aplicación sea satisfactorio.  
No obstante nos vemos obligados a recordar las siguientes normas.  
    -No intentar atacar.  
    -respetar los datos de los demás.  
Pecera dev esta montado usado Debian 11 como base
```

y un directorio vacío llamado public_html. Este directorio se usa en combinación con el módulo de apache userdir y sirve para que todos los usuarios puedan subir allí sus páginas web, a las que acceden en el navegador con ip~/nombreUsuario. Este método se incluye como opción b para aquellos usuarios que no quieran utilizar wordpress como servicio de blogs

Base de datos.

Es necesario un servicio de base de datos para otros apartados de la practica (openldap, wordpress etc) y para crear nuestra propia base de datos de usuarios. Se ha escogido el servicio mariadb. Lo instalamos usando apt

```
apt install mariadb-server
```

para la mayoría de gestiones usaremos el usuario por defecto root usando el comando

```
mysql -u root
```

creamos también otro usuario gestor con contraseña, que será el que usaremos en la mayoría de scripts y en otros servicios

```
create user 'gestor' @'localhost' identified by 'Yg0yubme7jyHzCOF';
```

```
grant all privileges on * . * to 'gestor'@'localhost';
```

también creamos nuestra base de datos pensada para almacenar los datos de nuestros usuarios y servir de manera auxiliar a algunos de los scripts. La base de datos contiene una única tabla. Los campos incluidos de la tabla son los siguientes

- id: clave primaria con auto_increment
- uid: uid del usuario de linux
- nombre: login del usuario.
- nombreUsr: nombre del usuario
- ape1, ape2: apellidos de los usuarios
- isProfesor: booleano que indica si los usuarios son del grupo profesores o alumnos
- email: almacena el correo electrónico del usuario
- direccion: dirección postal del usuario.
- código: cuando el usuario está registrándose almacena el código de verificación que se envía al correo electrónico. Cuando el usuario acaba el registro se pone a 0
- registrado: indica si el usuario ha acabado el proceso de registro.
- h_registro: almacena la hora de registro
- quota: indica si se le han aplicado cuotas al usuario.

El comando utilizado para crear la tabla es el siguiente.

```
CREATE TABLE usuarios{
    id BIGINT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    uid int(11) not null,
    nombre VARCHAR (30) NOT NULL,
    nombreUsr VARCHAR(30),
    ape1 VARCHAR(30), ape2 VARCHAR(30),
    isProfesor BOOL,
    email VARCHAR(60) NOT NULL
    direccion VARCHAR(60),
    codigo int(11),
    registrado bool,
    h_registro datetime,
    quota bool NOT NULL
}
```

Apache2

Para poder disponer de una interfaz principal a través de la cual los usuarios se administran y acceden a sus funciones principales hemos decidido utilizar apache como servidor de páginas web.

Para instalar apache empleamos el siguiente comando

```
apt install apache2
```

La instalación de apache por defecto establece el directorio /var/www/html como root del apache, así que en este directorio es donde tenemos que meter todo aquello que queramos que sea accesible desde el exterior, como nuestras páginas web o nuestro enlace a /home/apuntes. No es necesario ya que hemos implementado el servicio wordpress, pero hemos habilitado el módulo userdir a través del comando `a2enmod userdir` de modo que cada usuario disponga en su directorio personal de una carpeta donde subir sus propios ficheros html a modo de página web. Para acceder a ellos escribimos `ip~usuario`.

Módulo CGI

La interacción con el servidor se realiza a través de páginas web. Para que las páginas web puedan ejecutar scripts del lado del servidor se utiliza un módulo llamado cgi. Este módulo se puede utilizar con muchos lenguajes pero en esta práctica nos ceñiremos al enunciado y utilizaremos perl, que con su variedad de módulos (instalables por medio del comando `cpanm`) permite realizar tareas de administración de manera mucho más sencilla que si tuviéramos que programarlo todo desde 0.

Para habilitar el uso de cgi utilizamos el comando

```
a2enmod cgi
```

Conexiones web seguras

Primero hemos de habilitar el módulo ssl de apache mediante el comando `a2enmod ssl`. Tras esto hemos de generar un certificado y una clave que guardaremos en el directorio `/etc/apache2/ssl`. Para ello utilizamos el comando

```
openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout  
/etc/apache2/ssl/apache.key  
-out /etc/apache2/ssl/apache.crt
```

Rellenamos todos los campos de acuerdo a nuestras preferencias y en el campo common name, ya que no tenemos dominio, escribimos nuestra ip.

Por ultimo, en el fichero `/etc/apache2/sites-available/default-ssl.conf` añadimos las lineas necesarias para que el servidor reconozca la ubicacion de los certificados

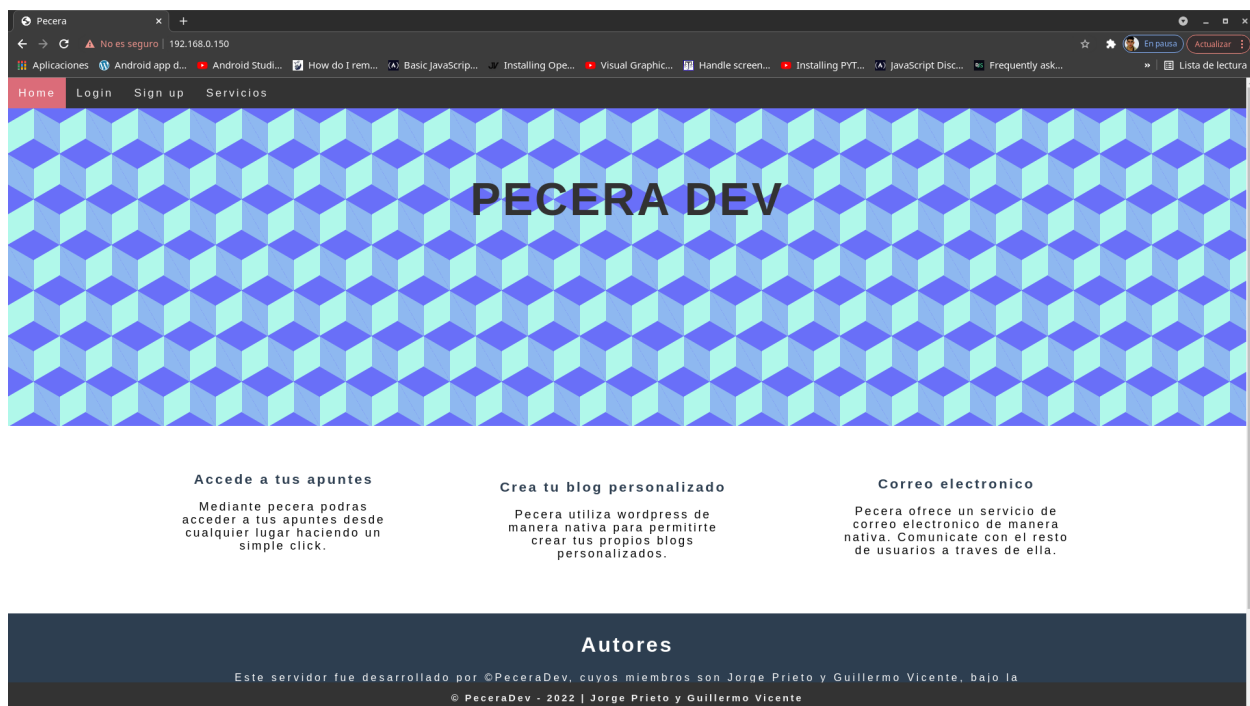
```
SSLCertificateFile      /etc/apache2/ssl/apache.crt  
SSLCertificateKeyFile  /etc/apache2/ssl/apache.key
```

Interfaz principal.

Una vez ya hemos preparado la infraestructura básica del sistema, podemos empezar a implementar la interfaz principal. Esta se compone de dos partes principales. ficheros.html accesibles a través de un navegador web que contienen formularios y scripts cgi que recogen datos de esos formularios y realizan el tratamiento oportuno. Para el correcto funcionamiento de estos scripts es necesaria la instalación de módulos auxiliares como Usermod u otros. Hay varios módulos cuya instalación ha sido más peliaguda y ha tenido que realizarse a través de apt ya que cpan no funcionaba correctamente, pero la inmensa mayoría de módulos han sido instalados utilizando el comando

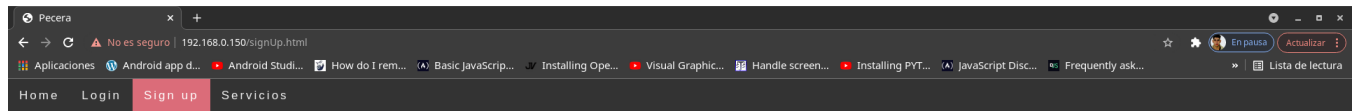
```
cpanm install nombreModulo
```

landing page (index.html)



La página principal o landing page simplemente ofrece información de los servicios que ofrece la empresa ficticia, indica los contactos de los autores e incluye una barra superior a través de la cual podemos desplazarnos a los diferentes apartados.

Sign Up



© PeceraDev - 2022 | Jorge Prieto y Guillermo Vicente

La página de registro se compone de un formulario html con los campos login, nombre, apellidos, contraseña, correo electrónico y dirección postal. Hemos añadido también un campo para especificar el rol del usuario (profesor o alumno)

El archivo correspondiente es signUp.html y en el la etiqueta forms contiene un campo action que indica que cuando se pulse el botón sign in se ejecutara el script signUp.cgi.

Para que este script se pueda ejecutar correctamente ha de estar ubicado en /usr/lib/cgi-bin y su propietario ha de ser el usuario del apache www-data. También ha de tener permisos de ejecucion asi que tecleamos los siguiente comandos

```
chown www-data /usr/lib/cgi-bin/signUp.cgi
chmod 700 /usr/lib/cgi-bin/signUp.cgi
```

además el script ha de cumplir 2 requisitos más, su cabecera ha de contener la línea #!/usr/bin/perl y ha de implementar el módulo CGI (entre otros) por lo que previamente hay que instalarlo usando cpanm

signUp.cgi registra usuarios, que en términos de linux implica escribir en los ficheros /etc/passwd y /etc/shadow. Los scripts de cgi ejecutados desde páginas web son ejecutados por www-data por lo que este usuario ha de tener permisos sobre esos ficheros.

La manera más segura que hemos encontrado para permitir esto es añadiendo a www-data al grupo shadow mediante el siguiente comando

```
usermod -aG shadow www-data
```

además debemos dar los siguientes permisos a los ficheros passwd y shadow

```
chmod 664 /etc/passwd  
chmod 660 /etc/shadow
```

signUp.cgi recoge los parámetros del formulario y los mete en variables, posteriormente verifica que los campos no estén vacíos y que el usuario introducido no exista en el sistema o sea potencialmente peligroso (admin, root, etc). Tras esto inicia sesión en la base de datos y mete los campos introducidos, genera un código de verificación y pone el campo registrado a 0. Usando el modulo usermod crea un usuario con los datos proporcionados escribiendo en los ficheros /etc/passwd y /etc/shadow indicando en el campo del intérprete de comandos nologin para que no se puedan loguear, pues el proceso de registro no ha terminado.

Esto está hecho así ya que en el enunciado se indica que el usuario no se ha de registrar del todo hasta que le llegue un correo con el código de verificación generado y el usuario lo introduce con éxito. Podríamos haber recogido la contraseña lo último tras el código de verificación o haberla almacenado en la base de datos pero la primera solución nos pareció poco elegante (teniendo en cuenta como funcionan este tipo de páginas) y la segunda poco segura ya que el método más seguro a nuestro alcance de almacenar la contraseña es el encriptado de shadow, así que optamos por esta solución.

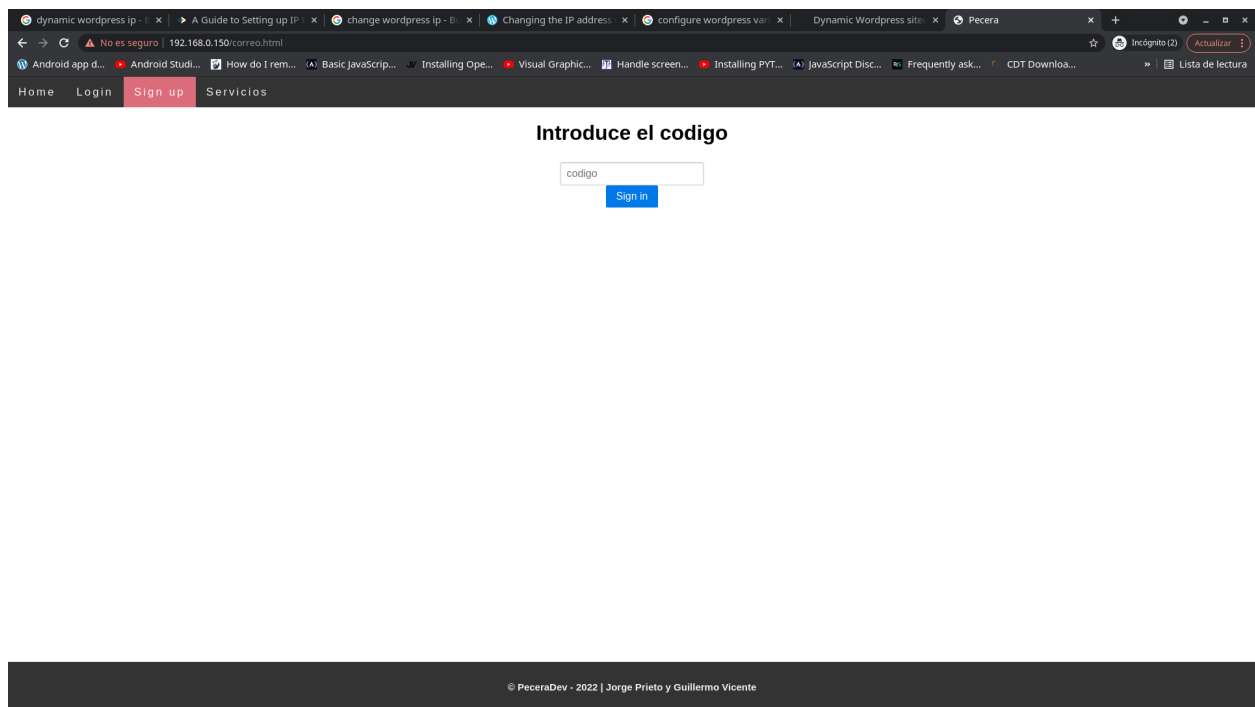
Por último se genera una cookie con algunos de los datos del usuario para utilizar en el próximo script, codigo.cgi y se manda un correo al usuario por medio del módulo Email::Send::SMTP::Gmail; y un correo de gmail llamado pecera.correo@gmail.com preparado para usar en este contexto. Para poder usar este correo de esta manera hay que activar la verificación en dos pasos y configurar una opción llamada contraseñas de aplicaciones, ya que en el plazo de realización de esta práctica se deshabilitó la opción de google que hacía este funcionamiento posible en el pasado (aplicaciones menos seguras). signUp.cgi también realiza cierto control de errores y en caso de que el programa falle en algún paso borra la entrada de /etc/passwd creada y la línea de la base de datos introducida. Después redirige a una página de error. Una vez hecho todo esto, el script redirige a correo.html.

Control de usuarios no registrados

Existe la posibilidad de que un usuario se registre pero nunca introduzca el código que se ha mandado por correo. En ese caso quedaría registrado en la base de datos y en `/etc/passwd` pero no registrado del todo ya que tendría `nologin` y aparecerá marcado como no registrado en la base de datos. Esto ocurre por nuestra insistencia en hacer el registro de usuarios de una determinada manera ya que podría ser evitable introduciendo la contraseña solo tras haber enviado el correo, pero como ya hemos explicado, este modo de funcionamiento no nos convencía por no ser el habitual en este tipo de webs.

Para parchear esta situación, existe el script `borrarNoRegistrados.pl` que se ejecuta cada hora mediante un cronjob y que sondea que usuarios no están registrados y hace más de una hora que iniciaron el proceso de registro (para eso hemos incluido el campo `h_registro` que almacena la hora de registro) y utilizando `usermod` los borra de `passwd` y de la base de datos. No es la solución más elegante pero tampoco la más chapucera.

Correo.html

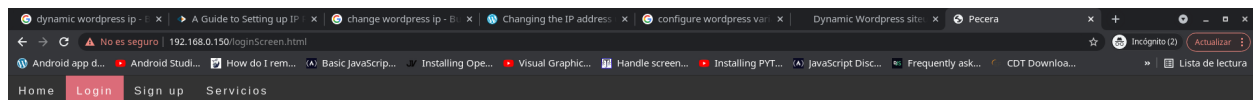


Correo.html se encarga de consultar la base de datos en busca del código que generó signUp.cgi y que se encuentra en el campo código de la base de datos. El usuario introduce un código en la casilla y el script se encarga de buscar alguna entrada de la base de datos que contenga ese código y realiza cierto control de errores.

- Si el código existe pero el usuario ya está registrado (escenario técnicamente imposible) deja al usuario intacto pero no crea al nuevo usuario
- Si el código no existe pero existe un usuario que se corresponde al introducido al apartado anterior (esto se sabe por la cookie) y este usuario no está registrado, se borra al usuario.
- Si el código existe y el usuario está con el campo registrado a falso, el script borra el código de la base de datos, pone el campo registrado a 1 y utiliza usermod para cambiar el intérprete de comandos de “nologin” a bash

Además hemos utilizado el fichero /etc/skel para que al crear los directorios de los usuarios automáticamente contengan condiciones.txt y la carpeta public_html en caso de que se quisiera subir ahí la página web.

Login



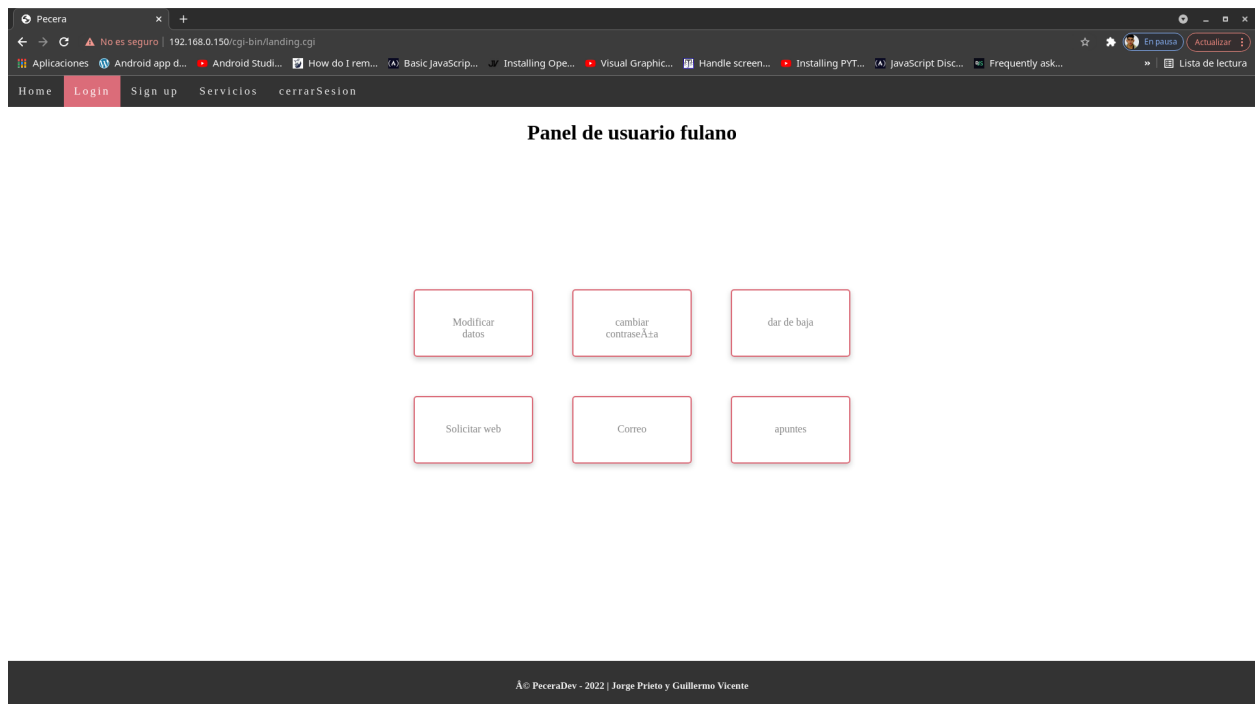
© PeceraDev - 2022 | Jorge Prieto y Guillermo Vicente

Si se pulsa el botón Login de la barra que aparece en todas las páginas se nos redirigirá al script verificarLogin.cgi, que verifica si existe una sesión activa o si no existe o ha expirado. Si existe una sesión activa, redirecciona al panel del usuario con la sesión activa y en caso contrario redirige a la página que se muestra arriba: loginScreen.html. Esta página muestra un formulario con dos campos, login y contraseña y al pulsar sign in ejecuta el script login.cgi, que toma dichos campos y utiliza el módulo Authen::PAM para verificar que sean correctos sin comprometer la contraseña y su encriptación. En caso de

que sean correctos utiliza el módulo CGI:Session para crear una sesión. Por último redirige al script landing.cgi, que toma los datos de la sesión y los utiliza para crear un panel de usuario.

Landing.cgi

Esta página no es un html. El script landing cgi imprime código html de manera dinámica. Esto lo hacemos para poder añadir algo de personalización y que muestre el nombre del usuario en el panel. landing.cgi también verifica la sesión antes de hacer nada y si ha expirado o no existe redirige a index o a una página de error



El panel de usuario está dividido en dos secciones. La primera sección (fila de arriba) está compuesta por funciones referentes al control de usuarios y la segunda a los servicios de los que estos disponen

- Modificar datos: permite al usuario cambiar todos los datos que introdujo en el login menos su login, su contraseña y su rol
- Cambiar contraseña: permite cambiar la contraseña
- dar de baja: borra al usuario de /etc/passwd y de la base de datos
- Correo: redirecciona al servicio de correo del servidor
- Solicitar web:redirecciona al servicio de wordpress del servidor
- apuntes: redirecciona al enlace simbólico que apunta a /home/apuntes donde se encuentran los apuntes subidos por los profesores.

Modificar datos

Modificar datos

nombre

1er apellido

2º apellido

correo

direccion postal

Confirmar credenciales

login

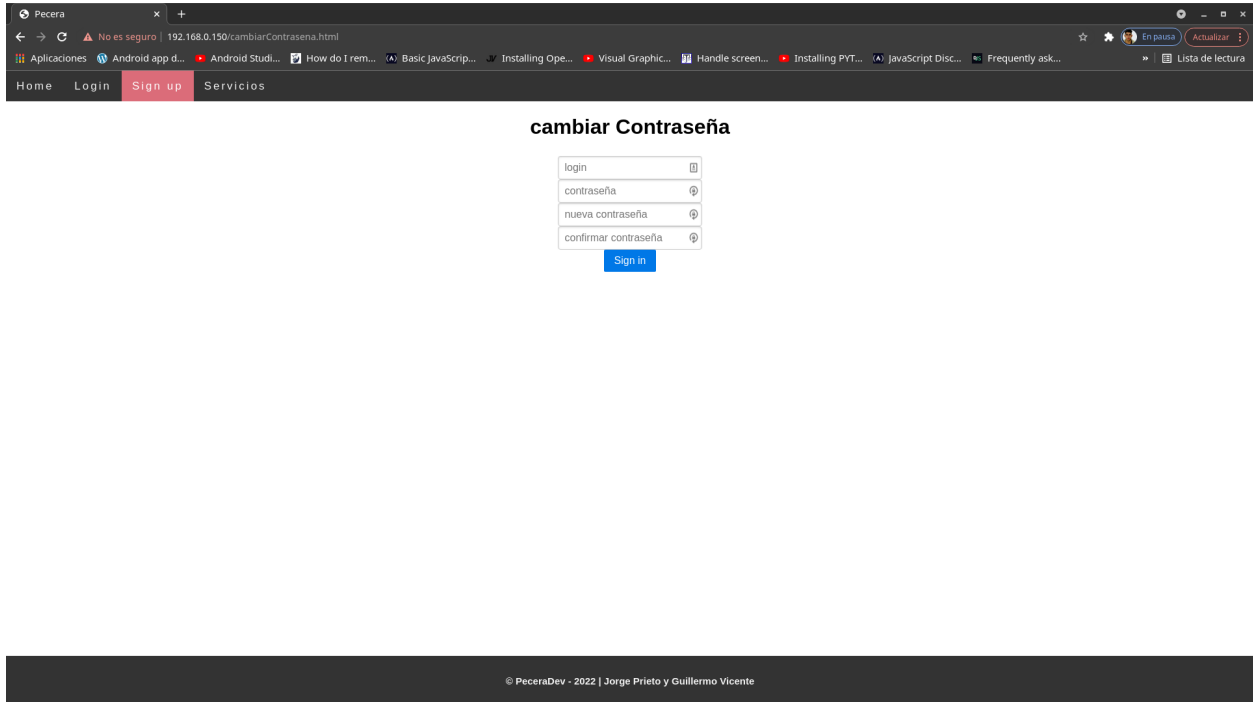
contraseña

Sign in

© PeceraDev - 2022 | Jorge Prieto y Guillermo Vicente

permite introducir valores distintos para los campos, nombre, apellidos, correo electrónico y dirección postal. Por último hay que introducir usuario y contraseña para que al pulsar el botón, se ejecute el script `modificarUsuario.cgi` el cual toma los parámetros mediante el módulo `cgi`, verifica que la sesión está activa y tras eso utiliza `Authen::PAM` para verificar que las credenciales son correctas y procede a cambiar los valores viejos por los nuevos tanto de `passwd` como de la base de datos mediante los módulos `DBI` y `usermod`.

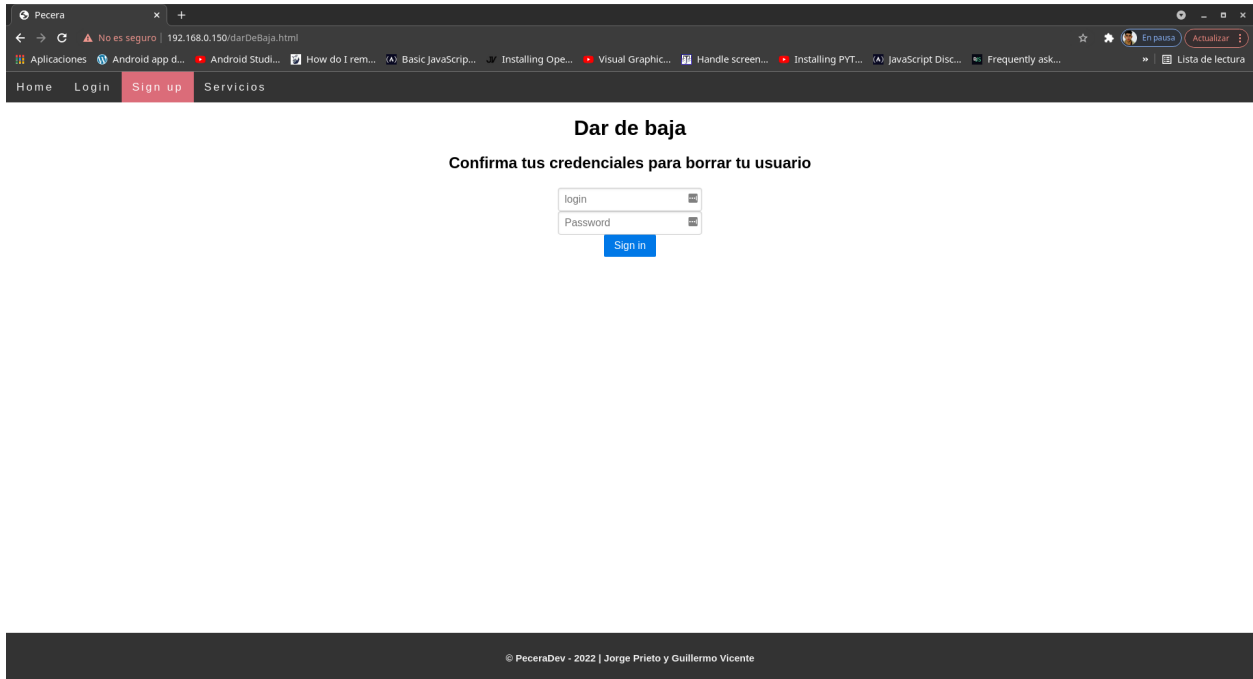
Cambiar contraseña



The screenshot shows a web browser window with the title 'Pecera' and the URL '192.168.0.150/cambiarContrasena.html'. The browser's address bar shows a warning 'No es seguro' (Not secure). The page has a navigation bar with links: Home, Login, Sign up (highlighted in red), and Servicios. The main content area is titled 'cambiar Contraseña' and contains a form with four input fields: 'login', 'contraseña', 'nueva contraseña', and 'confirmar contraseña'. Each field has a small icon to its right. Below the fields is a blue button labeled 'Sign in'. At the bottom of the page, there is a dark footer bar with the text '© PeceraDev - 2022 | Jorge Prieto y Guillermo Vicente'.

cambiarContrasena.html tiene un formulario con 4 campos, login contraseña, nueva contraseña y repetir nueva contraseña. Al pulsar el boton ejecuta el script cambiar contrasena.cgi, que primero verifica que exista una sesion, luego verifica que el login y contraseña introducidos sean correctos usando Authen::PAM y por último, tras verificar que las dos “nuevas contraseñas” son iguales usa usermod para establecer la nueva contraseña. Como no almacenamos la contraseña en la base de datos no hace falta cambiar esto.

dar de baja



La opción dar de baja redirige a la página darDeBaja.html, en la cual se pide un login y contraseña de confirmación, tras lo cual se ejecuta el script borrarUsr.cgi que realiza los trámites ya habituales (verificar la sesión y autenticar al usuario con CGI::Session y Authen:PAM respectivamente) y por último se procede a borrar la entrada correspondiente al usuario de la base de datos y a utilizar usermod para borrar al usuario de /etc/passwd y /etc/shadow. Para finalizar se redirige a la página principal.

Cerrar sesion.

La opción de cerrar sesion se encuentra en la barra superior del panel del usuario y pulsarla ejecuta el script cerrarSesion.cgi que es un script muy simple que sencillamente carga la CGI::Session la borra, hace un flush y redirige a index.html

Apuntes

La opción de apuntes redirige al enlace simbólico /var/www/html/apuntes que apunta a /home/apuntes, directorio que como ya se ha explicado anteriormente tiene permisos de lectura y ejecución para todos (si no no se podría ver desde el navegador) pero de escritura solamente para los usuarios del grupo profesores de modo que solo estos puedan subir y borrar apuntes pero todo el mundo pueda leerlos.

Correo

Para configurar el correo electrónico de los usuarios primero hay que instalar el servidor de correos postfix mediante el comando

```
apt install postfix
```

y tras instalar usamos dpkg-reconfigure postfix para introducir correctamente todos los datos según nuestras preferencias. Introducimos como nombre de dominio pecera.local

Tras instalar postfix tenemos que cambiar el fichero de configuración /etc/postfix/main.cf para hacer que los buzones tengan tamaño 3mb pero para que esto funcione también hemos de hacer que el tamaño máximo de los mensajes sea menor que el total del buzón.

Añadimos las siguientes líneas al fichero para conseguirlo y reiniciamos el servicio

```
mailbox_size_limit = 3145728  
message_size_limit = 31450
```

tras esto instalamos dovecot

```
apt install dovecot-imapd dovecot-pop3d
```

y reiniciamos el servicio con service dovecot restart

por ultimo instalamos squirrelmail mediante los siguientes comandos

```
wget  
https://sourceforge.net/projects/squirrelmail/files/stables/1.4.22/squirrel  
mail-webmail-1.4.22.xip  
unzip squirrelmail-webmail-1.4.22.zip  
mv squirrelmail-webmail-1.4.22 /var/www/html  
chown -R www-data:www-data /var/www/html/squirrelmail-webmail-1.4.22/  
chmod 755 -R /var/www/html/squirrelmail-webmail-1.4.22/  
mv /var/www/html/squirrelmail-webmail-1.4.22 /var/www/html/squirrelmail
```

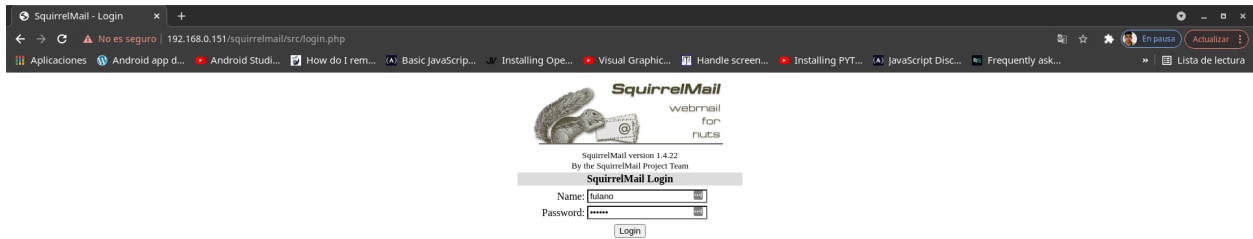
y tras esto usamos el comando

```
perl /var/www/html/squirrelmail/config/conf.pl
```

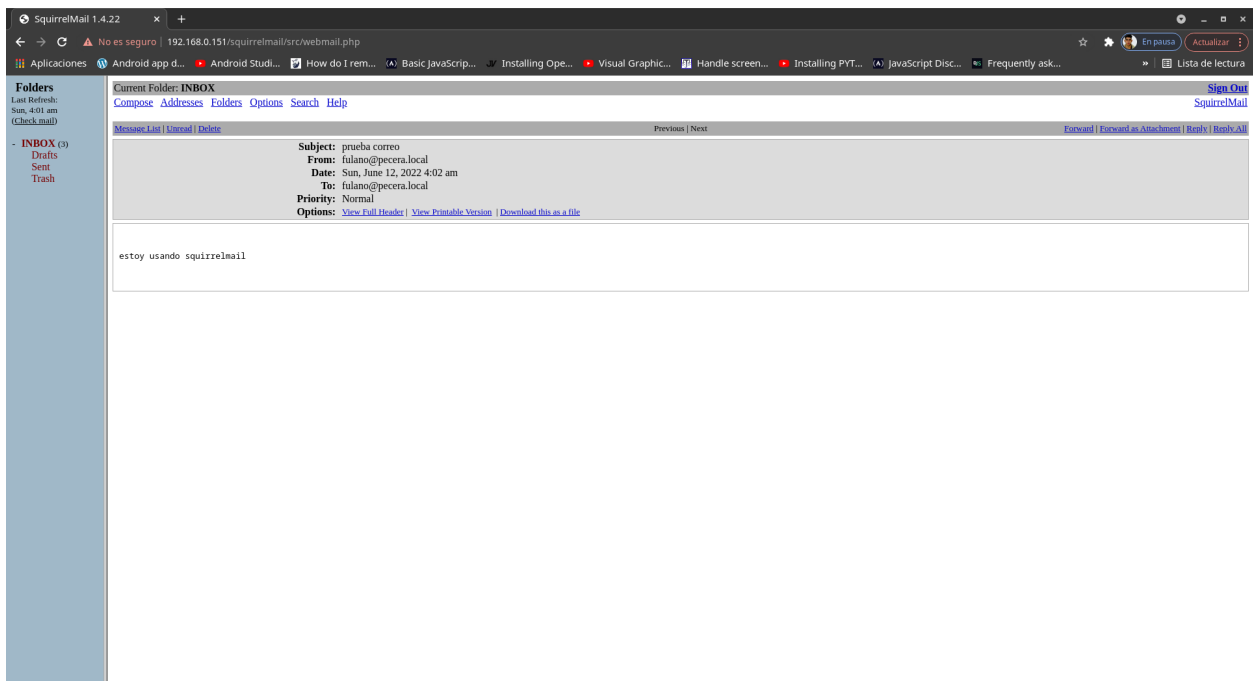
para configurar squirrelmail. En ese menu tenemos que cambiar las opciones Data Directory a /var/www/html/squirrelmail/data , Attachment to Directory a /var/www/html/squirrelmail/attach y allow server-side sorting a true

Tras esto podemos acceder a la web mediante el script landing.cgi que en la opcion

correo nos redirige a /squirrelmail



aquí nos logueamos con nuestras credenciales y nos enviamos un correo a nosotros mismos para comprobar el funcionamiento del webmail



Solicitar web

Para el hosting de webs hemos decidido utilizar wordpress

forgotPass.cgi

Se ejecuta si en la pantalla de login clickeamos en el enlace que dice “he olvidado mi contraseña”. Se redirige entonces a forgot.html que pide el login de usuario. Tras pulsar el botón se ejecuta el script forgotPass.cgi. Este script genera un número de 8 dígitos que envía por correo utilizando Email::Send::SMTP al correo asociado al login que se ha introducido. Mediante una consulta a la base de datos se obtiene el correo de ese login y se le manda el código por mail. Usando usermod se establece ese código como la nueva contraseña.

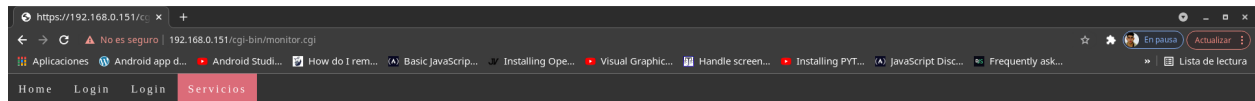
Si el usuario quisiera cambiar su contraseña a una “normal” solo tendría que loguearse con la que le ha llegado por correo y escoger la opción de cambiar contraseña.

Monitor.cgi

Este script es el encargado de mostrar el estado de los servicios principales del servidor (apache, postfix, quotas, sql ssh). Para ello usamos un script auxiliar llamado monitor servicios.sh para el cual hemos programado un cronjob que lo ejecuta cada 5 minutos y que genera directorios que contienen un 1 o un 0 en la ubicación /var/www/html/estado/nombreServicio en función del estado de dichos servicios. El estado de los servicios se obtiene usando el comando

```
systemctl is-active nombreServicio
```

El script monitor.cgi lee estos directorios y si al leerlos contienen un 1 (el servicio esta activo) genera una tarjeta de color verde con la leyenda “activo”. En caso contrario la tarjeta será de color rojo y contendrá la palabra inactivo. Para obtener esta información tenemos que pulsar la pestaña “Servicios” en la barra de navegación.



Estados

Web
Activo
SSH
Activo
SMTP
Activo
IMAP
Activo
BASE DE DATOS
Activo
CUOTAS
Activo

Pantallas de error.

Por último se han incluido varios html idénticos a los expuestos anteriormente con la diferencia de que cuentan con un mensaje informativo de error correspondiente a las posibles situaciones de error que se pueden dar.

Cuotas

Este script se ejecuta cada vez que un usuario hace login gracias al fichero de configuración de openldap /etc/pam.d/common-account cuyo funcionamiento hemos explicado en el apartado de openldap. El script implementa el módulo quota y se encarga de aplicar la cuota indicada en el enunciado a aquellos usuarios que no las tengan instaladas según la información de nuestra base de datos.

Backups.

De las copias de seguridad se encarga el script backups.sh ubicado en /var/pecera. y cuyo contenido es

```
rsync -ravzh /home/ /backups/home
rsync -ravzh /root/ /backups/root
```

```
rsync -ravzh /var/lib/mysql/ /backups/mysql/  
rsync -ravzh /var/www/html /backups/html  
rsync -ravzh /usr/lib/cgi-bin
```

Se ha programado un cronjob que ejecuta este script cada hora. y la información que hemos creído conveniente que guarde es:

- directorios home
- directorio home del root
- directorio de mysql con las bases de datos
- paginas web y scripts de cgi.

Monitorización

Para la monitorización son necesarios 2 scripts. El primero, generarDatos.sh genera ficheros con información referente al estado de la memoria y al uso de procesos.

```
free -m > /var/pecera/datos/free.txt  
sa -c > /var/pecera/datos/comandos.txt  
top -n 1 -b> /var/pecera/datos/carga.txt
```

Este fichero tiene un cron job programado que ejecuta este script cada día a las 23:59. Esto es así ya que hay otro cronjob programado para las 00 que ejecuta el script /var/pecera/registroAdmin.pl

Este segundo script crea un fichero llamado informe.txt ubicado en /tmp que mandará por correo electrónico utilizando el módulo Email::Send::SMTP::Gmail de perl y la cuenta de correo que creamos anteriormente a la cuenta del administrador, en este caso mi cuenta personal de gmail.

Para generar el informe, el script lee los ficheros generados además de utilizar el comando du para leer el uso de disco que hace cada usuario registrado en la base de datos.

Wordpress

Para que los usuarios puedan crear sus propios blogs, hemos decidido instalar wordpress.

Para instalar wordpress hay que seguir los siguientes pasos

1(Descargar el paquete comprimido de wordpress

```
wget https://es.wordpress.org/latest-es_ES.tar.gz
```

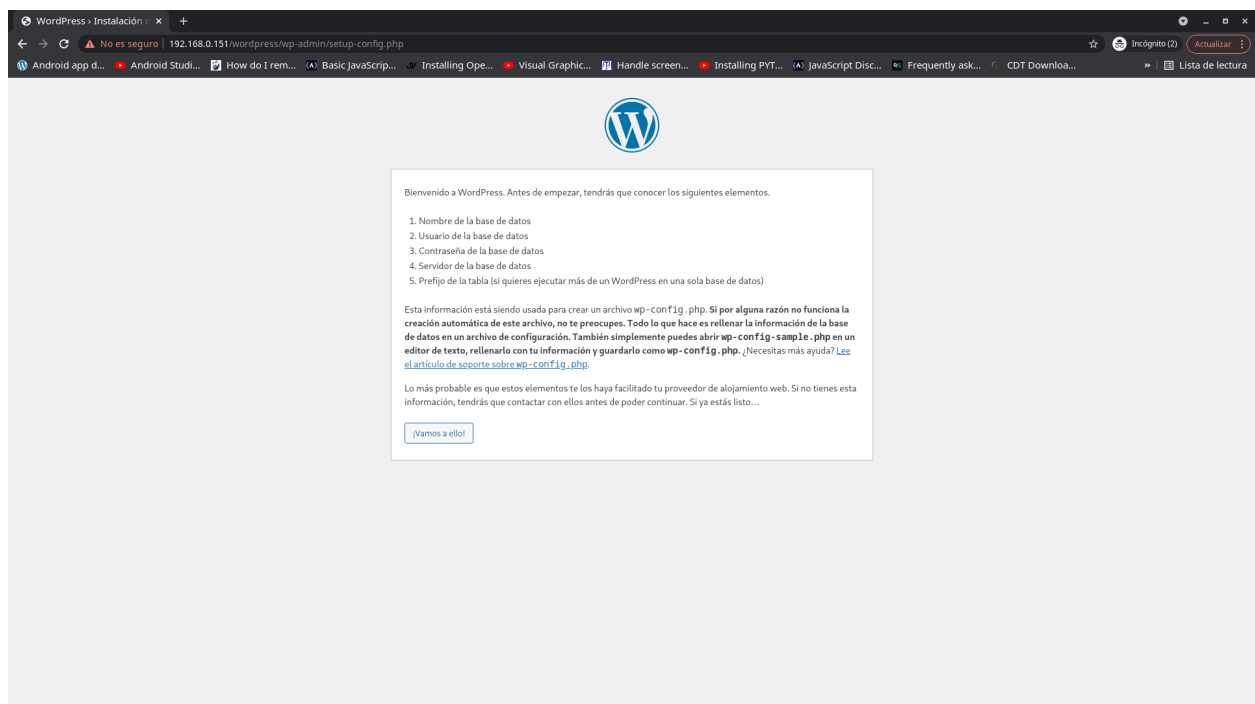
2) Descomprimir el paquete en /var/www/html

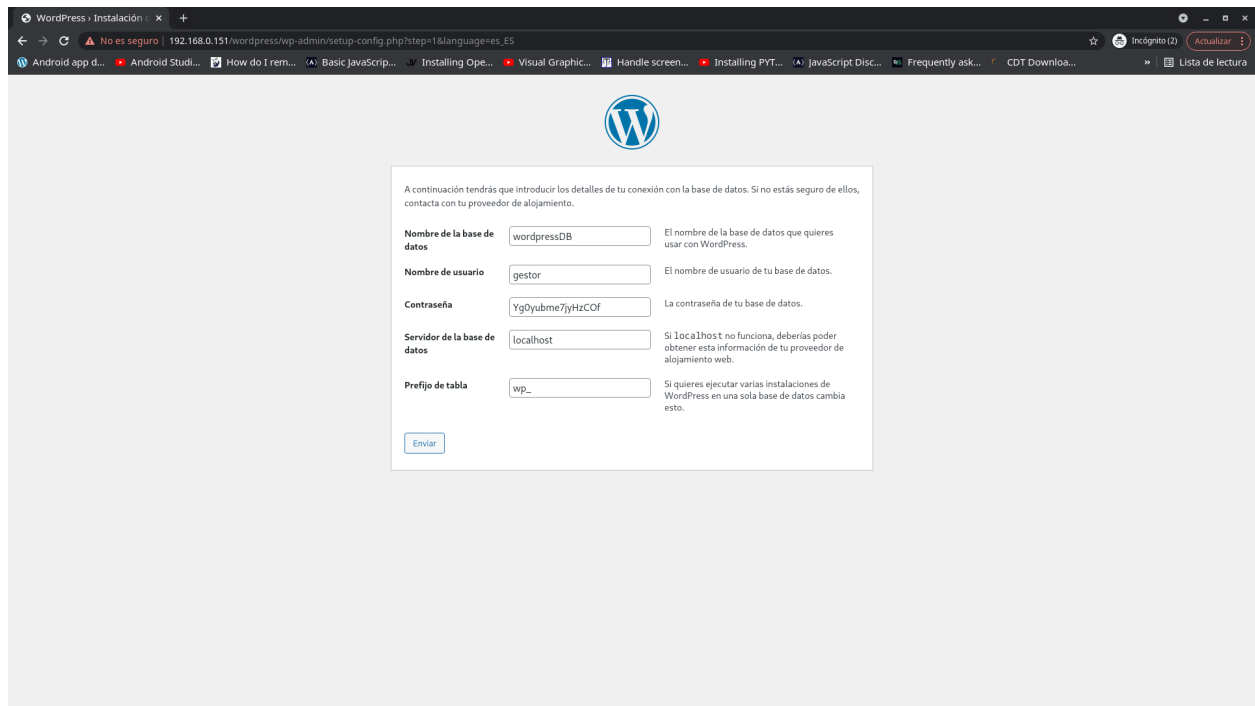
```
sudo tar xf latest-es_ES.tar.gz -C /var/www/html
```

3) darle los permisos de la carpeta a www-data

```
chown www-data:www-data /var/www/html/wordpress/ -R
```

Una vez hecho esto podemos acceder al panel de configuración inicial escribiendo en nuestro navegador la dirección ip del servidor/wordpress y nos redirigirá automáticamente

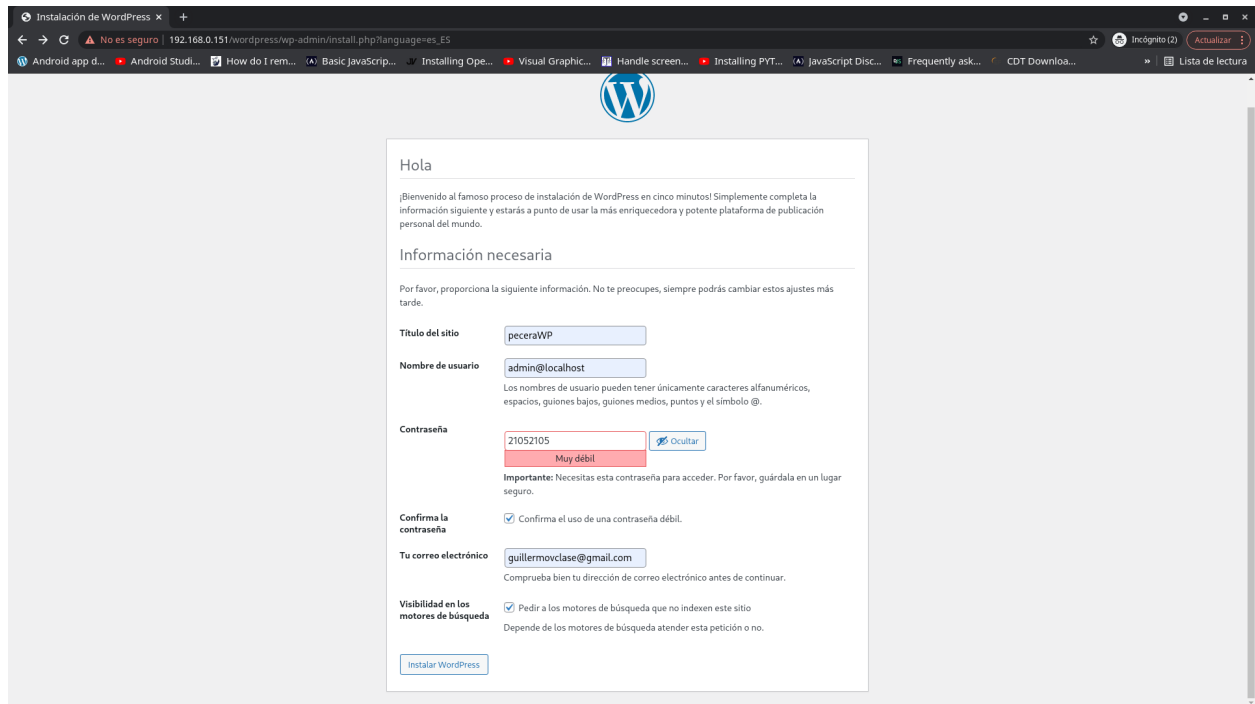




En esta pantalla introducimos la información pertinente y utilizamos el usuario de la base de datos gestor al que ya le hemos dado privilegios anteriormente. También tenemos que crear una base de datos vacía que ya se encargará de rellenar el configurador de WordPress.

```
create database wordpressDB;
```

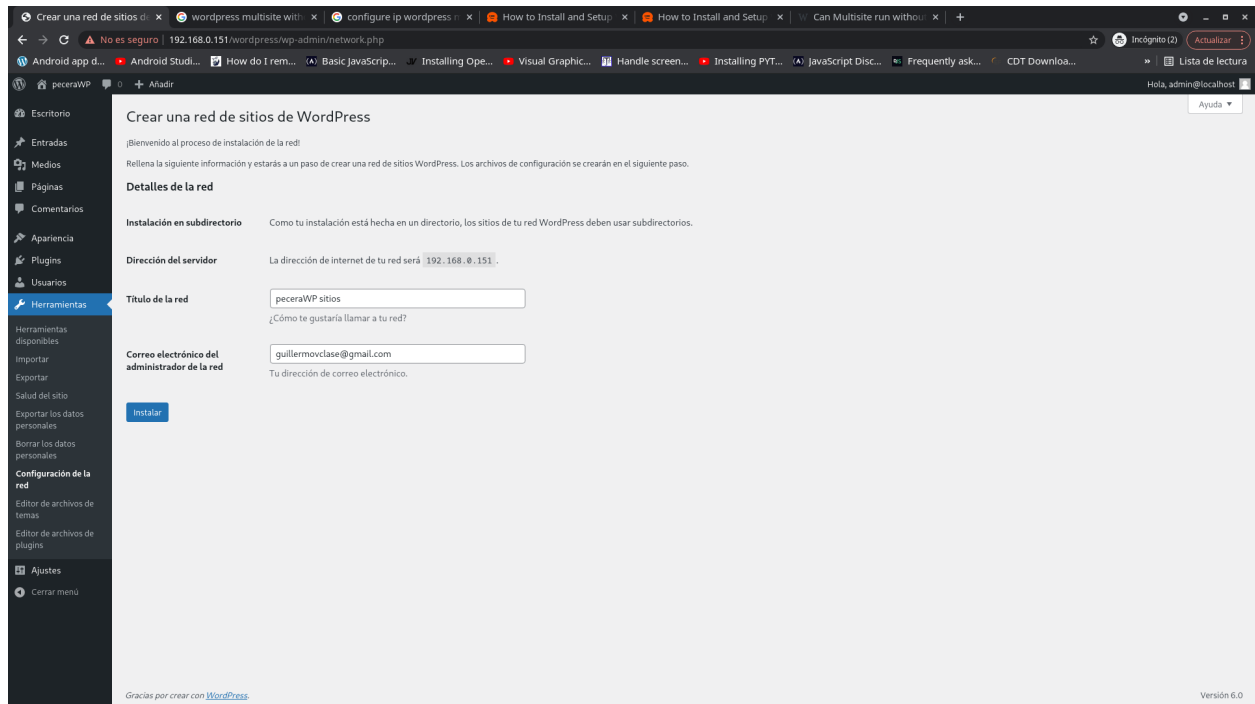
Tras crear y vincular la base de datos se nos volverá a redireccionar a otra página de configuración, esta vez del propio sitio de WordPress.



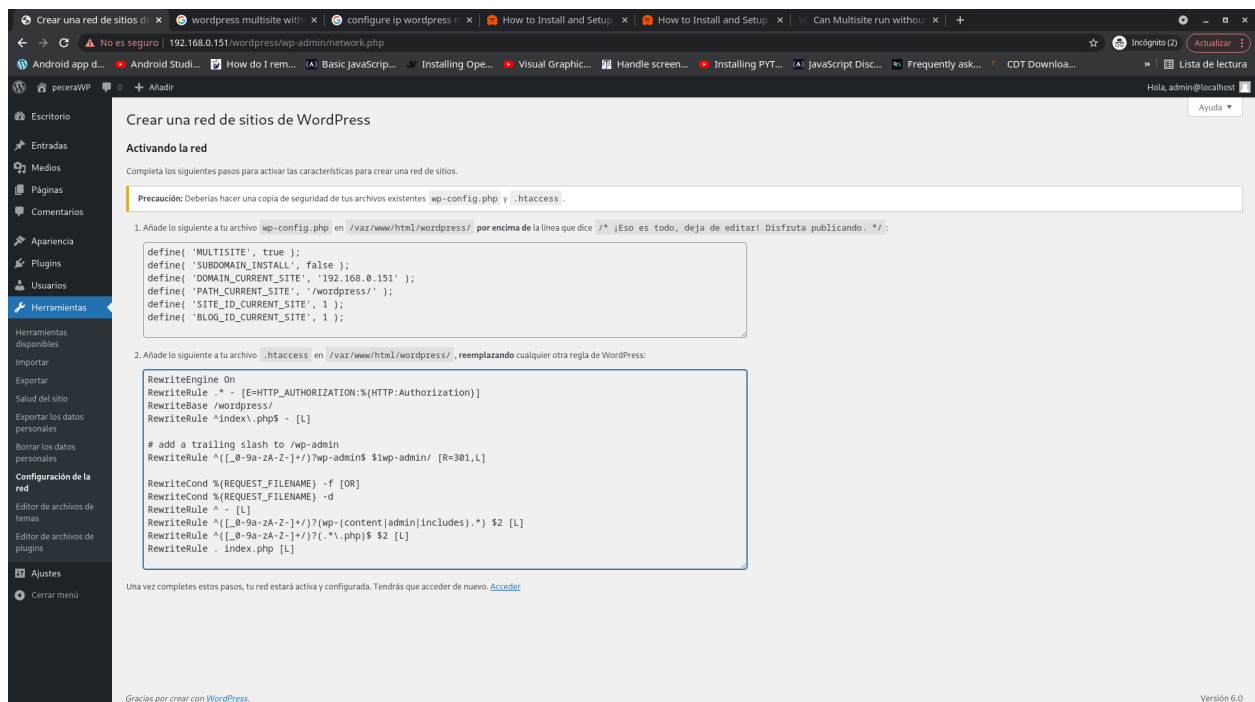
Con esto ya tenemos wordpress instalado, pero los usuarios aún no pueden crear sus propios blogs. Para eso tenemos que habilitar una función de wordpress llamada multisite.

Para poder acceder a la configuración de multi site desde el menú visual de wordpress hace falta añadir una linea al fichero /var/www/html/wordpress/wp-config.php

```
define('WP_ALLOW_MULTISITE',true);
```

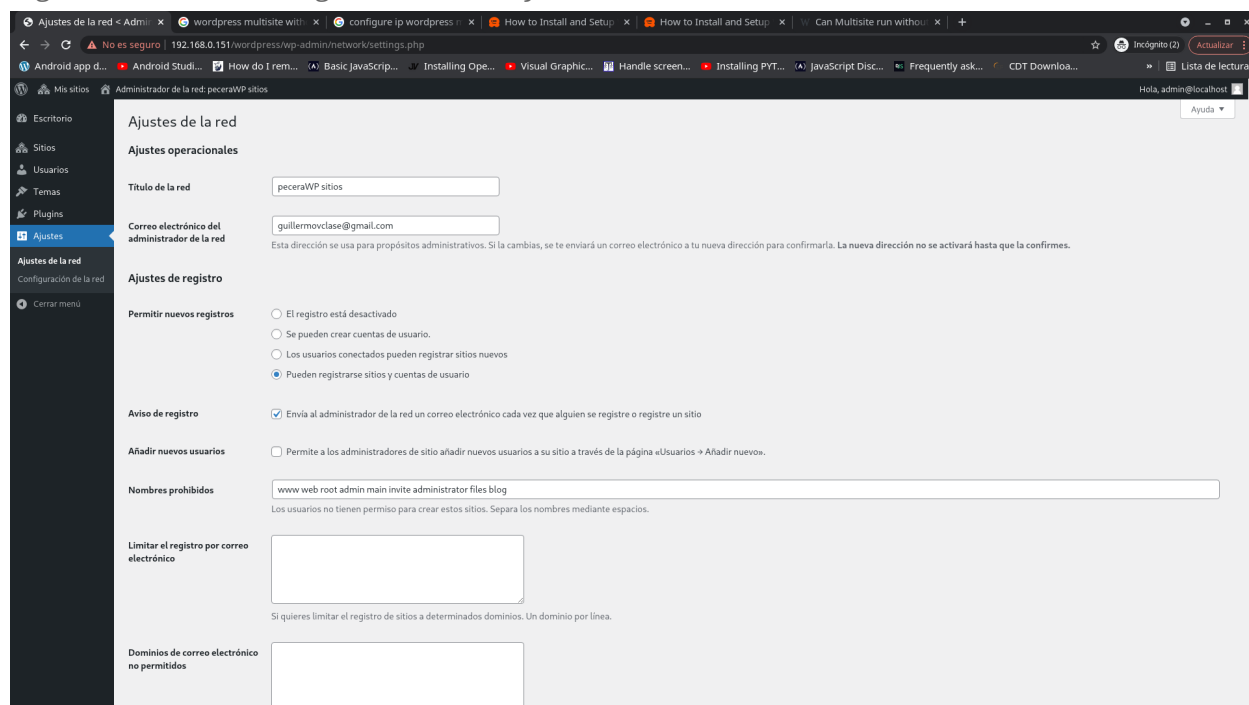


después nos dirigimos a la pagina de configuracion al apartado de configuracion de la red, tras pulsar instalar wordpress nos indica que tenemos que añadir las siguientes lineas a los ficheros wp-config.php y .htaccess así que las añadimos



Una vez completados todos estos pasos nos tenemos que dirigir a mis sitios > administrador de la red en el panel de configuración de wordpress y allí a ajustes de la red. Por último en esta pantalla de configuración cambiamos la opción permitir nuevos

registros a “Pueden registrarse sitios y cuentas de usuario”



Y con esto ya estaría wordpress configurado. landing.cgi, es decir, el panel del usuario tiene un botón que redirecciona al php de solicitud de sitio. Como método auxiliar, hemos activado el modulo userdir (como se explica en el apartado de apache) y creado un directorio /public_html bajo /etc/skel de modo que si la manera en la que los usuarios quieren subir sus blogs es escribiendolos en html y subiendolos a su carpeta public_html a través de sftp puedan.

Otras funcionalidades.

Mensaje de login al root.

Es interesante que se mande un mensaje al root cada vez que se realiza un login con su usuario para detectar posibles ataques. Con esto en mente hemos escrito el script correoAdmin.pl que de nuevo utiliza Email::Send::SMTP::Gmail para mandar un correo desde correo.pecera@gmail.com a mi cuenta personal de gmail avisando de que se ha producido un login y la hora a la que se ha producido.

Para que este script se ejecute cada vez que el root o el admin se logean hemos añadido la línea

```
/usr/bin/perl /var/pecera/correoAdmin.pl
```

al final del fichero .profile del root y del admin. De este modo se ejecutará cada vez que se produzca un login.

Crontab

Se ha ido especificando en cada apartado los scripts auxiliares que tenemos en crontab, pero dado que son varios hemos considerado oportuno incluir esta apartado para saber y tener a mano los procesos que se pueden estar ejecutando y en que momento

El contenido del crontab lo leemos ejecutando `crontab -e` y en nuestro caso es el siguiente:

```
# m h dom mon dow  command
0 * * * * /usr/bin/perl /perlScripts/borrarNoRegistrados.pl
*/20 * * * * /scripts/monitorServicios.sh
#estos dos se generan seguidos
59 23 * * * /var/pecera/generarDatos.sh
0 0 * * * /usr/bin/perl /var/pecera/registroAdmin.pl
0 0 * * * /var/pecera/backups.sh
```

- borrar no registrados: se encarga de borrar de passwd y la base de datos aquellas entradas de usuarios que no han completado el proceso de registro una vez iniciado
- monitor servicios: utiliza `systemctl` para generar ficheros con el estado de dichos servicios. Estos ficheros serán leídos por `monitor.cgi` que genera una página web mostrando su estado.
- `generarDatos.sh`: se ejecuta a las 23:59 cada día y genera ficheros de texto con las salidas de varios comando (`free`, `sa`, `du`) que posteriormente serán leídos por `registroAdmin`
- `registroAdmin` lee los ficheros generados por `generarDatos` y los mete en un fichero de texto que envía al admin por correo.
- `backups.sh`: realiza backups del sistema cada día.

Conclusiones

A pesar de que el trabajo contiene todos los requisitos especificados en el enunciado y alguno de los opcionales, no estamos del todo satisfechos con nuestro trabajo ya que el servidor tiene algunas asperezas que con algo más de tiempo podrían haber sido resueltas o más medios (nombres de dominio o cosas similares).

También hay decisiones de seguridad que hemos tomado que pese a ser correctas entorpecen el avance en otros frentes. Por ejemplo, la gestión de directorios de usuario habría sido mucho más sencilla utilizando `system` o `suexec` permitiendo ejecutar los scripts de cgi con otro usuario distinto de `www-data` (principalmente `root`), pero consideramos que este tipo de soluciones son potencialmente peligrosas y hemos preferido tener una funcionalidad mas reducida o algo mas incompleta en ciertos aspectos que un sistema inseguro.

Otro ejemplo de esto mismo es el registro de usuarios, que técnicamente solo se puede dar una vez el solicitante ha introducido un código que le ha llegado por correo. Podríamos haber hecho esto de otras mil maneras, como por ejemplo guardando los datos del usuario en la base de datos y tomándolos después de que se introdujese el código para (solo entonces) registrarlo, pero todas estas soluciones pese a ser superiores en funcionalidad o ceñirse más al enunciado, considerábamos que eran bastante menos seguras.

Hay algunos otros detalles que podrían haber mejorado nuestra práctica, como un mayor control de errores, de sesiones, etc etc, pero pese a no estar presentes y pese a las pequeñas faltas de funcionalidad que reconocemos, también consideramos que es una sólida, trabajada, completa y cuya implementación no está hecha al azar sino que responde a una meditación previa y minuciosa de las implicaciones de cada decisión a tomar, y esperamos que se tenga en cuenta.

En cuanto a los fuentes incluidos en la entrega, los hemos entregado tal y como nosotros los tenemos clasificados por carpetas, pero en el servidor todas las páginas se encuentran en `/var/www/html`, todos los cgi se encuentran en `/var/lib/cgi-bin` y el resto de programas en perl y bash se encuentran en `/var/pecera`