

[Página Principal](#) / Mis cursos / [GRADUADO-A EN INGENIERÍA INFORMÁTICA \(2010\) \(296\)](#)

/ [INGENIERÍA DE SERVID \(2021\)-296 11 35 2021](#) / [Práctica 2](#) / [Examen P2 Grupo B1](#)

Comenzado el miércoles, 11 de noviembre de 2020, 15:35

Estado Finalizado

Finalizado en miércoles, 11 de noviembre de 2020, 16:09

Tiempo 34 minutos 15 segundos

empleado

Puntos 20,00/20,00

Calificación 10,00 de 10,00 (100%)

Pregunta **1**

Correcta

Puntúa 1,00 sobre 1,00

¿Qué ocurre tras ejecutar: firewall-cmd --add-port=443/tcp --permanent?

- a. El puerto 443 estará abierto tras reiniciar la máquina □
- b. El puerto 443 está abierto siempre
- c. El puerto 443 está abierto para la sesión actual
- d. El puerto 443 estará cerrado tras reiniciar la máquina

Respuesta correcta

La respuesta correcta es:

El puerto 443 estará abierto tras reiniciar la máquina

Pregunta **2**

Correcta

Puntúa 1,00 sobre 1,00

¿Cómo añadimos archivo1.txt al repositorio local usando git?

- a. Ninguna de las otras respuestas es correcta □
- b. git add archivo1.txt
- c. git add archivo1.txt --stage
- d. git add archivo.txt

Respuesta correcta

La respuesta correcta es:

Ninguna de las otras respuestas es correcta

Pregunta 3

Correcta

Puntúa 1,00 sobre 1,00

¿Qué hace el servicio fail2ban?

- a. Bloquear usuarios tras varios intentos de acceso fallidos
- b. Activa el cortafuegos para evitar el acceso a través de un puerto
- c. "Banea" clientes en caso de uso abusivo del servidor
- d. Bloquear direcciones IP tras varios intentos fallidos de acceso



Respuesta correcta

La respuesta correcta es:

Bloquear direcciones IP tras varios intentos fallidos de acceso

Pregunta 4

Correcta

Puntúa 1,00 sobre 1,00

Tras instalar el paquete de servidor de MariaDB, ¿qué debemos hacer?

- a. Editar el archivo de configuración
- b. Ejecutar el script de instalación segura
- c. Dejar la instalación por defecto
- d. Ejecutar mysql



Respuesta correcta

La respuesta correcta es:

Ejecutar el script de instalación segura

Pregunta 5

Correcta

Puntúa 1,00 sobre 1,00

Al instalar el servidor Apache en CentOS

- a. hemos de reiniciar el servicio
- b. ya podemos acceder a la web por defecto
- c. hemos de habilitar e iniciar el servicio
- d. hemos de habilitar el servicio



Respuesta correcta

La respuesta correcta es:

hemos de habilitar e iniciar el servicio

Pregunta 6

Correcta

Puntúa 1,00 sobre 1,00

¿Cómo podemos saber en qué puerto está escuchando un servidor SSH en CentOS?

- a. systemctl status open-ssh
- b. cat /etc/ssh/sshd_config | grep port
- c. systemctl ssh state
- d. systemctl status sshd



Respuesta correcta

La respuesta correcta es:

systemctl status sshd

Pregunta 7

Correcta

Puntúa 1,00 sobre 1,00

¿Cómo podríamos transferir todos los archivos (incluyendo ocultos) del directorio /home/opq/dir1 a una máquina remota para el usuario agp?

- a. rsync -re "ssh -p 22022" /home/agp/dir1/. opq@IP:/home/opq/dir1
- b. rsync -re "ssh -p 22022" /home/opq/dir1/. agp@IP:/home/agp/dir1
- c. rsync -re "ssh -p 22022" /home/opq/dir1/. opq@IP:/home/agp/dir1
- d. rsync -re "ssh -p 22022" /home/opq/dir1/* agp@IP:/home/agp/dir1



Respuesta correcta

La respuesta correcta es:

rsync -re "ssh -p 22022" /home/opq/dir1/. agp@IP:/home/agp/dir1

Pregunta 8

Correcta

Puntúa 1,00 sobre 1,00

Al hacer yum/dnf install mariadb

- a. hemos instalado el servicio de MariaDB
- b. hemos activado el servicio de MySQL
- c. hemos instalado el cliente y el servicio de MariaDB
- d. hemos instalado el cliente de MariaDB



Respuesta correcta

La respuesta correcta es:

hemos instalado el cliente de MariaDB

Pregunta 9

Correcta

Puntúa 1,00 sobre 1,00

Los comandos para comprobar el estado de ufw y abrir el puerto 22022 son

- a. sudo ufw state y ufw allow 22022 --permanent
- b. sudo ufw status y ufw --add-port=22022/tcp
- c. sudo ufw state y ufw allow 22022
- d. sudo ufw status y ufw allow 22022



Respuesta correcta

La respuesta correcta es:

sudo ufw status y ufw allow 22022

Pregunta 10

Correcta

Puntúa 1,00 sobre 1,00

¿Es posible sincronizar una fuente local con un destino remoto usando rsync?

- a. Sí
- b. Sí, sólo si fuente y destino están en la misma red local
- c. No
- d. Sí, sólo si fuente y destino tienen instalado git



Respuesta correcta

La respuesta correcta es:

Sí

Pregunta 11

Correcta

Puntúa 1,00 sobre 1,00

El nombre del archivo de clave pública creado por defecto es

- a. id_rsa.pub
- b. id_key
- c. id_key.pub
- d. id_rsa



Respuesta correcta

La respuesta correcta es:

id_rsa.pub

Pregunta 12

Correcta

Puntúa 1,00 sobre 1,00

¿Qué comando debemos ejecutar para que SELinux permita la conexión del servicio SSH en el puerto 22022?

- a. semanage port -a -t ssh_port_t -u udp 22022
- b. semanage port -a -t 22022 ssh_port_t
- c. selinux port -a -t ssh_port_t -p tcp 22022
- d. semanage port -a -t ssh_port_t -p tcp 22022



Respuesta correcta

La respuesta correcta es:

semanage port -a -t ssh_port_t -p tcp 22022

Pregunta 13

Correcta

Puntúa 1,00 sobre 1,00

¿Qué archivo debemos modificar para bloquear el acceso root en SSH?

- a. /etc/ssh/sshd_config
- b. /var/www/html/sshd_config
- c. .ssh/known_hosts
- d. /etc/ssh/ssh_config



Respuesta correcta

La respuesta correcta es:

/etc/ssh/sshd_config

Pregunta 14

Correcta

Puntúa 1,00 sobre 1,00

¿Cuál es el significado del stack LAMP?

- a. Linux Arch MySQL PHP
- b. Linux Apache MySQL PHP
- c. Ubuntu Apache MariaDB Python
- d. Linux Apache2 MariaDB PHP



Respuesta correcta

La respuesta correcta es:

Linux Apache MySQL PHP

Pregunta 15

Correcta

Puntúa 1,00 sobre 1,00

¿Cómo podemos unirnos al screen con ID 1968?

- a. screen -r 1968
- b. screen -d 1968
- c. screen 1968
- d. screen



Respuesta correcta

La respuesta correcta es:

screen -r 1968

Pregunta 16

Correcta

Puntúa 1,00 sobre 1,00

El acceso root para SSH está habilitado por defecto en

- a. Ubuntu
- b. Ningún sistema operativo
- c. CentOS
- d. CentOS y Ubuntu



Respuesta correcta

La respuesta correcta es:

CentOS

[◀ Examen P1 Grupo B1](#)

[Ir a...](#)

[Examen B3 ▶](#)

Pregunta 17

Correcta

Puntúa 1,00 sobre 1,00

¿Usando git puede diferir el contenido del repositorio local con el del directorio de trabajo actual?

- a. Sí, siempre difiere
- b. No, commit se encarga de eso
- c. No, add se encarga de eso
- d. Sí, también puede diferir del stage



Respuesta correcta

La respuesta correcta es:

Sí, también puede diferir del stage

Pregunta 18

Correcta

Puntúa 1,00 sobre 1,00

¿Qué comando debe ejecutar para crear un archivo comprimido a partir de una carpeta?

- a. tar -xvf folder.tar.gz /home/folder
- b. tar cvzffolder.tar.gz /home/folder
- c. tar cvf folder.tar.gz /home/folder
- d. tar cvzf /home/folder/ folder.tar.gz



Respuesta correcta

La respuesta correcta es:

tar cvzf folder.tar.gz /home/folder

Pregunta 19

Correcta

Puntúa 1,00 sobre 1,00

¿Qué script debemos usar para crear las claves de autenticación?

- a. SSH keygen
- b. ssh-keygen
- c. ssh-key-generator
- d. ssh key-generator



Respuesta correcta

La respuesta correcta es:

ssh-keygen

Pregunta 20

Correcta

Puntúa 1,00 sobre 1,00

¿Cómo empezamos a usar git?

- a. En el directorio raíz tecleamos git init
- b. Creando un directorio git
- c. En el directorio principal tecleamos git init
- d. En el directorio principal tecleamos git init start



Respuesta correcta

La respuesta correcta es:

En el directorio principal tecleamos git init

TEMA 2

¿Qué comando hemos empleado en prácticas para generar una llave pública y una privada?

ssh-keygen

¿Cuál es el OpenTag por defecto para PHP en CentOS?

<%php

¿Cuál es el CloseTag por defecto para PHP en CentOS?

?>

Los SCM centralizados como Subversion son obsoletos y están descontinuados

Falso

¿Cómo comparto la configuración de .gitignore con el resto de colaboradores de un proyecto GIT?

Lo registro en el repositorio compartido como cualquier otro contenido de git

SSH emplea cifrado de llave privada

Tras el handshaking por su alta eficiencia

¿Qué opción del firewall de centos deshabilita el acceso al puerto 81?

--remove-port=81

¿Los cambios en la configuración del firewall de CentOS realizados por linea de comandos están activos tras reiniciar?

Solo si se emplea la opción --permanent en los cambios

En la instalación por defecto de las distribuciones de Linux empleadas en clase, ssh estaba...

No instalado en Ubuntu pero si en Centos

rsync permite sincronizar directorios locales y entre equipos remotos
Verdadero

¿Dónde se encuentran los logs de Apache?
/var/log/httpd

¿Qué comando configura el email del usuario de GIT?
gitconfig--local user.email yo@ugr.es

La llave privada la utilizamos para
Identificarnos en las comunicaciones

¿Qué opción muestra si PHP está en ejecución?
Ninguna de los anteriores, PHP no funciona como servicio

¿Qué opción configura apache para arrancar con el Sistema en CentOS?
systemctl enable httpd

¿Cómo busco el nombre de un paquete para instalar en Ubuntu?

apt-cache search

¿Cómo valida SSHD la identidad de un usuario que accede sin contraseña?

SSHD tiene su llave publica, luego ya lo tiene identificado

Las ventajas aportadas por un Sistema de Control de Versiones son :

Todas las anteriores son ciertas

Identifica los cambios y permite trazarlos

Mantiene la historio completa del proyecto

Facilita el desarrollo en ramas

¿Qué comando realiza una copia de seguridad completa de la primera partición del segundo disco sata del equipo?

dd if=/dev/sdb1 of=/tmp/sdb1.img

SSH emplea...

Cifrado de Llave Pública-Privada y de llave Simétrica

¿Con qué comando GIT inicializamos un repositorio?

init

¿Dónde se encuentra el fichero de configuración de PHP?

/etc/php.ini

¿Qué afirmación es correcta sobre git?

- Es un VCS con repositorio distribuido
- Facilita el trabajo en ramas y mezclas
- Optimiza el uso en disco de los repositorios
- Todo lo anterior es cierto

¿Cómo asegura SSH la identidad del cliente?

Pidiéndole que cifre información con su llave privada.

¿Qué comando restaura todos los archivos contenidos en una copia de seguridad backup.cpio?

cpio -iu <backup.cpio

¿Qué comando empleamos para comprobar si el firewall de Ubuntu está activo?

ufw status

Para que rsync funcione con ssh debemos configurar los servidores origen y destino de la información

Instalar rsync en origen y destino, pero sshd solo en el destino.

¿Dónde se encuentra el fichero de configuración de Apache en CentOS?

/etc/httpd/conf/httpd.conf

Tras realizar una mezcla "merge" entre dos ramas y resolver los conflictos ¿Qué afirmación es cierta? DUDA!!!!!!!!!!!!!!

Los cambios necesitan commit y push para ser visibles en el repositorio de origen.

Los cambios son visibles en el repositorio de origen. (ESTA ES LA QUE CREEMOS)

Los cambios precisan una operación pull y push para publicarlos en el repositorio de origen.

En un merge no se producen cambios, solo la mezcla de ramas, y no es necesario comunicarlos.

Cual es la diferencia entre los comandos git branch y git checkout -b

branch crea la rama, checout la crea y cambia la rama de trabajo

El directorio /var está montado en /dev/sdb1 ¿Qué comando realiza una copia de todos los archivos .log de la partición?

find /var -iname '*.log' | cpio -o > varlog.cpio

[Página Principal](#) / Mis cursos / [GRADUADO-A EN INGENIERÍA INFORMÁTICA \(2010\) \(296\)](#) .

/ [INGENIERÍA DE SERVID \(2021\)-296_11_35_2021](#) / [Práctica 2](#) / [Examen P2 Grupo C1](#)

Comenzado el martes, 10 de noviembre de 2020, 15:40

Estado Finalizado

Finalizado en martes, 10 de noviembre de 2020, 16:13

Tiempo empleado 33 minutos 32 segundos

Puntos 13,33/20,00

Calificación 6,67 de 10,00 (67%)

Pregunta 1

Correcta

Puntúa 1,00 sobre 1,00

¿Cómo podríamos enviar una copia de seguridad a través de SSH usando RSync?

- a. rsync prueba/* usuario@IP:/home/usuario/pruebaBackup
- b. rsync -e "ssh -p 22022" usuario@IP:/home/usuario/pruebaBackup
- c. rsync -e "ssh -p 22022" prueba/* usuario@IP:/home/usuario/pruebaBackup □
- d. rsync -e "ssh -p 22022" prueba/* /home/usuario/pruebaBackup

Respuesta correcta

La respuesta correcta es:

rsync -e "ssh -p 22022" prueba/* usuario@IP:/home/usuario/pruebaBackup

Pregunta 2

Incorrecta

Puntúa -0,33 sobre 1,00

¿Qué sucede si se borra el directorio .git?

- a. Se borra el control de versiones para los ficheros en local
- b. Se borra el control de versiones de la rama actual □
- c. Se borra tanto el control de versiones como los ficheros en local
- d. No se borra nada importante, solo guarda información de Logs

Respuesta incorrecta.

La respuesta correcta es:

Se borra el control de versiones para los ficheros en local

Pregunta 3

Incorrecta

Puntúa -0,33 sobre 1,00

¿Cuál es la contraseña para root tras instalar MariaDB en CentOS?

- a. Depende de qué paquete hayamos instalado
- b. La misma que la del SO
- c. La que le indiquemos en el archivo de configuración
- d. Por defecto no tiene contraseña



Respuesta incorrecta.

La respuesta correcta es:

Por defecto no tiene contraseña

Pregunta 4

Correcta

Puntúa 1,00 sobre 1,00

¿Cómo enviamos archivo1.txt al repositorio remoto?

- a. git commit -am "archivo.txt"
- b. git push origin NombreRama
- c. Ninguna de las otras respuestas es correcta
- d. git add archivo1.txt



Respuesta correcta

La respuesta correcta es:

git push origin NombreRama

Pregunta 5

Correcta

Puntúa 1,00 sobre 1,00

¿Cual es el significado del acrónimo LAMP?

- a. Linux, Apt, Mysql, Perl
- b. Level, Apache, Mysql, PHP
- c. Linux, Apache, Mysql, PHP
- d. Linux, Apt, Mysql, PHP



Respuesta correcta

La respuesta correcta es:

Linux, Apache, Mysql, PHP

Pregunta 6

Correcta

Puntúa 1,00 sobre 1,00

¿Cómo enviamos archivo1.txt al repositorio local?

- a. git add archivo1.txt --stage
- b. git add archivo.txt
- c. Ninguna de las otras respuestas es correcta
- d. git add archivo1.txt



Respuesta correcta

La respuesta correcta es:

Ninguna de las otras respuestas es correcta

Pregunta 7

Correcta

Puntúa 1,00 sobre 1,00

¿Qué diferencia hay entre git reset y git revert?

a.

- a. git reset retrocede a un commit eliminando los anteriores mientras que git revert crea un nuevo commit del deseado sin eliminar los anteriores
- b. git reset y git revert hacen exactamente lo mismo
- c. git revert retrocede a un commit eliminando los anteriores mientras que git reset crea un nuevo commit del deseado sin eliminar los anteriores
- d. git reset te resetea el repositorio y git revert solo le lleva a un commit antes

Respuesta correcta

La respuesta correcta es:

git reset retrocede a un commit eliminando los anteriores mientras que git revert crea un nuevo commit del deseado sin eliminar los anteriores

Pregunta 8

Correcta

Puntúa 1,00 sobre 1,00

El comando: git diff HEAD

- a. mostrará las diferencias entre el stage y el directorio raíz
- b. no mostrará nada si el o los archivos modificados acaban de ser seguidos
- c. mostrará las diferencias entre el repositorio y el directorio local
- d. ninguna de las otras respuestas es correcta

Respuesta correcta

La respuesta correcta es:

mostrará las diferencias entre el repositorio y el directorio local

Pregunta 9

Correcta

Puntúa 1,00 sobre 1,00

¿Cómo comprobamos el estado del repositorio git?

- a. git commit
- b. git state
- c. git repo status
- d. git status



Respuesta correcta

La respuesta correcta es:

git status

Pregunta 10

Correcta

Puntúa 1,00 sobre 1,00

¿Qué gestores de paquetes reemplazarán a apt y yum?

- a. DNF y snap
- b. NDF y smap
- c. DNS y Snap
- d. sanp y FDF



Respuesta correcta

La respuesta correcta es:

DNF y snap

Pregunta 11

Correcta

Puntúa 1,00 sobre 1,00

¿Usando git puede diferir el contenido del repositorio con el del directorio de trabajo actual?

- a. Sí, también puede diferir del stage
- b. No, add se encarga de eso
- c. No, commit se encarga de eso
- d. Sí, si no hemos hecho commit



Respuesta correcta

La respuesta correcta es:

Sí, también puede diferir del stage

Pregunta 12

Incorrecta

Puntúa -0,33 sobre 1,00

¿Cómo podemos saber en qué puerto está escuchando un servidor SSH?

- a. systemctl sshd.service status
- b. cat /etc/sshd/sshd_config | grep port
- c. systemctl status sshd.service
- d. systemctl status ssh.service



Respuesta incorrecta.

La respuesta correcta es:

systemctl status sshd.service

Pregunta 13

Correcta

Puntúa 1,00 sobre 1,00

¿Qué hace el servicio fail2ban?

- a. Bloquear usuarios tras varios intentos de acceso fallidos
- b. Bloquear direcciones de un protocolo de Internet tras varios intentos fallidos de acceso
- c. "Banear" clientes en caso de uso abusivo del servidor
- d. Activa el cortafuegos para evitar el acceso a través de un puerto



Respuesta correcta

La respuesta correcta es:

Bloquear direcciones de un protocolo de Internet tras varios intentos fallidos de acceso

Pregunta 14

Correcta

Puntúa 1,00 sobre 1,00

¿Qué hace el script ssh-copy-id?

- a. Copia la llave privada en el servidor local
- b. Copia la llave pública en el servidor remoto
- c. Copia la llave privada y publica en el servidor
- d. Copia la llave privada en el servidor remoto



Respuesta correcta

La respuesta correcta es:

Copia la llave pública en el servidor remoto

Pregunta 15

Incorrecta

Puntúa -0,33 sobre 1,00

¿Qué diferencia una página estática de una dinámica?

- a. Las páginas estáticas no tienen ninguna animación y las dinámicas si
- b. Las páginas dinámicas permiten conexión a BBDD
- c. Las páginas estáticas no tienen interacción alguna con el usuario y las dinámicas si
- d. Las páginas estáticas no pueden llevar links

[◀ Examen A1](#)[Ir a...](#)**Pregunta 16**

Correcta

Puntúa 1,00 sobre 1,00

¿Cuál de los siguientes comandos permite crear una imagen a partir de un disco?

- a. rsync -aviz /home/imagen.iso root@localhost:*.iso
- b. dd if=/dev/sda of=~/hdadisk.img
- c. tar -xvf imagen.tar.gz /home/misimagenes
- d. cpio -idv /dev/sda > /var/imagen.iso

Respuesta correcta

La respuesta correcta es:

dd if=/dev/sda of=~/hdadisk.img

Pregunta 17

Correcta

Puntúa 1,00 sobre 1,00

¿Cuál es el problema principal al instalar LAMP mediante tasksel?

- a. Hay que instalar tasksel y es espacio innecesario en el disco duro
- b. Que puede dar fallo y hay que hacer un apt-get
- c. Al instalar LAMP desde tasksel puede haber problemas de seguridad
- d. No se pueden controlar las versiones que se instalan y podría llegar a dar algún tipo de problema

Respuesta correcta

La respuesta correcta es: No se pueden controlar las versiones que se instalan y podría llegar a dar algún tipo de problema

Pregunta 18

Correcta

Puntúa 1,00 sobre 1,00

¿Qué comando debemos ejecutar para que SELinux permita la conexión del servicio SSH en el puerto 22222?

- a. selinux port -a -t ssh_port_t -p udp 22222
- b. semanage port -a -t ssh_port_t -u udp 22222
- c. semanage port -a -t ssh_port_t -p tcp 22222
- d. semanage port -a -t 22222 ssh_port_t



Respuesta correcta

La respuesta correcta es:

semanage port -a -t ssh_port_t -p tcp 22222

Pregunta 19

Incorrecta

Puntúa -0,33 sobre 1,00

El comando: git diff

- a. no mostrará nada si el o los archivos modificados acaban de ser seguidos
- b. mostrará las diferencias entre el archivo seguido y el del directorio raíz
- c. mostrará las diferencias entre el stage y el repositorio
- d. ninguna de las otras respuestas es correcta



Respuesta incorrecta.

La respuesta correcta es:

no mostrará nada si el o los archivos modificados acaban de ser seguidos

Pregunta 20

Correcta

Puntúa 1,00 sobre 1,00

¿Para que sirve git log?

- a. Permite ver un resumen de un commit
- b. Permite ver un resumen los ficheros que hay en ese git
- c. Permite mostrar mensajes de log en la terminal
- d. Permite ver un resumen de todos los commits en orden de entrada en la pila



Respuesta correcta

La respuesta correcta es:

Permite ver un resumen de todos los commits en orden de entrada en la pila

· Si modificamos un archivo en el directorio de trabajo y lo añadimos con git add nom_fich
¿qué mostrará git diff?

NADA

· ¿Qué diferencia hay entre git reset y git revert?

Git reset retrocede a un commit eliminando los anteriores mientras que git revert crea un nuevo commit del deseado sin eliminar los anteriores

· ¿Qué pasos hay que seguir en CentOS para que un servidor web muestre el contenido de un fichero index.php?

Se debe crear el fichero en /var/www/html e indicarle al fichero de configuración de Apache que será el fichero principal del servidor

· El comando: git diff

No mostrará nada si el o los archivos modificados acaban de ser seguidos

· ¿Qué diferencia hay entre añadir --permanent o no al abrir un puerto con firewall-cmd?

El --permanent lo abre al recargar pero para siempre, de la otra forma es inmediato y temporal

· ¿Para qué sirve el comando screen?

Para permitir dejar SSH en Background

· Al instalar el servidor Apache en CentOS

Hemos de activar/habilitar e iniciar el servicio

· ¿Cómo enviamos archivo1.txt al repositorio local?

Ninguna es correcta (todas llevaban add, yo creo que es con push)

· ¿Por qué debemos bloquear el usuario root en un servidor SSH?

Por seguridad para evitar ataques a usuarios por defecto

· ¿Cómo se avisa a SELinux que se va a cambiar el puerto de SSH?

semanage port -a -t ssh_port -t tcp 22022

· ¿De dónde surge SSH?

Es un protocolo desarrollado para paliar vulnerabilidad de Telnet

· ¿Cuál es el problema principal al instalar LAMP mediante tasksel?

No se pueden controlar las versiones que se instalan y podría llegar a dar algún tipo de problema

· ¿Cómo comprobamos el estado del repositorio git?

Git status

· ¿Cuál es la contraseña para root tras instalar MariaDB en CentOS?

Por defecto no tiene contraseña

· ¿Qué ocurre tras ejecutar: firewall-cmd --add-port=443/tcp ; firewall-cmd --reload?

El puerto 443 no altera su configuración

· ¿Qué ocurre tras ejecutar: firewall-cmd --permanent --add-port=443/tcp ; firewall-cmd --reload?

El puerto 443 está abierto

· ¿Para qué sirve git log?

Permite ver un resumen de todos los commits de orden de entrada en la pila

· ¿Qué hace el servicio fail2ban?

Bloquear direcciones de un protocolo de Internet tras varios intentos fallidos de acceso

· ¿Qué comando debe ejecutar para crear un archivo comprimido a partir de una carpeta?

tar cvzf folder.tar.gz /home/folder

· ¿Cómo podemos saber en qué puerto está escuchando un servidor SSH?

```
systemctl status sshd.service
```

¿Qué ocurre tras ejecutar: firewall-cmd --add-port=22022/tcp?

El puerto 22022 está abierto para la sesión actual.

Cuando instalamos en CentOS un servicio, ¿cómo se comporta por defecto?

Está deshabilitado e inactivo.

Rkhunter viene de

RootKit Hunter

Los puertos por defecto para SSH y HTTP son

El 22 y el 80 respectivamente

¿Cuál de los siguientes comandos permite activar la apertura de un puerto configurado anteriormente?

```
Firewall-cmd --reload
```

LECCION 1

Cortafuegos: Igual que los RAID que tienen HW y SW. Hacen que una máquina dedicada a dejar caer ciertos paquetes en base a un análisis o programación o un software que los monitorice y los deje caer según una condición. Uno de los firewall es hipertable (?) que establece una cadena de reglas, es relativamente complejo y se sale un poco de la asignatura. Servidores web de altas prestaciones si llegan a configurarlo.

Para los que nunca han usado git, se explican dos o tres cosas y los resultados de la practica fallan en los exámenes.

YUM (REDHAT, CENTOS, FEDORA) y APT(DEBIAN, UBUNTU,...): Aplicaciones que nos permiten conectarnos con un servidor, ver la lista de paquetes que tiene y descargarlos de ahí.

Hacemos yum search para buscar qué tenemos. No ponemos interfaces graficas en nuestros servidores porque bajan el rendimiento, es un programa en ejecución y es mas seguro tener menos cosas en ejecución y porque las maquinas no van a tener pantalla, sino que vamos a hacer ssh desde casa o el trabajo, de forma remota para acceder al cluster.

Yum redirige a DNF, la evolución de yum (que tiene por debajo dnf)

En la rama DEBIAN tenemos apt. Se empezó a trabajar en los paquetes que son snap y Canonical o Red Hat querían converger a los snap. Los snap llevan las dependencias incorporadas, con lo cual es menos caótico de cara a gestionar esas dependencias sin resolver. Red Hat y Fedora siguieron con dnf y Canonical y Ubuntu usan de forma oficial los snaps.

SSH:

Viene de Secure Shell. Cuando nos conectábamos de una maquina a un servidor utilizábamos telnet y con wireshark veremos que la info que viaja es con el usuario y la contraseña en texto plano. Entonces, un alumno detectó la vulnerabilidad e implantó por seguridad el ssh. En centos no lo pregunta, directamente lo instala.

El cliente es ssh e inicia la conexión y nuestro servidor ejecuta un servicio que es sshd, que no es lo mismo. Son diferentes /etc/sshd_config y /etc/ssh_config.

Cosas importantes a configurar sobre seguridad. Root es el usuario de todas las maquinas, así que pruebo a hacer root en la opción de login:

Ssh IP -l root

Resulta que centos tiene el permit root login puesto a true. En Ubuntu:

Ssh IP -l root

No te deja porque root no tiene contraseña en Ubuntu. Por ello vamos a editar con VI en Ubuntu. Si pones contraseña con “sudo passwd root” igualmente no te dejaría.

Los servicios



Con sshd nos conectamos al puerto (el puerto del que escucha) es el 22. Podemos cambiar el puerto para que nadie se conecte. Lo cambiamos con “sed”

COSAS QUE HACE EN MAQUINAS VIRTUALES:

CENTOS:

- Yum search ssh
 - o Aparecen las coincidencias en nombre
 - o También “dnf search ssh”
- Yum provides -> quien te proporciona el archivo
 - o Por ejemplo “yum provides libc.so.6”
- Yum provides libmpfr.so.4
 - o Te dice que viene de base
- Yum provides mc
 - o Mc-1:4.8.19.....
- Ps -Af | grep ssh
 - o Nos encontramos un proceso de sshd que esta como demonio
- Ssh localhost
 - o Accedes a tu propia maquina. Te dice que es la primera vez que te vas a conectar a ti mismo
 - o Yes
 - o Practicas, ISE
 - o Con eso te logeas
 - o Ctrl+d -> te sales
- Cd .ssh -> y te aparece dentro “known_hosts”
 - o Estará localhost y su fingerprint si le haces un cat
- Vi /etc/ssh/sshd_config

- Vemos que permit root login esta a yes
 - Lo pongo a que no, escape y :wp
- Vamos a probar que lo he restringido
 - Ssh IP -l root
 - Vemos que si te deja porque no ha reiniciado el servicio.
- Systemctl restart sshd.service
- Porbamos de nuevo haberlo restringido y ya no nos deja.
 - Para asegurarnos probamos “Ssh IP -l root -v”
 - No te deja
- Sed s/'Port 22'/'Port 22322'/ -i /etc/ssh/sshd_config
- Lo comprobamos
 - Ssh IP -p 22322
 - Fallo porque no ha hecho “systemctl restart sshd”
 - Comprobamos de nuevo tras ello y tampoco te deja, pues lo haces con “-v” al final. Al hacer “vi /etc/ssh/sshd_config” vemos que hemos modificado el puerto pero que tenemos la línea comentada por defecto. Lo descomentamos y escape :wq y reiniciamos el sistema. Dara un fallo y ponemos “systemctl status sshd.service” y no ve nada. Y hace “journalctl -xe” ve que intentamos enlazar al puerto 22322 y da un fallo de permisos. Para terminar y arreglarlo vamos a “vi /etc/ssh/sshd_config” te explica que debemos avisar al so de que vamos a cambiar el puerto con:

```
SELinux about this change.
semanage port -a -t ssh_port_t -p tcp #PORTNUMBER
```

UBUNTU:

- Atp-get -> en desuso
- Apt es equivalente a yum
- Para buscar un proceso perdido:
 - Ps | grep ssh
 - No muestra nada porque ninguno cuelga de este bash
 - Ps -Af | grep ssh
 - Ahí si aparece
- Vamos a instalar ssh en Ubuntu:
 - Sudo apt instal tasksel
 - Sudo tasksel
 - Damos en openssh y OK
 - Apt search ssh | grep server
 - Sudo Apt install openssh-server
 - Es la alternativa a lo de tasksel y asi sabemos que versión instalamos
 - Ps -Af | grep ssh
 - Vemos que está en ejecución y eso esta mal, porque el tasksel no tiene permisos para arrancar el permiso y aun asi lo pone.
- Para configurar lo de root:
 - Vi /etc/ssh/sshd_config
 - Vemos que esta prohibido root con contraseña por defecto

LECCION 2

COPIAS DE SEGURIDAD Y CONTROL DE VERSIONES (CON GIT)

Pregunta de examen: ¿Opciones mas comunes de resync?

COPIAS VS BACKUP

Si haces Ctrl C de un archivo, has hecho una copia, pero la diferencia de un backup es que este tiene una metainformacion o metadatos superiores sobre versiones, fechas, etc... El backup también se puede colocar físicamente en otro sitio. El backup es una ciencia para él.

Comandos para las copias de seguridad:

Dd -> copia binaria bit a bit y se puede utilizar a dia de hoy para recuperar un disco o un dispositivo móvil. Tenemos un inputfile y un outputfile, creamos nuestra copia de archivos, podemos especificar un archivo de imagen y para deshacer el cambio, cambiamos el orden de los archivos y asi nos lo llevamos a nuestro disco. La entrada de bloque esta a 512 bytes y podemos poner 4Kb para que sea más rápida. Si haces eso, para recuperar el contenido de una tarjeta después dejas quieta la tarjeta y jugamos con el archivo generado o el dispositivo donde hayamos copiado la imagen de la información original, así, de tener que deshacer alguna modificación, aun se puede. El problema de dd es que nos permite hacer copias pero la parte de metadatos tenemos que ponerla nosotros (añadiendo al nombre de archivo la ejecucion, ubicación, etc) Como ventaja es que tenemos la copia exacta bit a bit

Cpio-> Coge el listado de archivos de la entrada estándar y lo convierte en un único archivo (pero no comprime). Aunque está en desuso, las ventajas de esto son que si usamos sftp para llevarte las copias de seguridad a otro sitio y queremos enviar millones de archivos, puede ser conflictivo y crea un trafico innecesario en la red, entonces con cpio coge esos millones de archivo, lo mete en uno solo y se ahorra tiempo. Otra cuestión es que si nos preocupa el tiempo que tarde en enviarse los archivos. Siempre queremos que sea cuanto más rápido, pero además, en el caso de que se interrumpa la copia. Para esto, tenemos en cuenta la frecuencia del backup (por ejemplo algunas cosas las quiero cada 12h, o 24h o algo asi) entonces si tardo en hacer la copia más que lo que hay entre copia y copia, no las estoy haciendo bien. Una posible solución es compresión pero el tio se refiere a una cache. Poner un servidor intermedio al que mandar los datos muy rápido y ya lleva la información a ese tercer lugar, que es el destino original el tiempo que necesite. El intermediario en modo Bach en cola va llevando la información a la ubicación tercera. Es como un proxy o buffer. Otra posible solución es que no necesitamos cada X tiempo el 100% de los datos, sino que solo necesitamos guardar lo que hemos modificado. Eso es un backup incremental: tenemos una imagen base y vamos almacenando las diferencias y es todo lo que mandamos. Pasa de "26" horas a "1" h. Tambien nos ahorraremos espacio. Otra cosa a tener en cuenta, la información delicada. Si va a tener información delicada la nube a la que mandamos la copia, mejor que este en España o

Europa para que este vigente la LOPD (española) y GDPR (europa) y ya como mucho mirar convenios para países asociados como canada (nunca china, ni eeuu, ni Australia, ni mexico).

Tar-> sustituo de cpio. La única diferencia es que en vez de tener que especificar en la entrada estándar el listado de archivos, se le especifica un conjunto de archivos o directorios que empaquetara. Tambien tiene la opción de comprimirlo y de ahí la extensión .gz. En la orden “tar cvzf MyImages...” cvzf significa create overboss zip file , que es en modo overboss y comprimido. Para comprimirlo es “tar -xvf public_html.tar” que es x -> extract v->overboss f->file. SFTP lleva la S delante para que sea seguro o no estaremos respetando la LOPD. Sabernos el tar de memoria.

- **cpio**

- Copia archivos a y desde (archivos)

- `ls | cpio -ov > /tmp/object.cpio`
 - `cpio -idv < /tmp/object.cpio`

- **tar**

- Ejemplos:

- `tar cvzf MyImages-14-09-17.tar.gz /home/MyImages`
 - `tar -xvf public_html-14-09-17.tar`

Rsync -> Dropbox de los viejos. Hacian sincronizaciones a mano. Permite sincronizar una fuente y un destino de los datos y se puede hacer a través de ssh. Así se cumple la normativa legal. Además permite los envíos incrementales y reduce el tiempo que necesita. Puede agrupar todos los cambios en un archivo único y si sincronizamos dos directorios y hemos borrado algo sin querer, tenemos un problema porque lo borra.

Otra cosa importante en los backups es el tiempo de recuperación. Este hombre dejaba var en var_OLD porque si queremos volver a la configuración original, con hacer un mv /var_old /var , estaba todo listo. Hay que tener cuidado con -- delete en rsync porque borra. Se prueba:

Crear un archivo: mkdir 1 mkdir2 touch 1 y cosas así y probar a hacer resync, --delete y cosas así. Hay que probarlas para saber usar el . * / username, maquina y eso. Para usar ssh para que vaya cifrado es “ -e “ssh” ”. Hay que saber de memoria el uso común de rsync:

`rsync -aviz /home usu@maquina:/backup`

La iz del final es que haga un informe y que vaya comprimido. Para restaurar con rsync invertimos el orden del origen y del destino. Lo usaremos con algún cluster. Va a preguntar si o si las opciones más comunes, precauciones a tomar, como hacer transferencia de información, etc

SO para backups-> tartarus, backup2l, duplicity, sftpclone que implican un servicio completo con BD, demonios y cosas así. Es más compleja y tiene sobre cargar mayor en el sistema. A veces puede interesar monitorizar el estado del backup sin ir a la consola. Error para que aprendamos:

Todo esto puede pasarle fácilmente...

Destroyed Working Backups with Tar and Rsync (personal backups)

I had only one backup copy of my QT project and I just wanted to get a directory called functions. I end up deleting entire backup (note -c switch instead of -x):

```
cd /mnt/bacupusbharrdisk  
tar -zcvf project.tar.gz functions
```

I had no backup. Similarly I end up running rsync command and deleted all new files by overwriting files from backup set (now I have switched to [rsnapshot](#))

```
rsync -av -delete /dest /src
```

Again, I had no backup.

CONTROL DE CAMBIOS

Se puede hacer un cp pero es cutre porque no se añaden metadatos. No es lo ideal aunque pueda ser útil. GIT es un sistema de control de versiones. Hay herramientas que permiten guardar etc por ejemplo con git. Ventajas:

- Ventajas (para ISE)
 - Ponemos un pie en devops
 - [Seguimiento de los cambios \(y de su autor\)](#)
 - Permite planificar un entorno de prueba entre varias personas
 - ...

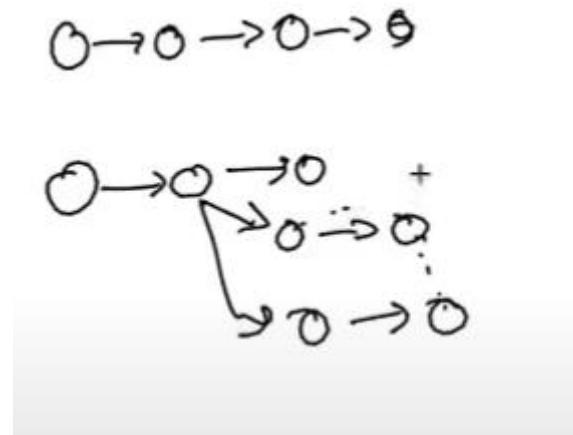
Desarrollo git -> Linus Benedict Torvalds. Github ahora pertenece a Microsoft. Desarrollo el kernel de Linux y alternativamente git. Github establece un frontend web para sus servidores que están ejecutando los servicios de git.

Cómo funciona git:

Tenemos un directorio de trabajo, trabajamos con estos archivos y los cambiamos, por lo que los marcamos para seguimiento y lo metemos a nivel abstracto en una zona que es el stage y en un paso final se pasa al repositorio. Ahí tenemos la instantánea actual de nuestro proyecto que puede ser código o cualquier otro tipo de archivos. No aportaría mucho que lo hiciésemos sobre imágenes porque se realiza una compresión que no podríamos hacer con imágenes y se nos iría el espacio. Mejor para archivos de texto. Cuando hacemos un commit, que es pasar de los archivos de seguimiento (los cambiados y marcados para seguir, que por ejemplo contraseñas y tal no se marcan para seguir) al repositorio. Cuando hacemos esos commit, se genera un hash de unos 40 caracteres para identificar esa transacción aunque con los 7 primeros suele valer. Esos commit se van colocando en forma de grafo dirigido acíclico y ese último commit es un puntero denominado a head. Podemos hacer referencias relativas de movernos hacia atrás x posiciones y cosas así:

- Podemos desplazarnos a partir del identificador (con ~ y ^):
 - HEAD~3 == HEAD^^^ == HEAD^3
 - Se pueden combinar: HEAD~3^2
- Podemos desplazarnos a partir del identificador (con ~ y ^):
 - HEAD~3 == HEAD^^^ == HEAD^3
 - Se pueden combinar: HEAD~3^2

si nos perdemos, nos reubicamos con git reflog. Cuando veamos la salida de git veremos los dos puntos que significan un intervalo lineal entre un commit y otro, y si son 3 puntos es que no es lineal. Lo de lineal significa que vamos haciendo commits y puede que se ramifiquen, entonces de algún commit a otro no haya referencia lineal:



Tenemos unos blobs (binary large objects) que son archivos en binario. Entonces tiene los datos y los inodos correspondientes. Cada objeto tiene su hash identificativo, por lo que git es un sistema de archivos que es direccionable por contenido, ya que este contenido está dentro del blobs. Al final lo que tenemos son llaves y valores identificativos. Se pueden poner etiquetas también a los objetos.

AHORA VA A LA TERMINAL DE CENTOS PARA HACER PRUEBAS:

WORKFLOW: servidor central donde tenemos un repositorio con dos equipos de trabajo y cada uno tiene su directorio de trabajo, su zona de seguimiento y su repositorio propio. Y un servidor que va a ser centos. No viene instalado por defecto:

- Dnf/yum install git
- Git init -> iniciar un repositorio
- Crear un directorio para los repositorios:
 - o Sudo mkdir /git
 - Practicas,ISE
- Sudo dnf install git
 - o si
- sudo dhclient
 - o para lanzar el cliente de dhcpc o dhcp o algo asi para que refresque la ip de np0s3 y np0s8.
- Cd /git
- Si haces git init te da fallo porque como lo has creado en root entonces el propietario es root y el grupo es root (dar permiso de 777 es mejor que no, es una locura)
- Creamos un usuario git y añadimos el resto de usuarios al grupo de git.
 - o Sudo adduser git
 - o Groupadd le da fallo asi que prueba otra cosa
 - o Sudo vi /etc/group
 - Abajo del todo junto a "git:x:...."escribe: Alberto:Ana
 - :wq
 - o Sudo chgrp -R git /git
- Primero el añadimos el 77 en los permisos
 - o Sudo chown -R git /git
 - o Ls -la / -> para probarlo
 - o Sudo chmod 770 /git
 - o Ls -la / -> comprueba
- Git init repo1 - -bare
 - o Le da fallo y se tira la vida. Termina haciendo trampas.
 - o Lo que hemos hecho con el init es iniciar el bare repo. El bare no tiene directorio de trabajo, con respecto a los repos normales.
 - o Ls -> se ve repo1
 - o Cd repo1
 - o Ls -la -> se vera:

```
(alberto@localhost:repo1) ls  
branches config description HEAD hooks info objects refs  
(alberto@localhost:repo1) ls -la  
total 16  
drw-rw-r--. 2 alberto alberto 119 oct 29 12:03 .  
drw-rw-r--. 3 git git 19 oct 29 12:03 .git  
drw-rw-r--. 2 alberto alberto 6 oct 29 12:03 branches  
rwx-w-r--. 1 alberto alberto 66 oct 29 12:03 config  
rwx-w-r--. 1 alberto alberto 23 oct 29 12:03 description  
rwx-w-r--. 1 alberto alberto 23 oct 29 12:03 HEAD  
drw-rw-r--. 2 alberto alberto 4096 oct 29 12:03 hooks  
drw-rw-r--. 2 alberto alberto 21 oct 29 12:03 info  
drw-rw-r--. 4 alberto alberto 38 oct 29 12:03 objects  
drw-rw-r--. 4 alberto alberto 31 oct 29 12:03 refs
```

- Sudo useradd Ana
 - o Practicas,ISE
- Sudo vi /etc/ssh/sshd_config
 - o Pone el port 22
- Systemctl start ssh

- Abre una terminal y se mete en el usuario Ana y en otra terminal esta dentro de Alberto. Ahora vamos a traernos el repositorio del servidor desde alberto
 - o Git clone IP:/git/repo1
 - Se obtiene la IP con “Ip addr” y mira la 3
 - Practicas,ISE
- Ana se va a su home y hace:
 - o Git clone IP:/git/repo1
 - Practicas,ISE
- Alberto entra dentro de repo1
 - o Cd repo1/
 - o Ls -la

```
alberto@localhost:~/BORRAME/repo1$ ls -la
total 12
drwxr-xr-x. 3 alberto alberto 4096 oct 29 17:08 .
drwxr-xr-x. 4 alberto alberto 4096 oct 29 17:08 ..
drwxr-xr-x. 7 alberto alberto 4096 oct 29 17:08 .git
```

- Ana hace lo mismo.

```
[Ana@localhost ~]$ cd repo1/
[Ana@localhost repo1]$ ls -la
total 12
drwxrwxr-x. 3 Ana Ana 4096 oct 29 17:08 .
drwx-----. 5 Ana Ana 4096 oct 29 17:08 ..
drwxrwxr-x. 7 Ana Ana 4096 oct 29 17:09 .git
```

- Lo que vemos es el working directory. Lo particular es el subdirectorio de .git. Es lo que tenemos en el bare repo. Si tengo nosecuantos commit y se te escapa un rm repo1, la información se pierde porque es donde esta la info. De las 3 copias, las 3 son validas. Pueden tener sus cambios locales pero a nivel de repositorio son validos.
- Alberto se va a repo1:
 - o Touch README.md
 - o Git status -> mucha información como que readme es archivo sin seguimiento. Que si queremos seguirlo es con git add <nombre de archivo>
 - o Git commit -> para actualizar las repos, pero no hay nada
 - o Git add README.md
 - o Git commit -m “hemos creado el archivo README.md”
 - o Ls .git/ -la -> hemos cambiado el head
 - o Git status -> no hay nada porque esta todo actualizado. Hemos actualizado el repo de Alberto pero no el bare y lo hacemos con push y pull
 - o Git push
 - Practicas,ISE
- CENTOS:
 - o En repo1 seguimos viendo lo mismo, por que? Porque es tipo bare y no hay working directory
- Ana:
 - o Ls -> dentro de repo1 y no hay nada
 - o Git pull
 - Practicas,ISE
 - o Ls-> ya se ve README.md
- Alberto:

- Vi README.md -> dentro escribe # Empezamos con git...
 - Git status
 - Git diff -> vemos las diferencias entre el working directory y el área de seguimiento.
 - Git add README.md
 - Git diff -> no muestra nada porque ya no hay diferencia entre lo del working dir y el área de seguimiento
 - Git diff HEAD -> diferencia entre wd y el repositorio y se ve que hemos añadido esa línea.
 - Git commit -m "Actualizamos README.md"
 - Git diff -> nada
 - Git diff HEAD -> nada
 - Git diff -staged -> nada
 - Vi README.md
 - Añade "#Titulo:"
 - Git diff -> salen cosas
 - Git diff HEAD -> también
 - Git add README.md
 - Git diff -> ya no sale nada
 - Git diff HEAD -> sale la diferencia
 - Git diff -staged -> diferencia entre lo del repo y el seguimiento
 - Git commit -m "Añadimos titulo"
 - Git push
- Ana:
- Git fetch
 - Git merge
 - Cat README.md
- Alberto
- Vi README.md
 - Pone después de titulo: en un lugar...
 - Git commit -a -m "readme con titulo"
- Ana:
- Vi README.md
 - Pone después de titulo: otro titulo
 - Git add README.md
 - Git commit -m "titulo nuevo"
 - Git push -> le da fallo

COSAS IMPORTANTES A TENER EN CUENTA:

-m "mensaje" en los commit, el mensaje tiene que ser ilustrativo y significativo. Los commits deben ser lo mas atomicos posibles. A lo mejor no cada vez que cambiemos una línea pero si hacerlos atomicos porque nos permitirá luego encontrar esa aguja en el pajar. Se busca el mensaje y lo encontramos rápido. Si no, haciendo un commit con 100 lineas, habría que revisarlas todas. Lo ideal es que los commit, cuanto mas pequeños mejor. Aporta trazabilidad.

Pull lo que hace es traer información del servidor y las une. (fetch y merge). Eso es que te la descargas del servidor y luego la recuperas. Git pull = git fetch + git merge. La cosa es que si

una persona hace un commit y otra hace otro commit y se adelanta, le va a dar fallo y tendrás que hacer el push a mano.

LECCION 3

SSH viene instalado en Centos pero en Ubuntu server no, nos pregunta en la instalación si lo queremos. En cuanto a nomenclatura, en centos es sshd y en Ubuntu es ssh y sshd. Sobre root puede entrar en Ubuntu no con password y en centos el permit root login esta a yes. El firewall esta activado por defecto en centos (firewallcmd) y en Ubuntu no, aunque lo tiene (ufw), pero desactivado.

Sobre un fallo que le dio la semana pasada: Existen unos bits que hacen que un proceso pase a ejecutarse en modo superusuario, entonces con un chmod 2770 seria suficiente para que le deje hacer cosas con git. Entonces por la tarde, al no hacerlo como root, esta desactivado por defecto y no tenia permisos.

PARA CAMBIAR EL PUERTO:

Hay que modificar: /etc/ssh/sshd_config

Había una #Port 22 y hay que editar con “sed” o da fallo. Hay que quitar la almohadilla. Una vez hecho el cambio de puerto vemos que ocurre en centos y Ubuntu.

Ssh-keygen lo que hace es que si en el cliente ejecutamos esa orden, se generan una llave publica y una privada. Se utiliza un intercambio de clave para cifrar la comunicación. En el protocolo los servidores mandan la información cifrando con la publica y solo se puede descifrar con la privada que solo conocemos nosotros. Es un cifrado asimetrico por haber dos llaves diferentes.

- ALBERTO:

- o Su -> pasa a super usuario para poder modificar sshd
- o Vi /etc/ssh/sshd_config
 - Quita # del port y cambia de 22 a 22022
- o Systemctl restart sshd
- o Ssh IP -p 22022 -> para comprobarlo
- o Logout -> se sale

- SE PASA A CENTOS PORQUE TENIA YA EL PAR DE CLAVES GENERADAS:

- o Ls -la -> para ver que tiene y le sale

```
[alberto@localhost ~]$ ls -la
total 28
drwx-----Z alberto alberto 99 oct 27 07:58 .
drwxr-xr-x. 4 root root 32 oct 29 02:03 ..
-rw----- 1 alberto alberto 1485 nov 4 05:17 .bash_history
-rw-r--r--. 1 alberto alberto 18 nov 8 2019 .bash_logout
-rw-r--r--. 1 alberto alberto 141 nov 8 2019 .bash_profile
-rw-r--r--. 1 alberto alberto 312 nov 8 2019 .bashrc
-rw----- 1 alberto alberto 18 oct 27 07:58 .lesshst
```

- o Ssh-keygen -> nos pregunta si queremos ubicar la llave en algún lugar por defecto. Le puedes dar a enter pero ojo, es importante quien tenga

esa llave. Por defecto dentro de home y un directorio oculto esta bien.

Luego te pregunta una contraseña por si alguien intenta entrar a ese archivo. Lo deja en blanco dos veces y genera la llave privada

- Ls -la -> vemos el directorio .ssh
- Ls -la .ssh/ -> y vemos las dos llaves, la privada y la publica. En los permisos se nota.
- Scp -> no lo hace, es algo que se podría hacer para distribuir la publica
- Ssh-copy-id -> hace el procedimiento de forma automática y transparente y podemos especificar el puerto
- Ssh-copy-id IP -p 22022 -> le da fallo
- Sudo dhclient -> practicas, ISE
- Ssh-copy-id IP -p 22022 -> pide la contraseña del usuario que es practicas, ISE
- Ssh IP -p 22022 -> para logearse y le deja, con eso entra en Ubuntu. Así no hay que teclear la contraseña al conectarnos al servidor. Util si hay 1000 nodos y vamos a pasar mensajes. Nos vale para todos los servidores a los que nos vayamos a conectar. Si la perdemos tenemos que crear ambas de nuevo pero tenemos que borrar la publica del servidor o ese usuario si tendrá acceso.

- UBUNTU:

- Vamos a limitar el acceso por contraseña:
 - Sudo vi /etc/ssh/sshd_config
 - PasswordAuthentication no -> es lo que tenemos que descomentar y poner, pues esta a yes por defecto
 - Systemctl restart sshd
 - Practicas, ISE

- CENTOS:

- Ssh IP -p 22022
 - Puede hacerlo perfectamente, pero si lo hace desde su anfitrión (misma orden) no le deja.
- Ssh-copy-ide IP -p 22022 -> para copiar las claves. No te va a dejar. Para que te deje se va a Ubuntu de nuevo

- UBUNTU:

- Sudo vi /etc/ssh/sshd_config
 - Pone a yes el passwordauthentication por un tiempo limitado.
Se puede hacer con el “sed”

- Systemctl restart sshd

- DESDE SU ANFITRION:

- Ssh-copy-ide IP -p 22022 -> Ya si le deja

- UBUNTU:

- Sudo vi /etc/ssh/sshd_config
 - Pone a no el passwordauthentication
- Systemctl restart sshd

- DESDE SU ANFITRION:

- Ssh IP -p 22022 -> dentro de Ubuntu sin teclear contraseña
 - logout
- CENTOS:
 - Su
 - Vi etc/ssh/sshd_config
 - Ponemos el puerto 22022 y descomentado. Para modificar esto hay que comentárselo a SELinux y al meter "semanage" no lo tiene
 - Dnf provides semanage -> ve el paquete que tiene semanage que es policycoreutils o algo así
 - Dnf install policycoreutils-python-utils
 - S
 - Semanage port -l | grep ssh -> enseña los puertos que está usando ssh que es el 22
 - Semanage port -a -t ssh_port_t -p tcp 22022-> vamos a manejar un puerto y la añadimos. Es normal que tarde unos segundos.
 - Semanage port -l | grep ssh -> vemos el cambio. Ya no va a dar fallo el cambiar el puerto.
 - Systemctl restart sshd
 - Systemctl status sshd -> está en el 22022
- SU HOST:
 - Ssh IP -p 22022 -v -> da fallo porque no hay ruta al host. Esto no ha pasado con Ubuntu.
- CENTOS
 - ssh localhost -p 22022 -> para ver si das servicio a tu propia máquina y veo que sí puedo.
 - Logout
 - ¿Entonces qué hay que me impide el acceso desde fuera? Un cortafuegos. Vamos a modificarlo
- UBUNTU:
 - ufw status -> se ve desactivado
 - ufw enable -> ya lo tenemos
- SU HOST:
 - ssh IP -p 22022 -> ya no te deja
- UBUNTU:
 - ufw allow 22022
- SU HOST:
 - ssh IP -p 22022 -> ya te deja
- CENTOS:

```
firewall-cmd --add-port=443/tcp
firewall-cmd --permanent --add-port=443/tcp
```

- Con la primera línea se cierra al reiniciar, la segunda es permanente.
- Firewall-cmd --permanent --add-port=22022/tcp

- HOST:
 - o Ssh IP -p 22022 -> sigue sin acceso porque hemos añadido el puerto pero hay que recargar el cortafuegos o poner la primera línea para que en esta ejecución se tenga en cuenta.
- CENTOS:
 - o Firewall-cmd --add-port=22022/tcp
- HOST:
 - o Ssh IP -p 22022 -> ya si

Para acceder sin contraseñas haces lo del ssh-copy-id IP -p 22022 y ya se emparejan las claves y no te la pide más. Podemos poner ya todos los passwordauthentication a no.

Mirar en el guion screen, terminator y tmux (evolución de screen).

Fail2ban -> si fallas en el acceso vas a la lista de baneados. Si alguien se equivoca un numero determinado de veces, cancelan tu IP.

- CENTOS:
 - o Su dnf install epel-release
 - o Dnf search fail2ban
 - o Dnf install fail2ban
 - o Systemctl status fail2ban -> cargado pero inactivo
 - o Systemctl enable fail2ban -> en el próximo reinicio arrancara, pero esta inactivo aun
 - o Systemctl start fail2ban -> activo y habilitado
 - o Fail2ban-client status -> hay 0 carceles definidas
 - o Cd /etc/fail2ban
 - o Ls -la:

```
[root@localhost alberto]# cd /etc/fail2ban/
[root@localhost fail2ban]# ls -la
total 64
drwxr-xr-x.  6 root root   158 nov  5 10:25 .
drwxr-xr-x. 83 root root  8192 nov  5 10:25 ..
drwxr-xr-x.  2 root root  4096 nov  5 10:25 action.d
-rw-r--r--.  1 root root  2817 ago 28 07:55 fail2ban.conf
drwxr-xr-x.  2 root root    6 ago 28 07:55 fail2ban.d
drwxr-xr-x.  3 root root  4096 nov  5 10:25 filter.d
-rw-r--r--.  1 root root 25740 ago 28 07:55 jail.conf
drwxr-xr-x.  2 root root    31 nov  5 10:25 jail.d
-rw-r--r--.  1 root root  2827 ago 28 07:55 paths-common.conf
-rw-r--r--.  1 root root   930 ago 28 07:55 paths-fedora.conf
[root@localhost fail2ban]#
```

- o Less jail.conf -> nos dice que no modifiquemos el archivo. Lo vamos a copiar en el local para conservar los fallos si algo se restaura.
- o Cp -a jail.conf jail.local
- o Vi jail.local -> vamos a sshd y en enabled ponemos true
- o :wq
- o Vi /etc/ssh/sshd_config -> temporalmente ponemos a yes el passwordauthentication

- Systemctl restart sshd
- UBUNTU:
 - Ssh IP -p 22022 -> fallamos la contraseña unas cuantas de veces
- CENTOS:
 - Systemctl restart jail2ban.service
 - Fail2ban-client status -> vemos 1 ya y si desde Ubuntu intentamos entrar, no podemos
 - Fail2ban-client status sshd -> vemos la ip baneada.
- UBUNTU:
 - Ssh IP -p 22022 -> nos sigue dejando
- CENTOS:
 - vi jail.local -> tiene un parámetro para el puerto que vamos a cambiar porque por defecto esta el 22. Buscamos la parte de sshd y en port ponemos 22022
 - systemctl restart fail2ban.service
 - fail2ban-client status sshd
 - iptables -L -> rechaza todos los paquetes a Ubuntu server y con este comando se ve.

Rkhunter-> es un software malicioso que intenta esconderse (no es un virus por la replicacion) pero puede filtrar contraseñas. Con RootKit Hunter, semanalmente, podemos estar seguros de ver que no hay rkhunter.

AHORA VAMOS A CONFIGURAR EL SERVICIO LAMP

HTTP -> Hyper Text Transfer Protocol

HTML -> Hyper Text MARK-UP Language

Los hipertextos se escriben usando el lenguaje de markups html. En el tenemos el texto, el texto especial entre angulo para marcar en negrita, un enlace, titulo, etc.

Tenemos un servidor escuchando peticiones en el puerto 80, le llega una petición y quiere un documento html, el servidor lo devuelve pero es que los html son estáticos y no se pueden cambiar. Así que si quisieramos tener un documento con comentarios de los servidores podríamos tener el formulario como tal pero no podría subirse al servidor a no ser que tengamos un esquema mas complejo. Este dinamismo se añade con una lógica que compruebe si el usuario puede escribir en el documento y una base de datos. Por lo que tendremos una lógica y una BD. Podemos meter lógicas en el html con script y usaríamos javascript, pero este se ejecuta en el cliente. Por temas de seguridad con las contraseñas, es obligatorio una lógica en el servidor y un esquema que me permita almacenar datos. Html no tiene estado. Conocemos el sistema de gestión de bases de datos MySQL y MariaDB. Si quieras montar una web, que conjunto de programas podremos tener: un servidor con el puerto 80 que devuelva html

(apache o nginx por ejemplo) y te sirven PHP, Python como lógicas y MySQL como BD. Si juntas todo en una arquitectura Linux, tenemos la pila software LAMP (WAMP o XAMP para mac).

- UBUNTU:
 - o Sudo tasksel
 - Marcamos LAMP server
 - o Man curl -> para comprobar
 - o Curl localhost
 - o Mete la IP dentro del buscador a ver pero se cuelga. Es porque el cortafuegos esta activo
 - o Sudo ufw status -> lo vemos
 - o Sudo ufw allow 80 -> ahora si, vamos de nuevo al buscador y nos debería Salir ya la pagina de apache
 - o Php -a -> para ver que esta funcionando
 - o Sudo mysql -> funcionando tambien
- CENTOS (misma instalación a mano):
 - o Dnf search apache -> vemos:

```
===== Coincidencia en Resumen: apache =====
httpd.x86_64 : Apache HTTP Server
```

- o Dnf install httpd
 - S
- o Systemctl status httpd -> inactivo y disabled
- o Systemctl enable httpd -> activao papaaaa
- o Systemct start httpd
- o Systemctl status httpd
- o Curl localhost -> nos devuelve la pagina
- o Dnf search php :

```
===== Coincidencia en Nombre , Resumen: php =====
php.x86_64 : PHP scripting language for creating dynamic web sites
```

- o Dnf install php
 - S
- o Dnf search mysql

```
===== Coincidencia en Nombre , Resumen: mysql =====
mysql.x86_64 : MySQL client programs and shared Libraries
MySQLdb-x86_64 : MySQL backup manager
```

- o Dnf install mariadb-server
 - S
- o Systemctl status mariadb -> inactivo y disabled
- o Systemctl enable mariadb
- o Systemct start mariadb
- o Mysql -> accedemos bien
- o Mysql_secure_installation

- Te pide contraseña pero es ninguna, peligroso. Le ponemos practicas,ISE a root. Tambien tenemos un usuario anonimo pero no nos interesa y lo quitamos. No queremos que root se conecte desde una maquina de fuera. Borramos tambien la base de datos de test. Si cargamos la tabla de privilegios.
- Mysql -> ya no nos deja
- Mysql -u root -p
 - Practicas,ISE
- Firewall-cmd --add-service=http --permanent
- Firewall-cmd --reload -> ya podemos desde el navegador meter la ip
- Dnf install php-mysqlnd.x86_64
 - Si

EJEMPLO DE PAGINA:

SU HOST:

- Touch /var/www/html/miscript.php
- Vi /var/www/html/miscript.php
 - Ctrl + shit + v
 - Modificamos "my_user" = root, "my_password" = practicas,ISE "my_db" = db
- Curl localhost
- Mete en un buscador la IP
- Firewall-cmd --add-service=http --permanent
- Firewall-cmd --reload
- Vuelve a meter la IP y ya si sale
- Busca en el buscador: IP/miscript.php -> no sale bien
- Vi /etc/httpd/conf/httpd.conf
 - Buscamos directoryindex y lo ponemos asi

```
# DirectoryIndex: sets the file that Apache will serve if a directory
# is requested.
#
<IfModule dir_module>
    DirectoryIndex index.html *.php
</IfModule>
```

- :wq
- Systemctl restart httpd
- Vuelve al buscador y sigue dando fallo. Mira los fallos dando a f12 y dando a consola. Le pone http500 es un fallo del servidor.
- Para ver el error "less /var/log/httpd/error_log" y no ve nada, se va a "less /var/log/httpd/access_log" y vemos los fallos 500. Ahora va a "less /var/log/php-fpm/www-error.log"
- Dnf install php-mysqlnd.x86_64

- S
 - Otra forma de comprobar el error: cd /var/www/html y luego php miscript.php. Nos metemos en mysql -u root -p y ponemos CREATE DATABASE db;
 - Vuelve al buscador, y ooooooooooooo fallo.
 - Less /var/log/Audit/Audit.log -> pone que no da permiso a php-fpm
 - Getsebool -a -> tenemos booleanos que establecen políticas y restricciones.
 - Getsebool -a | grep http -> y hay uno que es httpd_can_network_connect_db
 - Setsebool httpd_can_network_connect_db=1
 - FUNCIONA SI TE VAS AL BUSCADOR

Cogeremos este texto:

```
<?php
$enlace = mysqli_connect("127.0.0.1", "mi_usuario", "mi_contraseña"
, "mi_bd");

if (!$enlace) {
    echo "Error: No se pudo conectar a MySQL." . PHP_EOL;
    echo "errno de depuración: " . mysqli_connect_errno() . PHP_EOL
;
    echo "error de depuración: " . mysqli_connect_error() . PHP_EOL
;
    exit;
}

echo "Éxito: Se realizó una conexión apropiada a MySQL! La base de
datos mi_bd es genial." . PHP_EOL;
echo "Información del host: " . mysqli_get_host_info($enlace) . PHP
_EOL;

mysqli_close($enlace);
?>
```

SCREEN, TERMINATOR Y TMUX

Terminator -> permite varias configuraciones, ponerlo mas grande, diferentes colores y es útil para cuando trabajamos con varios servidores. Eso mismo en modo texto seria tmux, que es una evolución de screen. En ssh hay un parámetro que es keepalive pero tras mucha inactividad, la conexión se cierra. Si se queda a medio hacer, al hacer un "ps xf" vemos la rama del proceso sshd que entra por un bash y tal, el resto de hijos, en cascada, al cerrar la conexión y terminar el proceso de golpe, se cerraran también. Se puede evitar poniendo screen o tmux para crear un proceso que adoptara a los procesos hijo y se quedan colgados del tmux que queda residente.

Sudo dnf install screen -> para instalarlo. Ahora hace un vi de lo que sea, por ejemplo, vi asdf y lo deja en background, poniendo ps xf vemos que cuelga vi también entre los procesos. Entonces hacemos screen y luego ps xf para ver que si que cuelga de screen ya. Ahora vi firfjif y lo dejas en background, hacemos ps xf y vi y ps cuelgan de screen. El anterior vi si que cuelga de bash. Entonces si muere el proceso, si hace screen -r (detrás de la orden se puede especificar)y se puede listar las sesiones con screen -list

ISE Práctica 2

Lección 1

En centos8 se usa yum pero es en realidad dnf y en centos7 directamente se usa yum.

Los paquetes snaps, a parte de homogeneizar los paquetes de software, añadían un control de dependencias para que no hubiese ningún error al respecto.

Secure Shell -> SSH

SSHD para hablar del servicio, (la d viene de daemon)

SSH para hablar del cliente.

Siendo tu propia máquina el servidor y las MVs los clientes, si haces **ssh <ip>** te conectarás a la máquina de la ip dada y en ella en el archivo:

/home/ssh/known_hosts verás la ip y la fingerprint que ha dejado.

En Ubuntu podemos instalar ssh con **apt install openssh-server** o con tasksel, pero este puede dar problemas. Si usamos el apt install además podemos saber qué versión estamos instalando.

Hay que configurar ssh en Centos porque tiene el PermitRootLogin a yes. Ubuntu lo tiene desactivado por defecto.

Los archivos configuración de los servicios están en /etc y con **vi /etc/ssh/sshd_config** podemos editar el de ssh.

Los parámetros de los archivos de configuración se ejecutan cuando se leen al inicio por lo que para que el cambio sea efectivo, tienes que reiniciar el servicio. Esto se hace con **systemctl restart sshd**.

Por motivos de seguridad, vamos a cambiar el puerto al que escucha ssh que por defecto es el 22.

Para ello podemos usar **sed s/'Port 22'/'Port 22322'/ -i /etc/ssh/sshd_config**.

(Hemos puesto 22322 por poner algo, supongo).

Reiniciamos el servicio.

La línea alterada estaba comentada así no funcionará hasta que la descomentemos, atento a esas cosas.

Tampoco funcionará pero si te metes en el archivo de configuración de sshd verás que hay una nota que dice que para cambiar el puerto al que escucha ssh tienes que notificar a SELinux de este cambio, de otra manera no habrá permisos.

Para esto usamos **semanage port -a -t ssh_port_t -p tcp <Número de puerto>**.

Y con esto debería funcionar el cambio de puerto.

Lección 2

Un backup es una copia de los datos añadiendo una información extra (la versión por ejemplo.)

dd hace una copia bit a bit del contenido.

dd if=/dev/sda of=/dev/sdb

if es input file y of output file

Lo malo es que los metadatos los tenemos que poner nosotros.

cpio coge una serie de archivos y los pone en un único archivo especial.

“Empaque los archivos”. (Pero no los comprime).

ls | cpio -ov > /tmp/object.cpio

El proceso de backup puede mejorar teniendo o bien compresión para que la velocidad de transferencia sea mejor, tener un búfer intermedio entre tu máquina y la nube y usar backup incremental que solo envíe los cambios hechos para que la transferencia sea menor. Además es importante tener el servidor en España para que se acoja a la LOPD y en Europa para que se acoja a la GDPR si es que la información es sensible.

tar es el sucesor de cpio pero siendo más cómodo al poder indicar directorios (La sintaxis de cpio es un poco rara)

tar cvzf MyImages-14-09-17.tar.gz /home/MyImages

Al principio no comprimía pero luego se lo añadieron (de ahí la extensión gz).

rsync sincroniza una fuente y un destino. (Se puede usar a través de ssh)

- Copia enlaces, dispositivos, propietarios, grupos y permisos
 - Permite excluir archivos
 - Transfiere los bloques modificados de un archivo
 - Puede agrupar todos los cambios de todos los archivos en un único archivo
 - Puede borrar archivos
- Ejemplos (<https://rsync.samba.org/examples.html>)
- rsync -a --delete source/ destination/

Hay que aprenderse los comandos de uso común de rsync. Va a caer en el examen. Los errores que hay en las diapositivas también podrían caer.

Control de cambios:

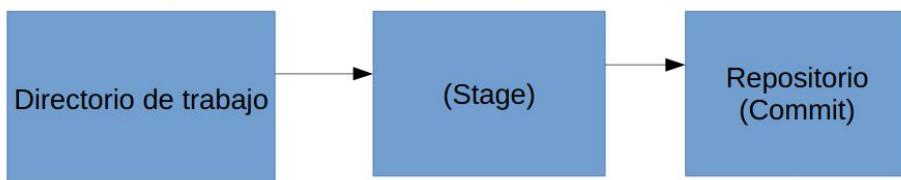
El método más cutre es copiar un archivo con otro nombre antes de modificar nada.
Es más apropiado usar **git**.

Git:

En sus ventajas para ISE podemos hacer un seguimiento de los cambios (con su autor), permite planificar un entorno de pruebas entre varias personas.
Desarrollado por Linus que es un señor finlandés.

La filosofía es la siguiente:

- Directorio de trabajo: lo que tenemos en desarrollo
- Stage: Zona donde se registran los cambios. Ahí incluimos los cambios que queremos seguir.
- Commit: Zona donde los cambios se convierten en permanentes (podemos hacer commits en diferentes ramas “branches” y ponerles etiquetas “tags”)
- La evolución del trabajo puede mostrarse como un grafo dirigido acíclico (DAG)



Cuando se hace commit se genera un hash (SHA-1 de 40 caract) que lo identifica pero normalmente con los 7 primeros lo puedes diferenciar.

El último commit tiene un puntero hacia él que se llama HEAD.

Con **git reflog** podemos ver un log del desplazamientos por los commits.

Los .. hacen referencia a intervalos lineales de commits y los ... a no lineales.

Iniciando un repositorio:

Instalamos git. **sudo dnf install git**.

Hacemos una carpeta aparte **mkdir /git**.

Iniciamos el repositorio con **git init repo1 --bare**.

La diferencia entre un repositorio “bare” y uno normal es que el bare no tiene directorio de trabajo.

Creamos tanto en la MV como en el anfitrión el usuario ana (por ejemplo).

En el anfitrión, desde el user david hacemos

sudo git clone 192.168.56.110:/git/repo1.

Para clonar el repositorio.

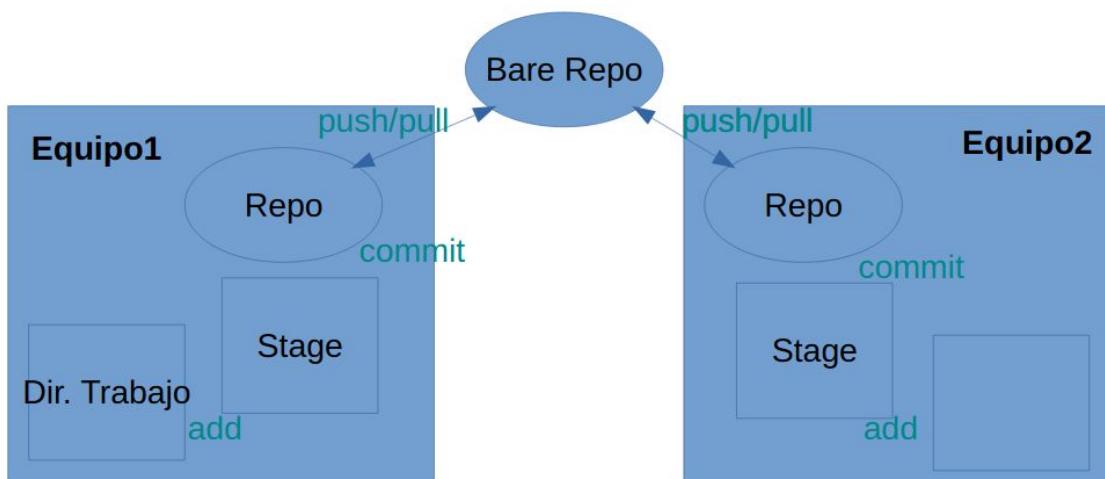
Luego también lo hacemos desde ana.

Con **git status** obtenemos información del repositorio.

Hacemos **git add README.md** desde repo1.

Y luego su commit con **git commit -m “ hemos creado el archivo README.md”**.

Ahora tendríamos el archivo en “Repo” (antes estaba en “Stage” pero hemos hecho commit) en uno de los equipos y queremos pasarlo a Bare Repo.



Ahora con **git push** llevamos el archivo a Bare Repo (en CentOS no lo veremos aún así porque es de tipo Bare, no hay directorio de trabajo o sea que el repositorio solo puede pasar información, no se puede hacer nada con los archivos desde él).

Ahora desde ana hacemos **git pull** para pasar el archivo de Bare Repo a Repo.

Con **git diff** podemos ver las diferencias entre el directorio de trabajo y el área de seguimiento (el Stage).

Una vez hecho **git add**, no habrá diferencias.

Con **git diff HEAD** podrás ver las diferencias entre el Repo y el Stage (hasta que hagas un commit).

git pull = git fetch; git merge.

Lección 3

Ahora vamos a crear una llave privada con **ssh-keygen** (yo lo estoy haciendo en el home de CentOS8).

Esto creará en .ssh una llave .pub pública y una privada. Con **ssh-copy-id <ip> -p <puerto>** podemos pasarnos la llave pública a la otra máquina y hacemos ssh en esa máquina para probar que la tenemos.

Después en ubuntu editamos el /etc/ssh/sshd_config y descomentamos la línea de **PasswordAuthentication yes** y la ponemos en **no**.

Entonces ahora desde CentOS podremos hacer ssh sin tener que usar contraseña (porque ahí teníamos nuestra key privada) y desde el anfitrión no podemos hacer ssh porque hemos deshabilitado la autenticación por contraseña.

Vamos a poner **PasswordAuthentication** en yes otra vez.

Usamos **ssh-copy-id <ip> -p <puerto>** en el anfitrión para tener también la clave y lo volvemos a poner en no.

Ahora en CentOS cambiamos el puerto (habrá que usar semanage **port -a -t ssh_port_t -p tcp 22022**).

Si intentamos hacer ssh a CentOS nos dirá que no hay ruta. Hay que modificar el cortafuegos.

Ubuntu también tiene un cortafuegos pero es el Uncomplicated Fire Wall (UFW) y está desactivado por defecto.

Vamos a activar el de Ubuntu y vamos a permitir que solo permita del puerto 22022. Esto se hace con **ufw enable** y **ufw allow 22022**.

Esto en CentOS se hace con **firewall-cmd --permanent --add-port=22022/tcp**.

También habría que hacer **firewall-cmd --reload** para refrescar la nueva configuración.

Con esto ya habremos configurado SSH.

screen hace que los procesos se queden en una “sesión” aparte (usando **screen**) para que en caso de pérdida de conexión, podamos recuperarlos con **screen -r**. terminator permite tener varias configuraciones en la terminal así como dividir una ventana de terminal en dos verticales, horizontales, cambiar de perfil...etc tmux hace lo mismo que screen.

Qué son, para qué sirven y cómo son preguntas de examen.

Vamos a instalar ahora fail2ban.

Desde el anfitrión hemos hecho ssh a CentOS y allí hemos usado **dnf search fail2ban** y luego hemos usado **dnf install fail2ban** para instalarlo.

Hacemos **systemctl enable fail2ban** y **systemctl status fail2ban** para comprobar que se ha activado para la próxima vez que iniciemos.

Y para iniciarla ahora **systemctl start fail2ban**.

Con **fail2ban-client status** veremos que hay 0 jaulas definidas. Vamos a etc/fail2ban para ver los archivos de configuración.

Hay un archivo jail.conf que no se debe editar porque una actualización del servicio borraría las ediciones. Tienes que copiarlo a .local con **cp -a jail.conf jail.local**.

Ahora podemos editarlo con **vi/jail.local**.

Hay que irse al apartado de cárcel de sshd y añadir **enabled = true**.

Cambiamos el puerto porque por defecto es 22 en el mismo apartado y reiniciamos el servicio con **systemctl restart fail2ban.service**.

Podríamos desbanearla con **fail2ban-client set sshd unban ip <ip a desbanear>**.

RootKit es un software que analiza el sistema en busca software malicioso en la máquina (los más conocidos).

HTTP -> Hyper TextTransfer Protocol

HTML -> Hyper TextMarkup Language

Los ficheros HTML son estáticos, para añadir dinamismo hay que añadir una lógica en la parte del servidor y un esquema que permita almacenar datos.

Este esquema para almacenar datos puede ser SQL.

De SQL salieron MySQL que lo compró Oracle y la parte libre que es MariaDB.

Estos tienen la misma sintaxis.

Para hacer una página web vamos a necesitar un servidor que escuche por el puerto 80 y que devuelva documentos HTML (Apache por ejemplo).

Entonces tenemos la pila LAMP Linux Apache MySQL Python/Pearl.

Primero instalamos apache. En ubuntu hemos usado tasksel que no es recomendable y en CentOS lo vamos a hacer a mano.

Con **dnf install httpd**.

Después para hacer que esté activo cuando se inicie la máquina se usa **systemctl enable httpd** y para activarlo ahora **systemctl start httpd**.

Instalamos php con **dnf install php**. Y ahora usamos **dnf install mariadb** y **dnf install mariadb-server** para instalar el cliente y la base de datos respectivamente.

Los activamos con enable y start como siempre.

Usamos **mysql_secure_installation** y nos pedirá una contraseña para el root de la BD.

Borramos el usuario anónimo y no queremos que root se conecte remotamente.

Y remove test database. Ya tendremos Maria configurado.

Hacemos **touch /var/www/html/miscript.php**.

En /html apache buscará los scripts de forma predeterminada. Aquí copiamos el código que se ve en clase (el profesor lo saca de internet).

Dejamos esta línea así:

```
<?php  
$enlace = mysqli_connect("127.0.0.1", "root", "practicas,ISE", "db");
```

Hacemos **vi /etc/httpd/conf/httpd.conf** y en la parte donde se listan los archivos que Apache sirve si se pide un directorio, añadimos *.php.

Si buscamos la ip en internet no encontraremos nada, si buscamos la <ip>/miscript.php e inspeccionamos elemento veremos un error interno del servidor. En **/var/log/php-fpm/www-error.log** veremos el error devuelto por el archivo php. El error se debe a que tenemos instalado el intérprete de php pero no su biblioteca que lo conecta con mysql. La instalamos con **dnf install php-mysqlnd.x86_64**. También podríamos haber detectado el error yendo a la localización del script y haciendo **php miscript.php**.

Ahora en MariaDB creamos una base de datos con **CREATE DATABASE db;** Y usamos otra vez php en el script.

En internet tendremos otro fallo:

```
Error: No se pudo conectar a MySQL. errno de depuración: 2002 error de depuración: Permission denied
```

Esto se debe a SELinux.

Con **getsebool -a | grep http** vemos los valores booleanos referentes a http.

Y vemos que:

```
httpd_can_network_connect_db --> off
```

Lo ponemos a true con:

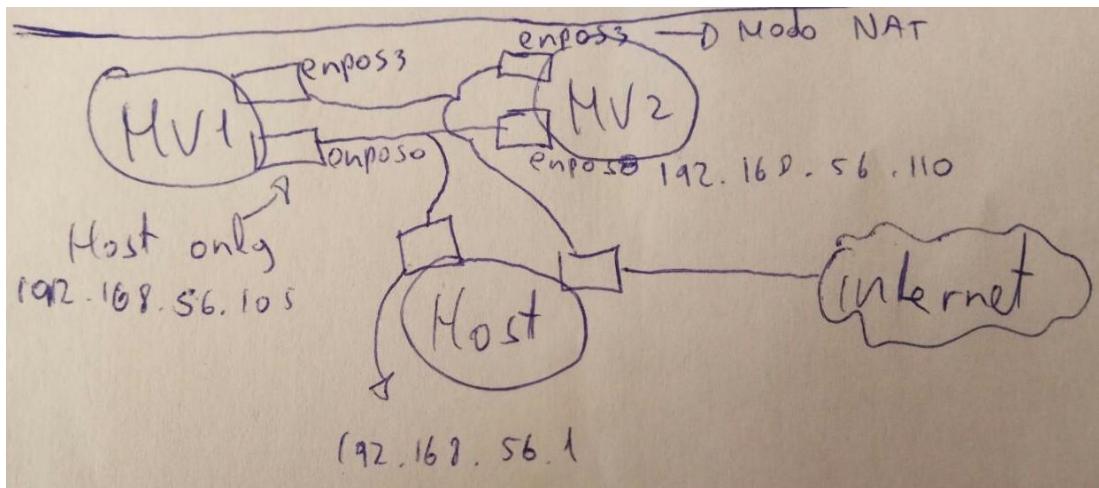
```
setsebool httpd_can_network_connect_db=1.
```

Y finalmente, en internet:

```
Éxito: Se realizó una conexión apropiada a MySQL! La base de datos mi_bd es genial. Información del host: 127.0.0.1 via TCP/IP
```

Prerrequisitos:

Para realizar esta práctica, necesitamos que las maquinas tengan esta configuración de red:



Mv1 → Ubuntu

Mv2 → centOS

Como crear la estructura de red:

Para crear una red interna en virtual box debemos ir a archivo → preferencias → red.

Aquí creamos una red con configuración 192.168.56.0/24.

Después de esto, en cada máquina virtual debemos activar la segunda interfaz de red y seleccionarla en modo solo-anfitrión (host-only en inglés).

Esto se consigue en configuración → red → adaptador 2.

Para modificar la configura de red de las maquinas

UBUNTU

Modificamos el archivo /etc/network/interfaces con: "sudo nano /etc/network/interfaces" y añadimos al final

```
iface enp0s8 inet static  
address 192.168.56.105  
netmask 255.255.255.0  
gateway 192.168.56.1
```

Para reiniciar la red:

/etc/init.d/networking restart

CENTOS

Vi /etc/sysconfig/network-scripts/ifcfg-enp0s8 (si no existe copiamos el ifcfg-enp0s3 con el nuevo nombre).

Borramos todas las líneas menos:

type
bootproto
name
device
onboot

Y añadimos

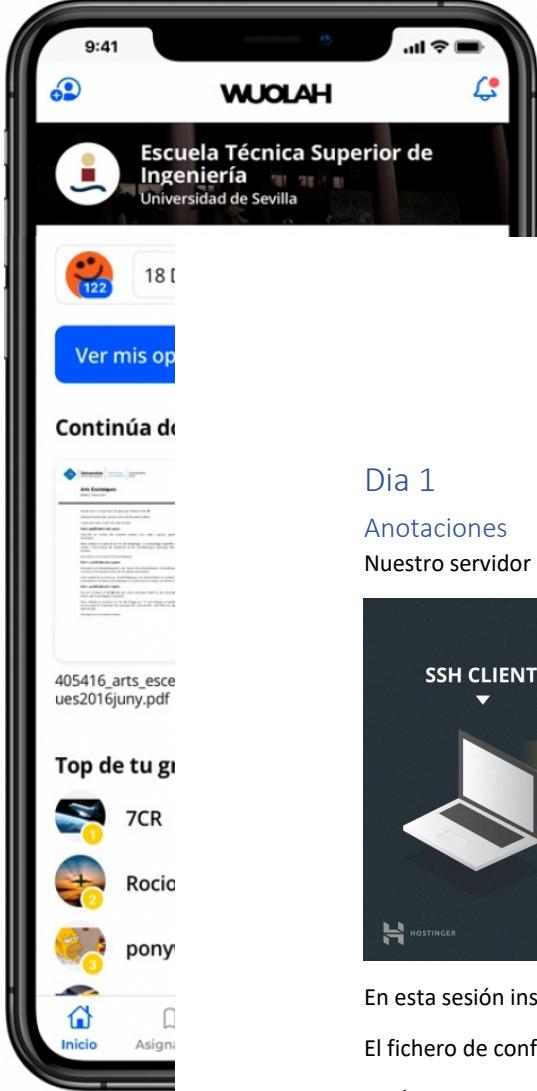
ipaddr=192.168.56.110

además, si el atributo onboot está en OFF, ponerlo en ON (VERIFICAR EN EL FICHERO /etc/sysconfig/network-scripts/ifcfg-enp0s3 QUE ESTE EN ON TAMBIEN ESTE ATRIBUTO).

para reiniciar la red:

Tiramos la interfaz con: ifdown enp0s8

La levantamos con: ifup ifdown enp0s8



Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.

Available on the App Store

GET IT ON Google Play

Dia 1

Anotaciones

Nuestro servidor será Ubuntu y nuestro cliente será centOS.



En esta sesión instalaremos ssh y configuraremos algunos parámetros.

El fichero de configuración de ssh es: /etc/ssh/sshd_config.

Parámetros:

- **port**: <int> identifica el puerto de conexión.
- **PermitRootLogin** <Prohibit-password, no> permite conectarse con el usuario root.
- **passwordAuthentication** <yes,no> permite conectarse con usuario y contraseña.

Comandos

- **ps -xf**: muestra todos los procesos en ejecución.
 - **x**: procesos que no están siendo ejecutados por ninguna terminal (también).
 - **f**: formato "forest" ("bosque") de familias en forma de árbol.
- **ssh <ipservidor> -l <usuario> -v**: conecta con un servidor remoto a través de ssh.
 - **l**: usuario.
 - **v**: verbose (muestra información extra).
- **apt search <cadena>**: busca los paquetes que se pueden instalar con apt.
- **apt install <paquete>**: instala el paquete indicado.
- **tasksel**: abre la interfaz de instalación de servicios que lanza Ubuntu durante su instalación.
- **systemctl status ssh.service**: comprueba el estado del servicio ssh.
- **systemctl start ssh.service**: inicia el servicio ssh.
- **systemctl stop ssh.service**: para el servicio ssh.
- **ssh-keygen**: genera un par de claves pública y privada.
- **ssh-copy-id <usuario>@<ipservidor> -p <puerto>**: copia la clave publica en el servidor destino. (necesario usuario y contraseña a no ser que ya haya sido configurado para acceder con llaves previamente).

Instalación y pruebas

En el servidor:

Comprobamos que ssh no está instalado (ni ejecutado en la maquina) con las siguientes ordenes:

Usamos el comando para listar los procesos que están en ejecución: “*Ps -xf*”.

Filtramos los procesos ssh con: “*ps -xf | grep ssh*”.

Para buscar el nombre de los paquetes ssh servidor utilizamos:

Sudo Apt search ssh | grep server.

Con este comando instalamos ssh: “*Apt install openssh-server*”.

“*Sudo tasksel*” abre la terminal de instalación de canonical (otra opción para instalar ssh).

“*Vi /etc/ssh/sshd_config*”.

Cambiamos el puerto a 22022 (parámetro **Port**).

Reiniciamos el servicio con: “*systemctl restart ssh.service*”.

Comprobamos el estado del servicio con: “*systemctl status ssh.service*”.

Comprobamos que la conexión se establece con éxito.

“*Ssh <usuario>@localhost -p 22022 -v*”.

Modificamos el fichero de nuevo para denegarle el acceso a root por motivos de seguridad con:

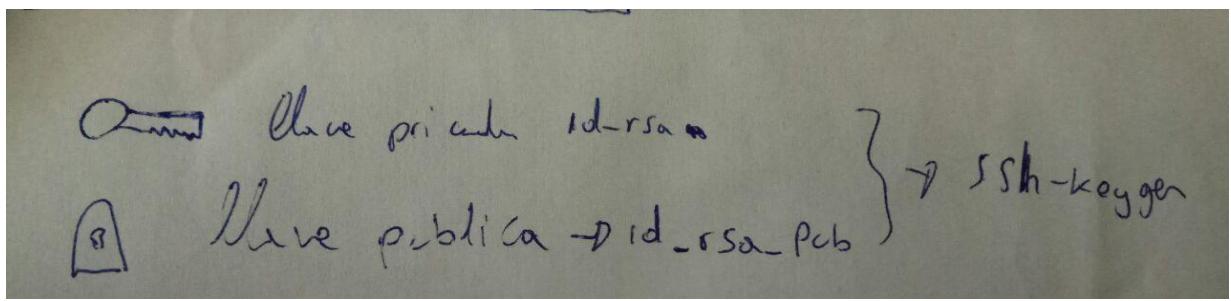
“*Vi /etc/ssh/sshd_config*”.

PermitRootLogin Prohibit-password → PermitRootLogin no.

Reiniciamos el servicio con: “*systemctl restart ssh.service*”.

Comprobamos el estado del servicio con: “*systemctl status ssh.service*”.

En el cliente:



Comprobamos que ahora no nos deja con el usuario root con:

"Ssh 192.168.56.105 -p 22022 -l root -v" (no nos deja).

Generamos las dos llaves (publica y privada) con: "Ssh-keygen".

Copiamos la clave publica al servidor con: "Ssh-copy-id <usuario>@192.168.56.105 -p 22022".

(pregunta la contraseña del usuario).

Para comprobar que la llave se copió correctamente y que funciona, intentamos conectar usándola con:

"ssh -p 22022 <usuario>@192.168.56.105".

(no debe pedir contraseña).

En servidor:

Modificamos el servicio para que ahora no deje entrar con usuario y contraseña a nadie (solo podrá acceder el que tenga la llave privada).

"Vi /etc/ssh/sshd_config".

passwordAuthentication no (decomentamos y lo ponemos a no).

Reiniciamos el servicio con: "systemctl restart ssh.service".

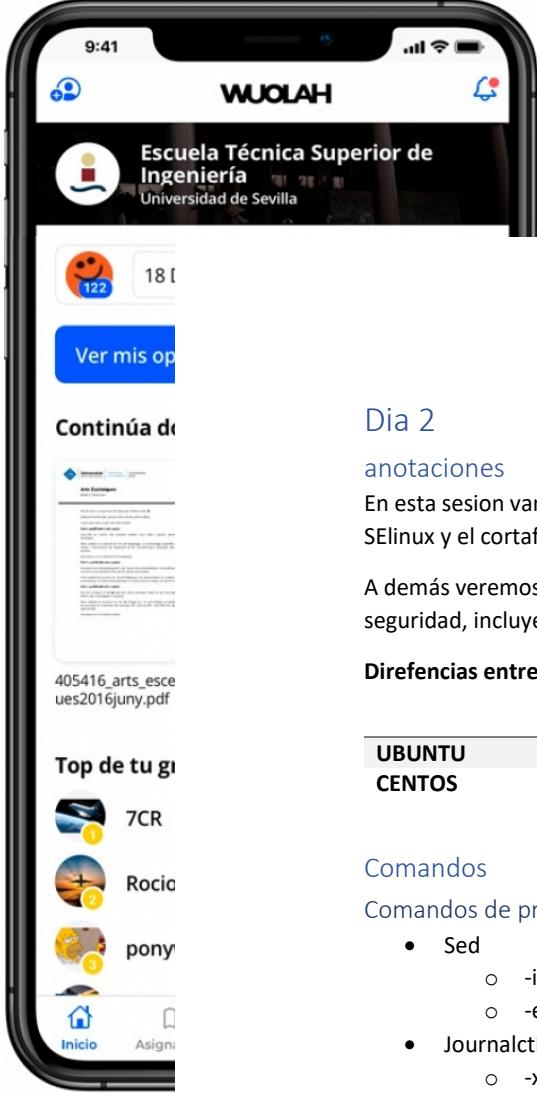
Comprobamos el estado del servicio con: "systemctl status ssh.service".

En el cliente:

Para comprobar que esta última modificación se ha hecho efectiva bastaría con intentar acceder con otro usuario que no sea en el que se copió la clave pública y/o desde otra máquina cliente:

"ssh -p 22022 <usuario2>@192.168.56.105".

Una vez aquí, podremos acceder desde la maquina que tenga la clave privada, pero desde ninguna más, aunque conozcamos los usuarios y contraseñas.



Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.

Available on the App Store

GET IT ON Google Play

Continúa d

Dia 2

anotaciones

En esta sesión vamos a instalar ssh en centOs y vamos a lidiar con una serie de problemas con SELinux y el cortafuegos.

A demás veremos que son, para qué se usan y cómo hacer un buen uso de las copias de seguridad, incluyendo control de versiones (git).

Diferencias entre centOS y ubuntu

	INSTALADO	NOMBRE	ARCHIVO CONF	FIREWALL ACTIVO
UBUNTU	No	Ssh/sshd	Sin comentarios	No
CENTOS	Si	Sshd	Con comentarios	si

Comandos

Comandos de práctica:

- Sed
 - -i modifica una línea
 - -e sustituye una cadena por otra
- Journalctl (muestra info de errores del sistema)
 - -x muestra explicación de los errores producidos
 - -f follow, hace un seguimiento de los errores en el tiempo
- Yum provides
- Semanage
 - -l muestra una lista
 - -a añadir puerto nuevo
 - -p puerto
 - -t tipo de puerto
- Ufw status
- Ufw enable
- Ufw allow
- Firewall-cmd
 - --add-port=
 - --permanente
 - --reload

Comandos de copias de seguridad:

- Dd (copia a nivel de bytes)
- Cpio (lista los ficheros de un directorio)
- Tar (comprime y descomprime)
 - c crear archivo
 - v verbose
 - z para comprimir
 - f archivo que creamos
 - x extract

- Rsync (sincroniza (dropbox antiguo))
 - -a guarda metadatos, permisos, privilegios...
 - -v verbose
 - -i informe final
 - -z comprime
 - --delete
- Rsnaptshot (crea un pantallazo del sistema)
- Programas de pago:
 - AMANDA
 - Bacula
 - C-Panel
 - Plesk

Comandos de control de versiones:

- git init (inicia el repositorio en la carpeta en la que estamos)
- git log (visualiza el log)
- git status (muestra el estado de git)
- git config (configura el usuario de git)
 - --global user.name <usuario>
 - --global user.email <correo>
- git add <nombre archivo> (añade un archivo al commit)
- commit (envia el commit y lo aplica)
 - -m (mensaje)
 - --ammend (modifica el mensaje)
 - -ammend -a (añade un archivo nuevo)
- Diff (muestra las diferencias entre el work directory y el staging area)
 - --taged (entre staging área y local repository)
 - HEAD (entre el working directory y local repository)
- checkout HEAD <nomb_archivo> (recupera el archivo que hay en el último commit)
- reset <7_primeros_caracteres_del_commit_SHA> (recupera el commit especificado)
- git revert <7_primeros_caracteres_del_commit_SHA> (recupera el commit y crea uno nuevo)
 - -n (evitamos que cree un commit nuevo)
- branch (nos muestra en qué rama estamos (master por defecto))
- checkout <nombre_rama> (para cambiar la rama que usamos)
- branch <nombre_rama> (crea una rama)
- checkout -b <nombre_rama> (crea una rama nueva y se une a ella)
- merge (une ramas)
- git init –bare (crea un repositorio en linea)
- git clone (clonamos el repositorio)
- git add remote (añadimos un origen remoto)
- git push (enviamos los commits al origen remoto)
- git pull (nos traemos los cambios del repositorio y unimos con los del directorio de trabajo actual)

Instalación y pruebas

En centOS:

Otra forma de modificar un archivo de configuración es con el comando sed.

Necesitamos modificar el puerto de ssh para ello usando sed reemplazamos la cadena que determina el puerto en el archivo /etc/ssh/sshd_config con:

```
'Sed -e "s/#Port 22/Port 22022/" -i /etc/ssh/sshd_config'
```

Reiniciamos el servicio con:

"*Systemctl restart sshd*"

"*Systemctl status sshd*"

Vemos que tiene error y que no se ha levantado el servicio.

Ahora vamos a ver el log de errores a ver que ha pasado con este comando:

"*Journalctl -xf*"

Vemos que el error se produce porque el cambio de puerto no se puede hacer efectivo.

En el archivo de configuración de ssh sale que debemos notificar a SELinux sobre el cambio de puerto, sino no funcionara.

Para ello instalamos el paquete semanage con:

"*Yum install semanage*"

Da también error porque le hacen falta paquetes que no tiene. para saber que comando usar:

"*Yum provides semanage*"

Nos dice que usemos el comando:

"*Yum install policycoreutils-python*"

Una vez instalado seguimos con lo que estábamos.

Para listar los puertos que están usando el sistema:

"*Semanage port -l*"

Para buscar el puerto de ssh:

"*Semanage port -l | grep ssh*"

Una vez vemos que el puerto de ssh es el 22, podemos **añadir otro puerto permitido** con el siguiente comando:

"Semange port -a -p tcp -t ssh_port_t 22022"

hecho esto ya nos dejara **levantar el servicio ssh** con:

"Systemctl start sshd"

añadimos un usuario con:

"useradd -m usuario"

Le ponemos contraseña con:

"passwd usuario"

Modificación del cortafuegos:

En ubuntu:

Probamos que ssh 192.168.56.101 -p 22022

Pero no funciona porque el firewall corta la conexión.

En Ubuntu para modificar el cortafuegos:

Ufw status

Ufw enable

Ufw allow 22022

En centos:

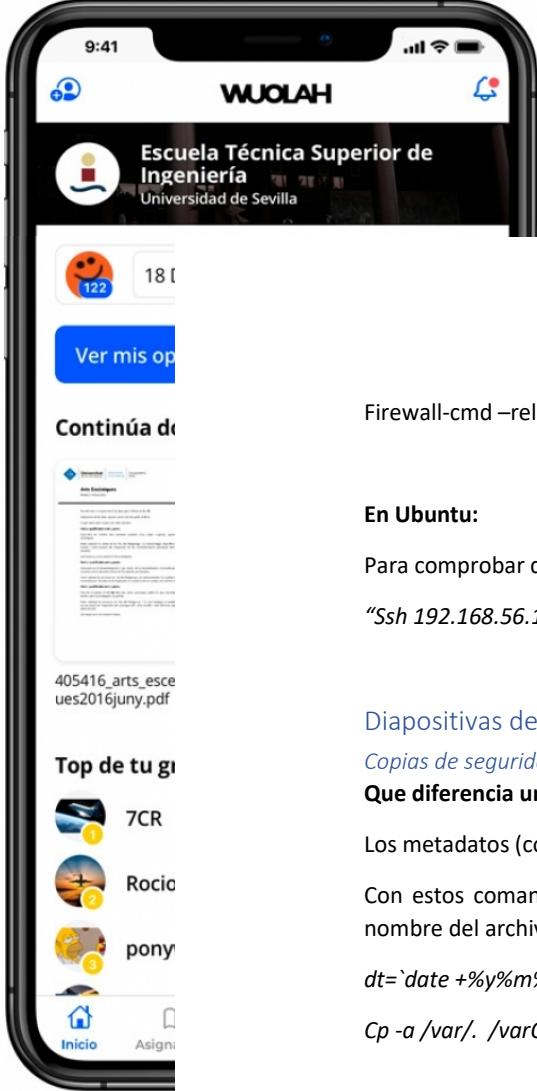
Firewall-cmd --add-port=22022/tcp

Funciona. Si reinicias ya no funciona, hay que ejecutarlo de nuevo

Para hacerlo permanente con:

Firewall-cmd --permanent --add-port=22022/tcp

Para reiniciar el cortafuegos con:



Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.

Available on the
App Store

GET IT ON
Google Play

Firewall-cmd –reload

En Ubuntu:

Para comprobar que todo esta bien configurado, conectar a ssh con:

`"Ssh 192.168.56.110 -p 22022"`

[Diapositivas de copias de seguridad y control de versiones](#)

[Copias de seguridad](#)

Que diferencia una copia de archivos de una copia de seguridad??

Los metadatos (como la fecha)

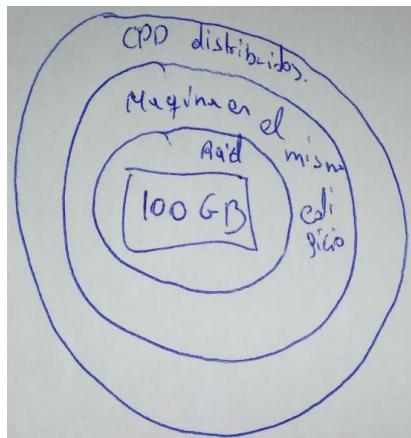
Con estos comandos podemos crear una copia de seguridad añadiendo la fecha al final del nombre del archivo:

`dt=`date +%y%m%d``

`Cp -a /var/. /varOLD-$dt`

Estaría bien cifrarlo y distribuirlo en distintos CPDs

La mejor forma es la seguridad en anillos.



Hay varios tipos de copias de seguridad:

Total: hace copia de todo el sistema

Parcial: hace copia de una parte del sistema

Incremental: compara las diferencias y las guarda con respecto a la última versión.

Al hacer la copia tenemos problemas con el espacio en disco y el tiempo

Con **LVM** evitamos poner en modo monousuario porque tiene **snapshots** (siempre que haya espacio suficiente)

Control de versiones

Hay algunas herramientas que controlan /etc directamente como: /etc/keeper

Utilizan git por debajo.

GIT es un sistema de archivos direccionable de contenido

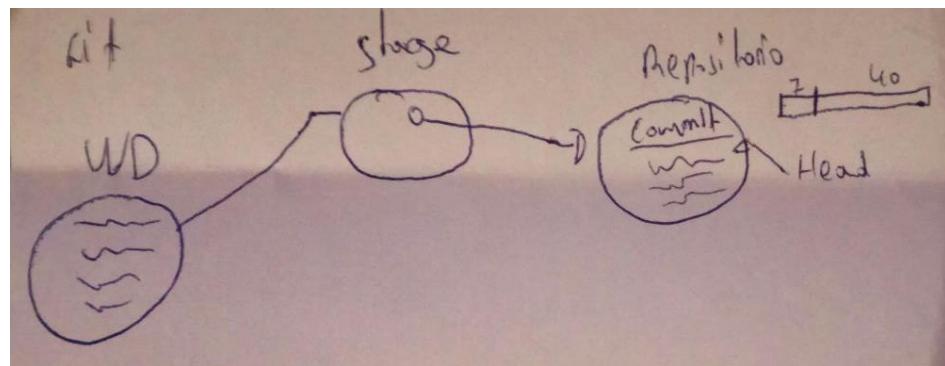
Lo programó Linus Torvalds.

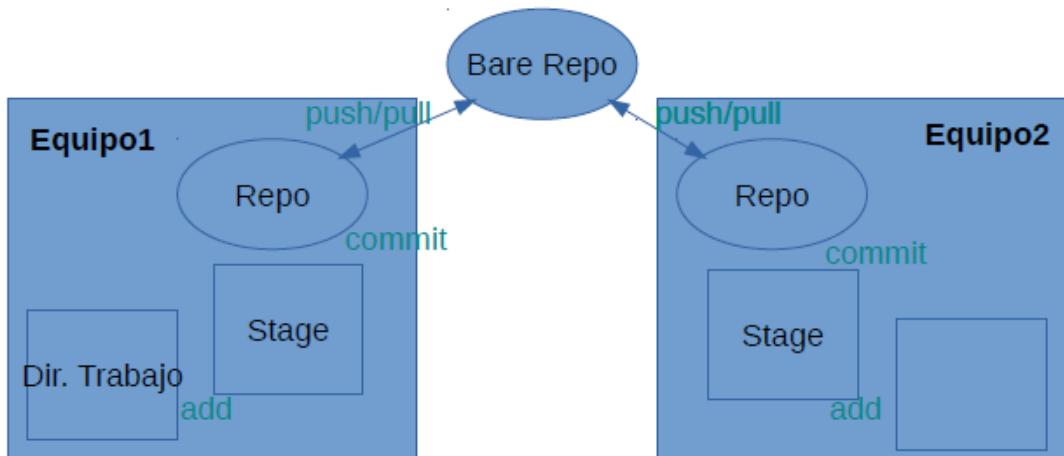
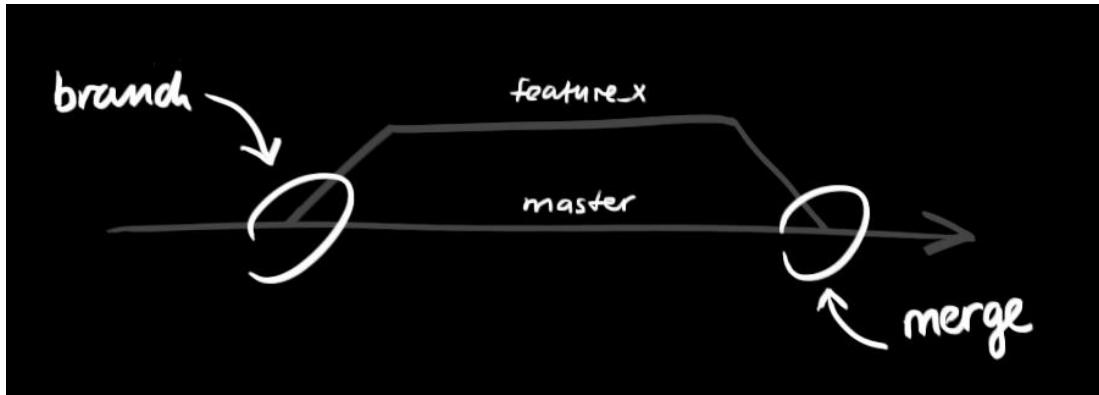
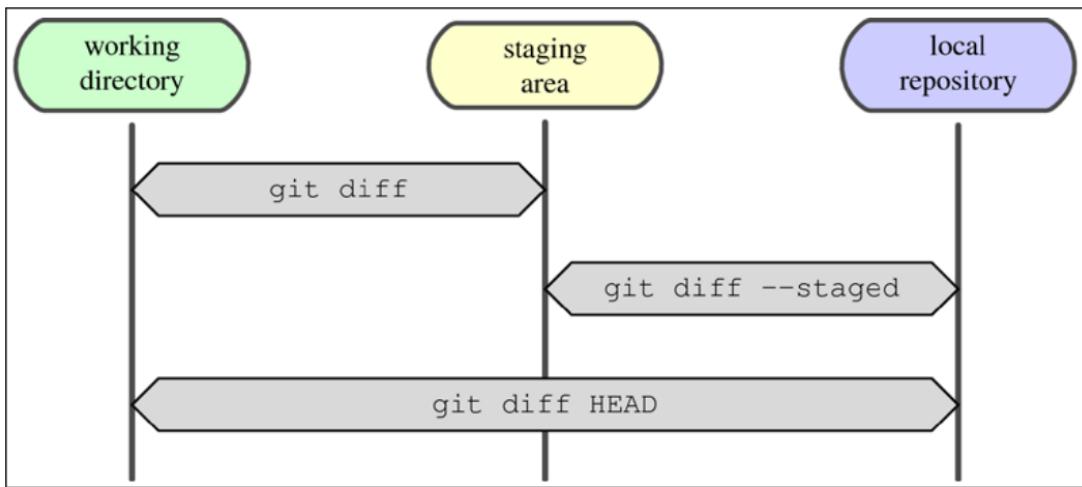
Ventajas (para ISE):

- Ponemos un pie en devops
- Seguimiento de los cambios (y de su autor)
- Permite planificar un entorno de prueba entre varias personas

Cómo funciona

- Directorio de trabajo: lo que tenemos en desarrollo
- Stage: Zona donde se registran los cambios. Ahí incluimos los cambios que queremos seguir.
- Commit: Zona donde los cambios se convierten en permanentes (podemos hacer commits en diferentes ramas “branches” y ponerles etiquetas “tags”)
- La evolución del trabajo puede mostrarse como un grafo dirigido acíclico (DAG)





Dia 3

Anotaciones

Información Básica

HTTP:

Hyper Text Transfer Protocol.

Utiliza el puerto 80 por defecto.

Hipertexto:

Documento que puede hacer referencia a otro documento o a una parte de el mismo.

Las peticiones se hacen desde un navegador (o mediante el comando curl).

HTML:

HyperTextMake-upLanguaje.

Incluyó la etiqueta <script> para poder añadir JavaScript a las páginas y hacerlas más dinámicas.

JavaScript:

Lenguaje de programación del lado del cliente.

No es capaz de almacenar información sensible.

Al ejecutarse en el cliente el código fuente está expuesto.

JSON:

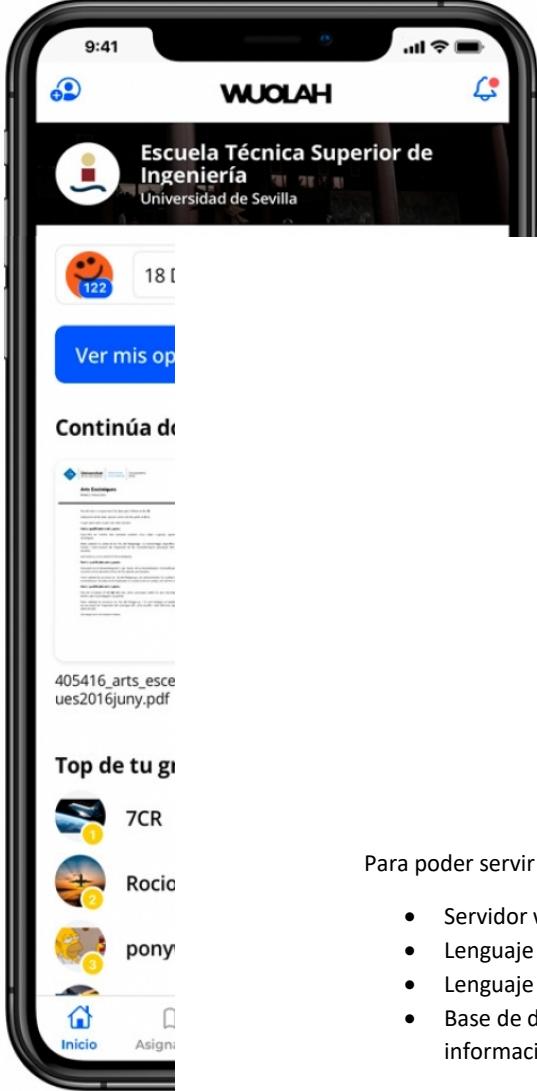
JavaScriptObjectNotation.

Se utiliza como estándar para que distintos lenguajes de programación puedan comunicar estructuras de datos entre sí.

Servidor web:

proceso que escucha el puerto 80 esperando peticiones de archivos y los transfiere a través del protocolo HTTP.

Apareció en el CERN en el año 89 y lo inventó Tim Berners-Lee.

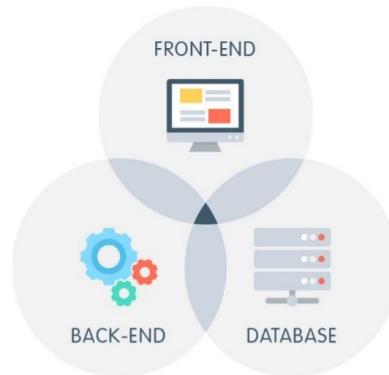


Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.

Available on the App Store

GET IT ON Google Play



Para poder servir una página dinámica y segura son necesarios al menos 4 elementos:

- Servidor web.
- Lenguaje Front-end: lenguaje que se ejecuta en el cliente.
- Lenguaje Back-end: lenguaje que se ejecuta en el servidor.
- Base de datos: ya sea relacional o no, una página dinámica necesita almacenar la información que se va a tratar o a mostrar a sus usuarios.

Algunos servidores web:

- Apache
- Nginx
- IIS
- Lighttpd

Algunas Base de datos:

- MySQL → mariadb
- PostgreSQL
- MongoDB → no relacional (noRDB) = NOSQL.

HDB → hybrid DataBase = es un tipo de bases de datos. Usan arquitectura hibrida entre relacional y no relacional.

Algunos lenguajes servidor (backend):

- PHP
- PYTHON
- PERL
- .NET
- NODE.JS
- GO

Los lenguajes backend tiene ciertas características en común. Estas son sus ventajas:

- no se tiene que compilar provocando menos fallos.
- La portabilidad.
- Librerías fáciles de usar para tratar strings y conectarse a BBDDs.

Pila:

Una pila es un conjunto de elementos software que nos permiten servir documentos manteniendo bajo control la información y la lógica del programa.

Algunas pilas:

- LAMP: LINUX APACHE MYSQL PHP
- NMP: NGINX MYSQL PHP

Framework:

Es una biblioteca de bibliotecas (metaBiblioteca)

Algunos framework pueden hacer de servidor web también, usando lenguajes como: Go, Python o node.js

Dato relevante:

servicio habilitado: cuando reinicio mi maquina system.d va a mirar si iniciar lo automáticamente.

servicio activo: el servicio está actualmente funcionando.

Buscar software malicioso npm

Npm es el gestor de paquetes de node.js

El último ataque conocido:

un hacker ha obtenido acceso a la cuenta npm de un desarrollador e injectado código malicioso en una popular biblioteca de node.js. El código malicioso robaba las credenciales de los usuarios que lo importaban a sus proyectos.

El paquete se llamaba eslint-scope (versión afectada 3.7.2).

El ataque se llevó a cabo en la madrugada del 11 al 12 de octubre del 2018.

Se cree que el ciberdelincuente usó el token npm recién generado para autenticar e insertar una nueva versión de la biblioteca eslint-scope en el repositorio npm de paquetes JavaScript.

Otros ataques:

El primer incidente ocurrió en agosto del 2017 cuando se detectaron 38 paquetes que robaban variables de entorno

El segundo incidente ocurrió en mayo de 2018 en el que alguien intento esconder una puerta trasera en el paquete getcookies.

El tercero y ultimo es el anteriormente descrito.

Calcular la diferencia de espacio para entre LAMP y servidor en centos
En centOs cuando instalas los paquetes te sale cuando ocupan (52,3MB totales):

Httpd: 10MB

Mariadb-server: 33MB

Php: 9.4MB

En Ubuntu con tasksel:

Ocupa 335MB (comprobado viendo cuando ocupa una instalación limpia y otra con LAMP)

Comandos

- curl <direccion> (hace una petición http a un servidor)
- php -a (abre un intérprete de php en la terminal)
- systemctl status/stop/restart mysql.service (Ubuntu)
- systemctl status/stop/restart apache2.service (Ubuntu)
- mysql -u <usuario> -p (conecta al servidor del localhost)
 - -u (usuario)
 - -p (pide contraseña)
- yum install httpd (instala el servidor http)
- yum install php (instala php)
- Yum install mariadb (instala el cliente mysql)
- yum install mariadb-server (instala el servidor mysql)
- systemctl status/stop/restart/enable httpd (centOs)
- systemctl status/stop/restart/enable mariadb (centOs)
- mysql_secure_installation (configura mysql de modo seguro)
- firewall-cmd --add-port=80/tcp --permanent (abre el puerto 80)
- Firewall-cmd –reload (reinicia el cortafuegos)
- yum install php-mysql -y (instala la librería para conectar a mysql con php)
- getsebool -a | grep httpd (lista las variable de SE asociada a httpd)
 - -a lista las variables
- setsebool -P httpd_can_network_connect_db on (permite a httpd conectar a db)
 - -P (hace el cambio permanente)
- yum install epel-release (instala epel, un conjunto de paquetes para Linux empresarial)
- yum install fail2ban (instala fail2ban)
- systemctl status/stop/restart/enable fail2ban
- fail2ban-client status (muestra las cárceles activas)
- fail2ban-client set sshd unbanip <ip>" (para sacar a alguien de la cárcel)
- fail2ban-client set sshd banip <ip>" (Para meterlo manualmente)
- Yum install tmux (instala tmux)
- Yum install screen (instala screen)
- Screen -list (lista los ficheros colgando de screen)
- Screen -r <número de tarea> (sigue con la tarea salvada)
- Tmux ls (lista los ficheros colgando de tmux)
- tmux attach -t <número de tarea> (sigue con la tarea salvada)

Instalación y pruebas

ubuntu:

Para instalar LAMP usamos el comando:

"tasksel"

seleccionamos solo LAMP server.

Comprobamos que el servicio este activo con:

"Systemctl status apache2.service"

Para ver que funciona correctamente usamos el comando

"curl localhost"

Para ver si php esta activo:

"php -a"

"echo('hola');"

"exit" (para salir)

Para comprobar el estado de mysql:

"Systemctl status mysql.service"

Para asegurarnos 100% nos conectamos con:

"Mysql -u root -p (para que nos pida el password)"

Podemos conectar a mysql

"exit" (para salir)

centOS:

Para instalar apache:

"Yum search httpd"

"Yum install httpd"

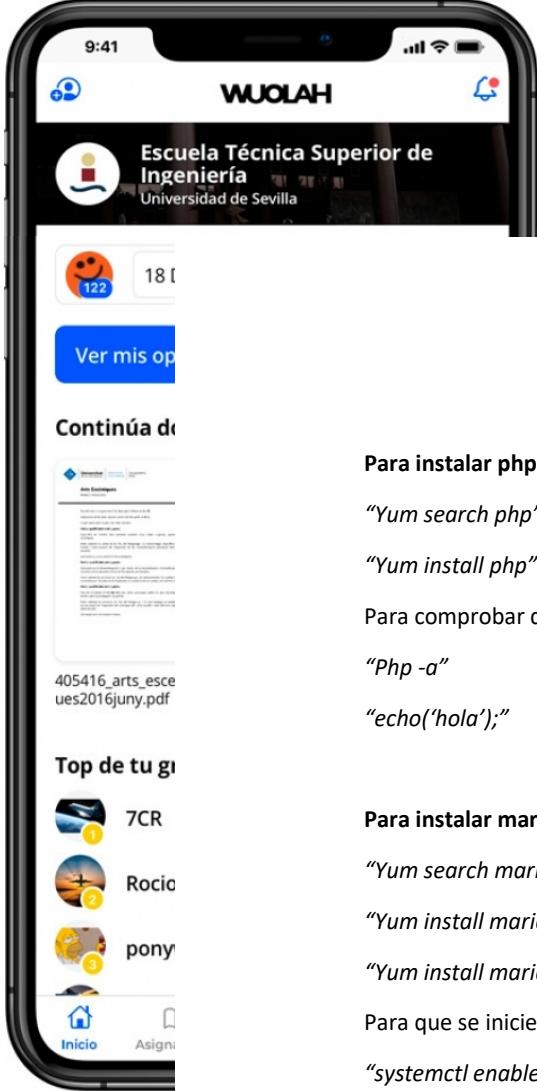
(Versión 2.4.6)

Para que apache se inicie cuando enciende el ordenador:

"Systemctl enable httpd"

Para activarlo ahora:

"Systemctl start httpd"



Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.

Available on the
App Store

GET IT ON
Google Play

Para instalar php:

"Yum search php"

"Yum install php"

Para comprobar que php funciona correctamente:

"Php -a"

"echo('hola');"

Top de tu gr

Para instalar mariadb:

"Yum search mariadb"

"Yum install mariadb" (instala el cliente)

"Yum install mariadb-server" (instala el servidor)

Para que se inicie al reiniciar el ordenador:

"systemctl enable mariadb"

"systemctl start mariadb"

Para comprobar el estado del servidor:

"Systemctl status mariadb"

para conectar al servidor mysql:

"mysql -u root -p" (sin contraseña, al principio)

para hacer segura la instalación de mysql:

"mysql_secure_installation" (pide la contraseña, eliminar usuarios anónimos, desactivar el login remoto de root, eliminar base de datos de test)

intentamos conectar al servidor web de centOs

comprobamos red y vemos que tienen comunicación:

"ping <ipservidor>"

Y hacemos la petición:

"curl <ipservidor>"

El firewall corta el acceso porque el puerto 80 está cerrado

Para solucionar el problema abrimos el puerto con:

“firewall-cmd --add-port=80/tcp –permanente”

Y reinicamos el cortafuegos para cargar la nueva configuración con:

“firewall-cmd --reload”

Ahora ya si se puede hacer la petición:

“curl <ipservidor>”

¿Se comunican mysql php y apache?

Hagamos un ejemplo para probarlo

Creamos un archivo php en /var/www/html/:

“Cd /var/www/html/”

“Touch miscript.php”

(Copiamos el ejemplo de la página del manual de php dentro del script)

<http://php.net/manual/es/function.mysql-connect.php>

```
<?php  
$enlace = mysql_connect('localhost', 'root', 'practicas,ISE');  
if (!$enlace){  
    die('No pudo conectarse: ' . mysql_error());  
}  
echo 'Conectado satisfactoriamente';  
mysql_close($enlace);  
?>
```

Hacemos petición desde el cliente

“curl <ipservidor>/miscript.php”

no funciona el código php

Modificamos el archivo de configuración de apache (/etc/httpd/conf/httpd.conf):

Tocar la directiva directoryIndex (ifModule dir_module) en apache para que ejecute todos los archivos .php (*.php)

Reiniciamos el servicio:

“systemctl restart httpd”

“systemctl status httpd”

“curl <ipservidor>/miscript.php”

Error al pedir página igual

solucionar:

`php miscript.php`

nos dice que no tenemos la biblioteca mysql_connect

para instalarlo:

`"yum search php | grep mysql"`

`"yum install php-mysql -y"`

y ya funciona el script:

`"php miscript.php"`

Aunque podamos ejecutar el script, todavía no podemos hacer peticiones a páginas php a nuestro servidor web porque SELinux nos bloquea

para listar las restricciones de SELinux:

`"getsebool -a | grep httpd"`

activamos que http pueda conectar a una bbdd:

`"setsebool -P httpd_can_network_connect_db on"` (la p mayúscula)

para que se habilite el cambio en selinux hay que reiniciar el servicio:

`"systemctl restart httpd"`

Ya debe funcionar todo perfectamente, para comprobar:

Con otra máquina: `"curl <ipservidor>/script.php"` o lo abrimos con el navegador

Atentos al contexto:

Si escribimos el script en el home de un usuario y lo movemos a /var/www/html seLinux no nos deja hacer peticiones a este archivo porque tiene contexto de una carpeta home

Lo arreglamos con:

`"restorecon <ruta al archivo>"`

Herramientas de seguridad:

Fail2ban:

Banea ips que atacan al servidor haciendo demasiadas peticiones seguidas. (mete esas ips en cárceles)

Fail2ban → EPEL (hay que instalarlo para poder usar fail2ban)

Para instalar y comprobar el estado de fail2ban:

"Yum install epel-release"

"Yum install fail2ban"

"Systemctl enable fail2ban"

"Systemctl start fail2ban"

"Systemctl status fail2ban"

Para mostrar información de las cárceles activas:

"Fail2ban-client status"

(número de cárceles 0)

Para añadir una cárcel hay que modificar el fichero de configuración de fail2ban:

Esta en la ruta: /etc/fail2ban

jail.local (no existe al inicio) será nuestro archivo de configuración porque si jail2ban se actualiza modifica el jail.conf y nos da esta posibilidad

entonces, para añadir una cárcel, primero copiamos el archivo y después modificamos jail.local:

"cp -a /etc/fail2ban/jail.conf /etc/fail2ban/jail.local"

modificamos el archivo:

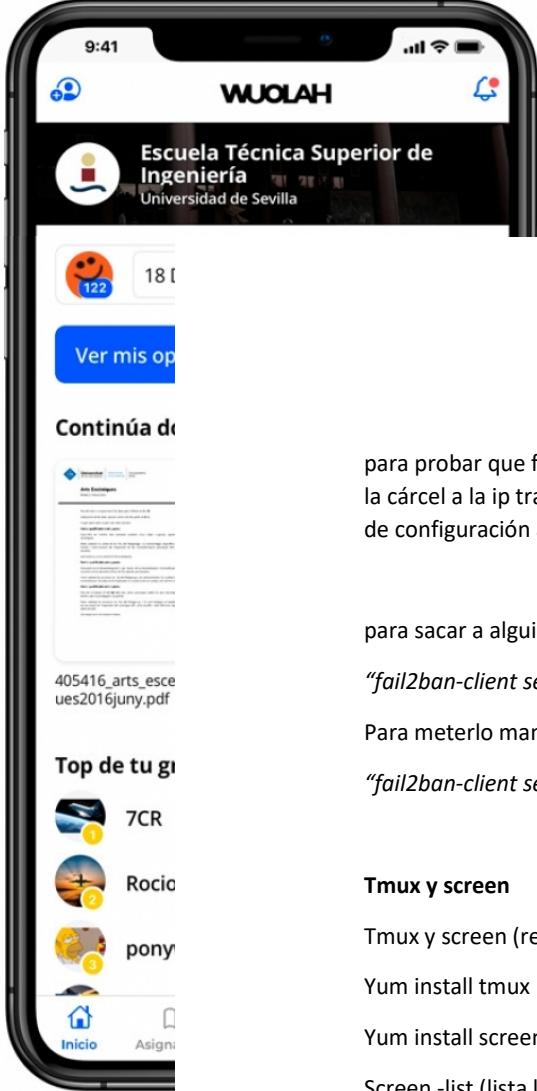
"vi /etc/fail2ban/jail.local"

enable = true

cambiar el puerto a 22022

reiniciamos servicio con:

"systemctl restart fail2ban"



Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.

Available on the
App Store

GET IT ON
Google Play

para probar que funciona bien intentamos conectar mal al servidor ssh. Como vemos, mete en la cárcel a la ip tras 5 intentos fallidos (se puede cambiar el número de intentos en el archivo de configuración anterior <maxretry>)

para sacar a alguien de la cárcel:

`"fail2ban-client set sshd unbanip <ip>"`

Para meterlo manualmente:

`"fail2ban-client set sshd banip <ip>"`

Tmux y screen

Tmux y screen (reengancha los procesos que se mueren por herencia)

Yum install tmux

Yum install screen

Screen -list (lista los ficheros colgando de screen)

Screen -r <número de tarea> (sigue con la tarea salvada)

Tmux ls (lista los ficheros colgando de tmux)

tmux attach -t <número de tarea> (sigue con la tarea salvada)

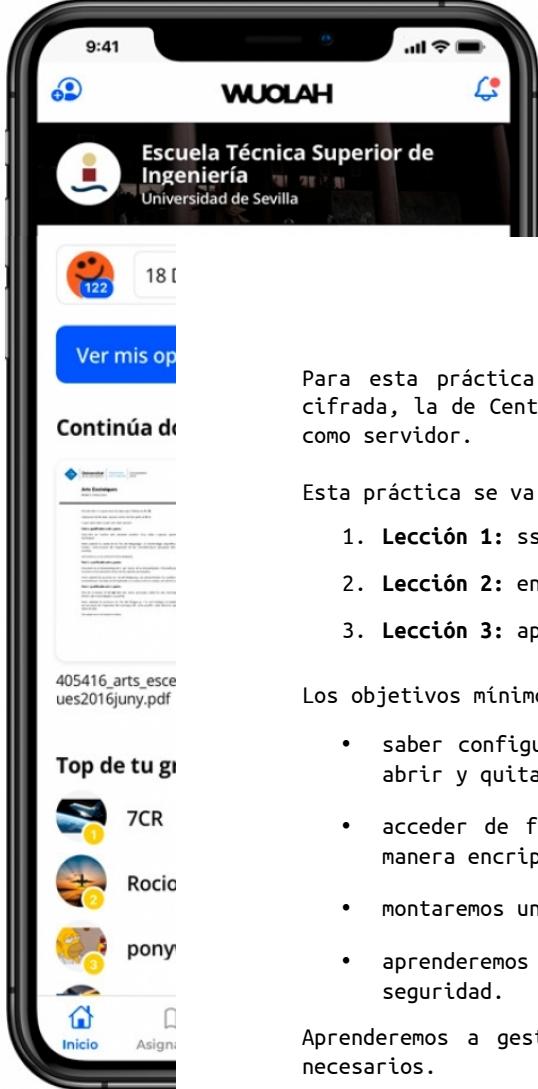
Rootkit hunter

(tambien es necesario instalar EPEL).

Rootkit = vulnerabilidades.

Rootkit hunter: rkhunter (Busca malware en el ordenador)

Sería conveniente hacer un análisis con este programa al menos una vez a la semana (o más veces)



Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.

Available on the
App Store

GET IT ON
Google Play

Para esta práctica se recomienda usar la máquina de la lección 2, la que no está cifrada, la de CentOS que la usaremos como cliente, mientras que la de Ubuntu se usará como servidor.

Esta práctica se va a dividir en 3 lecciones:

1. **Lección 1: ssh;**
2. **Lección 2: enfoque de seguridad y control de versiones (+ teórica);**
3. **Lección 3: apache server, para montar un servidor.**

Los objetivos mínimos son:

- saber configurar un cortafuegos, es decir, capar algunos puertos por seguridad, abrir y quitar unos puertos en nuestro sistema;
- acceder de forma remota a un puerto mediante ssh, es una forma de acceder de manera encriptada a un servidor;
- montaremos un servidor web con apache, SQL y PHP;
- aprenderemos sobre github, para hacer control de versiones y sobre copias de seguridad.

Aprenderemos a gestionar y montar un servidor web con todos los requisitos mínimos necesarios.

SSH

Es una forma de conectarse a un servidor remoto de forma encriptada.

→ **Historia:** ssh comenzó con un finlandés cansado de que le quitaran las contraseñas, por lo que montó ssh (Secure Shell), un telnet encriptado. Actualmente es estándar en todos los sistemas operativos.

Cortafuegos

El cortafuegos es una forma de capar, de bloquear los accesos no autorizados.

- **UFW (Uncomplicated Firewall)** → Ubuntu
- **firewall** → CentOS
- **iptables** → son módulos que tiene ya el sistema para poder manejar a mano los cortafuegos. Son más difíciles de usar con una nomenclatura más larga pero permite manejar muchas más funcionalidades.

fail2ban

Impide ataques de fuerza bruta para conseguir la clave a base de muchas peticiones al servidor. Funciona de forma que si la misma IP realiza varios ataques, varias peticiones seguidas con una fuente rara, elimina dicha IP con lo que todo lo que llega de dicha IP se ignora.

rkhunter (rootkit hunter)

Software desplegado, rkhunter busca vulnerabilidades del propio sistema, un resumen para que uno mismo corrija dichas debilidades.

1. Ubuntu como servidor

Máquinas virtuales:

Terminal del **servidor** → Ubuntu

Terminal del **cliente** → cualquier cliente host, CentOS, excepto si el host es Windows

- Comprobamos si tenemos conexión a internet, ya que nos hace falta para la práctica, con **ping**. Se supone que la IP que tenemos todos desde Ubuntu es la 105, hay que hacer:

```
$ ping 192.168.56.105
```

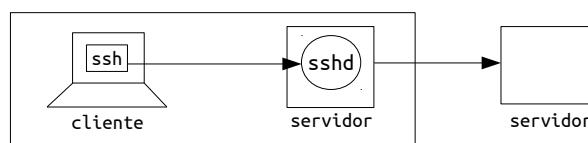
- Para instalar ssh, tenemos 2 formas:

1. **\$ sudo apt-get install openssh-server**
2. **\$ sudo tasksel** (esta es la que vamos a usar)

```
→ OpenSSH server  
→ Basic Ubuntu server  
→ Aceptar
```

Cuando se instala en Ubuntu el ssh, te crea dos cosas:

1. sshd, la d es de daemon, es un proceso que se arranca en un sistema normalmente en segundo plano, sin necesidad de ejecutarlo.
2. ssh, es ya como tal el cliente.



Ubuntu no respeta dicha nomenclatura cuando se quiere ver el servicio, ya que al ver el estado del servicio podemos realizarlo con ssh y sshd, cuando realmente sólo se debería ver con sshd. Sin embargo, con CentOS funciona bien.

Ssh hace conexión con el sshd que es un servidor. El servidor también tiene ssh cliente, que se podría conectar al sshd de otro servidor. En nuestra máquina ahora mismo está instalado todo, el ssh cliente, el sshd en background haciendo el servicio. Cualquier máquina puede funcionar como cliente o servidor. Es como funciona ssh, la línea está cifrada, los datos van cifrados.

- ¿Cómo sabemos si está ssh configurado después de instalarlo?

```
$ sudo su
```

```
$ systemctl status sshd
```

Nos dice que está activo y en qué puerto → 22 (puerto por defecto para las comunicaciones ssh); no es bueno dejar este puerto por seguridad, se recomienda cambiarlo.

- Cambiamos el puerto a 22022 (un puerto que no solape con otro):

```
$ vi /etc/ssh/sshd_config
```

Port 22022

```
# Package generated configuration file
# See the sshd_config(5) manpage for details

# What ports, IPs and protocols we listen for
Port 22022
```

- el sshd_config es el que hace referencia al servicio, al daemon
 - el ssh_config es el que hace referencia al cliente
- Comprobamos si ha cambiado:

```
$ systemctl status sshd
```

Vemos que no ha cambiado, ya que debemos reiniciar antes.

- Reiniciamos:

```
$ systemctl restart sshd
```

```
$ systemctl status sshd
```

Vemos que ya ha cambiado el puerto.

```
root@ps0:/home/ps0$ systemctl status sshd
● ssh.service - OpenBSD Secure Shell server
  Loaded: loaded (/lib/systemd/system/ssh.service; enabled; vendor preset: enabled)
  Active: active (running) since lun 2019-11-04 15:57:52 CET; 11s ago
    Process: 2610 ExecStartPre=/usr/sbin/sshd -t (code=exited, status=0/SUCCESS)
   Main PID: 2616 (sshd)
     Tasks: 1
       Memory: 1.8M
          CPU: 9ms
        CGroup: /system.slice/ssh.service
                  └─2616 /usr/sbin/sshd -D

nov 04 15:57:51 ps0 systemd[1]: Starting OpenBSD Secure Shell server...
nov 04 15:57:52 ps0 sshd[2616]: Server listening on 0.0.0.0 port 22022.
nov 04 15:57:52 ps0 sshd[2616]: Server listening on :: port 22022.
nov 04 15:57:52 ps0 systemd[1]: Started OpenBSD Secure Shell server.
```

- Intentamos acceder con ssh con el **cliente**:

```
& ssh usuario@192.168.56.105 -p 22022
```

- La IP podemos verla en ifconfig y debe ser la misma que ya se configuró
- Salimos de ssh y vamos al servidor.



Deberíamos hacer una copia del fichero de configuración de ssh, por si tenemos problemas cuando lo modificamos, poder volver al estado anterior.

- ¿Qué usuario está por defecto? El root. Vamos a cambiarlo para que solo entremos a root cuando estemos con nuestra cuenta:

```
$ vi /etc/ssh/sshd_config
```

```
PermitRootLogin no
```

```
# Authentication:  
LoginGraceTime 120  
PermitRootLogin no  
StrictModes yes
```

¿Por qué le quitamos permisos al root para hacer login?

Hemos configurado de forma que el root no entre directamente en el servidor a través de ssh, sino que el usuario tenga que hacer el login en ssh y el sudo. Esto se hace para dificultar los ataques por fuerza bruta que pueda realizar el hacker para intentar modificar mi sistema. Para entrar en mi sistema, debe conocer el usuario y la contraseña, y realizar sudo.

- Volvemos a reiniciar para que se actualicen los cambios:

```
$ systemctl restart sshd
```

- Comprobamos en el cliente que se han actualizado los cambios:

```
& ssh root@192.168.56.105 (IP de Ubuntu)
```

no nos debe permitir entrar.

- Actualmente, nos estamos conectando por contraseñas pero, ¿qué pasa si alguien nos coge la contraseña? Para ello creamos claves públicas y privadas, el ordenador y el servidor comprueban que las claves están bien, se conecta sin necesidad de pedir nada.
- Creamos las claves de los clientes, que después las pasaremos al servidor. Tenemos que asegurarnos que no estamos en root, si no, no se guardan en el directorio por defecto, sino que las guarda para root:

```
& ssh-keygen
```

lo ponemos todo por defecto y no ponemos contraseña.

- La clave pública se guardaba en ~/.ssh/id_rsa.pub;
- la clave privada se guardaba en ~/.ssh/id_rsa.





Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.

Available on the
App Store

GET IT ON
Google Play

¿Qué permisos deben tener estos archivos?

La clave privada → propietario y usuario deberían tener permisos de lectura/escritura

La clave pública → todos los procesos de usuario

¿Con qué permisos nos lo crea?

La clave privada → los de usuario y propietario

- Una vez que se han generado, enviamos la clave pública al servidor:

& ssh-copy-id cased@198.168.56.105 -p 22022

- El cliente que hemos configurado será el único que se pueda conectar al servidor:

\$ vi /etc/ssh/sshd_config

PasswordAuthentication no

Change to no to disable tunneled clear text passwords
PasswordAuthentication no

(quitamos el comentario si lo tiene y establecemos a no)

- Reiniciamos para que se guarden los cambios:

\$ systemctl restart sshd

```
paula@postdata9:~$ ssh pso@192.168.56.105 -p 22022
Welcome to Ubuntu 16.04.4 LTS (GNU/Linux 4.4.0-116-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

Pueden actualizarse 203 paquetes.
124 actualizaciones son de seguridad.

New release '18.04.3 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Tue Nov  5 23:18:39 2019
pso@pso:~$
```

Si intentamos entrar con nuestro usuario en el cliente, nos debería dejar entrar sin pedirnos contraseña.

Dar de alta a un nuevo usuario:

Si queremos añadir un usuario nuevo, no podríamos darle acceso a ssh, ya que está configurado de forma que solo yo puedo conectarme.

Si quisieramos dar de alta un usuario nuevo, tenemos que añadirlo en el fichero, habilitarlo otra vez, poner el PasswordAuthentication a "yes", que el usuario genere las claves, las pase al servidor y volver a modificar el PasswordAuthentication a no.

2. CentOS como servidor

Ahora vamos a intentar conectarnos al servidor mediante ssh utilizando CentOS como servidor y Ubuntu como cliente.

Terminal del **servidor** → CentOS
Terminal del **cliente** → Ubuntu

Adaptador de red Host-Only

El adaptador de red de Host-Only que añadimos en Ubuntu, también se puede añadir en CentOS, lo que nos permite que ambas máquinas virtuales se comuniquen entre sí, que establezcan conexiones entre ellas. Ambas direcciones IP deberían ser diferentes.

Arrancamos **CentOS**.

Comprobamos la conexión

- Comprobamos los conectores red que tenemos:

& **ip address**

deberíamos tener dos conectores de red, uno conectado a NAT, que es el que está conectado a la red Host-Only.

Gracias a la red Host-only debería haber comunicación entre las 2 máquinas. Si es así, vamos a probar si funciona la conexión ssh, desde Ubuntu a CentOS.

- Comprobamos si funciona la conexión ssh desde Ubuntu a CentOS:

\$ ping 192.168.56.110 (ip de CentOS)

```
ps0@ps0:~$ ping 192.168.56.110
PING 192.168.56.110 (192.168.56.110) 56(84) bytes of data.
64 bytes from 192.168.56.110: icmp_seq=1 ttl=63 time=2.38 ms
64 bytes from 192.168.56.110: icmp_seq=2 ttl=63 time=1.90 ms
64 bytes from 192.168.56.110: icmp_seq=3 ttl=63 time=1.81 ms
^C
--- 192.168.56.110 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2014ms
rtt min/avg/max/mdev = 1.812/2.032/2.383/0.255 ms
```

vemos que funciona la conexión. Nos avisa si queremos continuar y nos muestra el *fingerprint*, que es un hash de la clave pública (se utiliza para cerciorarse de que el servidor al que nos estamos conectando es al que nos queremos conectar de verdad, el servidor es el único que tiene dicho hash).

Esta es una diferencia con respecto a Ubuntu, en CentOS ssh viene instalado y funcionando, nos podemos loggear directamente sin problemas de Ubu a Cent.

Cambiamos el puerto por defecto en CentOS

Vamos a repetir la misma configuración de cambiar el puerto en el cual escucha por defecto CentOS al 22022.

- Entramos en la carpeta en la que está la configuración de ssh

```
& cd /etc/ssh
```

- Cambiamos en el fichero sshd_config el puerto de escucha con sed:

```
& sudo sed s /'Port 22'/'Port 22022'/ -i sshd_config
```

Función sed

sed → String Editor

Nos permite hacer cambios en un fichero de texto sin necesidad de tener un editor de texto. La opción -s es de sustituir una cadena; la opción -i dice qué fichero vamos a modificar.

- Comprobamos que se ha cambiado bien:

```
& vi sshd_config
```

Nos lo ha cambiado bien, pero la línea está comentada, por lo que tenemos que descomentárla.

```
# If you want to change the port on a SELinux system, you have to tell
# SELinux about this change.
# semanage port -a -t ssh_port_t -p tcp #PORTNUMBER
#
Port 22022
```

- Cargamos el fichero:

```
& systemctl restart sshd
```

→ nos da error

```
[root@localhost ssh]# systemctl restart ssh
Failed to restart ssh.service: Unit not found.
```

- Vemos qué ha pasado, por qué da error:

```
& systemctl status
```

no se ha podido iniciar el servicio, pero no nos da mucha más información

- Para saber más información acerca del error:

```
& journalctl -xe
```

nos dice que el proceso sshd de servicio no puede abrir el puerto 22022.



No puede abrir el puerto 22022, debido a que en SELinux el acceso está más dosificado, cada proceso tiene lo que necesita, y sshd tenía acceso al puerto 22, no al 22022, no tiene permiso para acceder a dicho puerto.

Esto en Ubuntu no dio problema debido a que no está instalado SELinux, mientras que en CentOS está instalado y activado.

Vamos a cambiar la configuración de SELinux:

- Necesitamos un paquete de utilidades que no vienen por defecto en CentOS, nos instala el comando semanage (Security Manage), que nos permite modificar las políticas de seguridad que nos impide ejecutar sshd en el puerto 22022:

```
& yum install policycoreutils-python
```
- Listamos los puertos que tiene SELinux y vamos a filtrarlos para que nos muestre el 22:

```
& semanage port -l | grep 22
```

```
pop_port_t          tcp      106,
ssh_port_t          tcp      22
zented_port_t       tcp      1229
```

vemos que el puerto para ssh es el 22

- Cambiamos la configuración de SELinux:

```
& semanage port -a -t ssh_port_t -p tcp 22022
```

-a: añadir
-t: tipo ssh_port_t
-p: puerto que permita conexiones tcp
- Comprobamos que ya ssh tiene permiso para el puerto 22022:

```
& semanage port -l | grep 22
```

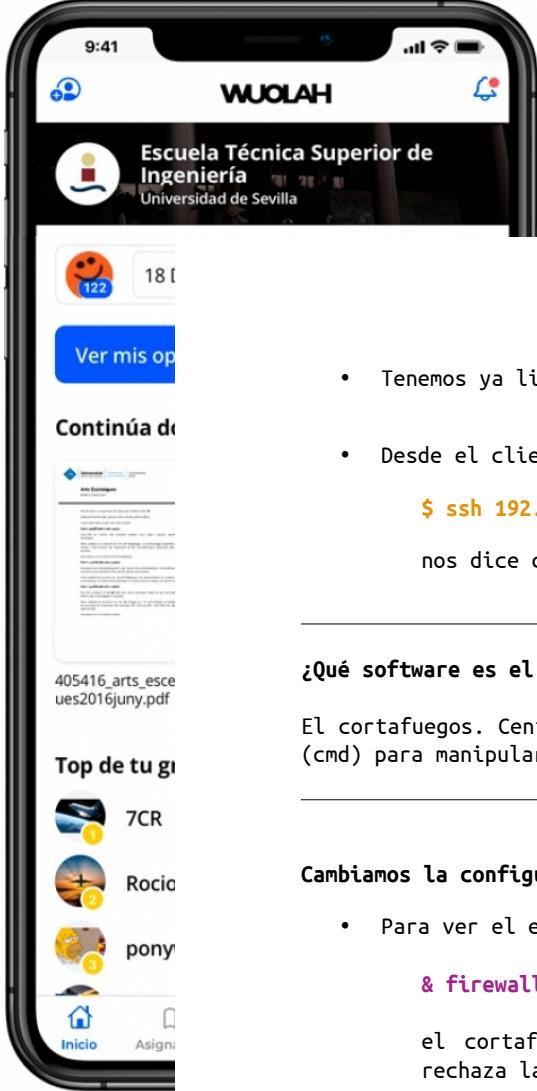
```
pop_port_t          tcp      106, 109,
ssh_port_t          tcp      22022, 22
zented_port_t       tcp      1229
```

- SELinux no nos debería de dar problema al ejecutar sshd.
- Reintentamos activar sshd:

```
& systemctl restart sshd
```
- Comprobamos que se está ejecutando y en el puerto 22022:

```
& systemctl status sshd
```

está *enabled*, está activo.



Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.

Available on the
App Store

GET IT ON
Google Play

- Tenemos ya listo el servidor para que se conecten los clientes.

- Desde el cliente (Ubuntu) nos conectamos:

```
$ ssh 192.168.56.110
```

nos dice conexión reusada.

¿Qué software es el que puede impedir que una conexión no surta efecto?

El cortafuegos. CentOS tiene un cortafuegos que se llama firewall y tiene unos comandos (cmd) para manipularlos.

Cambiamos la configuración del firewall en el servidor (CentOS)

- Para ver el estado del cortafuegos:

```
& firewall-cmd --stat
```

el cortafuegos se está ejecutando y como no tiene constancia del puerto, rechaza la conexión.

- Añadimos el puerto que necesitamos en el cortafuegos:

```
& firewall-cmd --add-port=22022/tcp --permanent
```

El cortafuego de CentOS

El cortafuegos de CentOS nos permite cambiar su configuración de dos formas:

- cambiando la configuración de la instancia que se está ejecutando en el cortafuegos;
- cambiando los ficheros de configuración del cortafuegos. Estos ficheros son los que se van a cargar en el arranque, de forma que cualquier cambio permanecerá.

Estos cambios no se pueden hacer a la vez. Con la opción `--permanent` modificamos los parámetros de configuración de forma que cuando reiniciemos la máquina, permanezca el puerto 22022.

- Hemos modificado los ficheros de configuración, pero no la instancia actual. Con el parámetro `reload`, podemos cambiar la configuración sin necesidad de reiniciar y ejecutar el comando con todos los parámetros:

```
& firewall-cmd --reload
```

- Nos conectamos desde el cliente a ssh, indicando el puerto de escucha 22022 y vemos que ya no nos da error:

```
$ ssh 192.168.56.110 -p 22022
```

Cambiamos la configuración del firewall del cliente (Ubuntu)

- Activamos el firewall:

```
$ sudo ufw enable
```

- Añadimos el puerto:

```
$ sudo ufw allow 22022
```

- Ya podemos conectarnos desde CentOS a Ubuntu.

```
$ ssh 192.168.56.110 -p 22022
```

3. rkHunter y fail2ban

rootkit

Es un software que permite a un usuario, normalmente malintencionado (un malware), acceder a un ordenador de forma que el ordenador víctima (anfitrión) no se da cuenta que están accediendo a él.

Tienen una característica muy llamativa, y es que burlan a los antivirus. Los antivirus para buscar virus/malware usan las funciones del sistema operativo para ir listando los ficheros que hay en los directorios, van abriendo los ficheros y los van escaneando buscando patrones.

El rootkit modifica las funciones que usan del SO para listar un directorio, de forma que cuando el antivirus va listando los ficheros y encuentra el del virus, lo omite. Esto se debe a que introduce código en las bibliotecas de las funciones del SO para que no detecte los virus como peligrosos.

¿Qué haría el antivirus para detectar malware si ni siquiera puede fiarse de las propias funciones del SO?

- Coger un USB con el SO limpio, reiniciamos con el SO limpio y pasar el antivirus con dicho SO, aunque esto es bastante ineficiente.
- Muchas funciones lo que hacen es implementar las funciones de listar fichero a mano, es decir:
 - leen los sectores del disco,
 - lee sus propios drivers para acceder al sistema de ficheros,
 - a mano va listando los ficheros,
 - compara esa lista de ficheros con el listado de ficheros que proporciona las funciones de las bibliotecas del SO.
 - Si no son los mismos, es porque alguno puede ser un malware.

rkhunter (RootKit Hunter)

Es el que vamos a usar y es un software que nos permite detectar los rootkit, de forma que calcula el hash de los ficheros de bibliotecas y esto es más complicado de falsificar. Lo usaremos con la opción -c para realizar un chequeo.

- Para instalarlo, por ejemplo, en Ubuntu:

```
$ sudo apt-get install rkhunter  
sin configuración
```

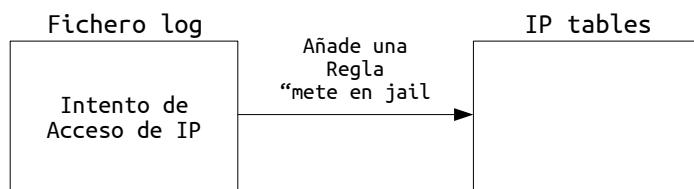
- Lo ejecutamos para ver que funciona:

```
$ rkhunter -c  
va listando el sistema y calculando el hash de archivos que son críticos
```

fail2ban

Es un software que monitoriza fallos de conexión de los ficheros de log. En los archivos log aparecen los intentos de accesos de direcciones IP y fail2ban los utiliza para modificar las reglas del cortafuegos.

El cortafuegos típico de Linux trabaja con las iptables, con lo que fail2ban añade una regla en la que “mete en la cárcel, mete en jail” esos intentos de accesos, la IP. Podemos modificar la configuración del fail2ban para establecer los intentos de conexión deben haber ocurrido para añadir dicha IP en jail, o cuánto debe haber transcurrido entre esos fallos de conexión.



- Lo instalamos, por ejemplo, en CentOS. Como no viene por defecto, debemos instalar primero el repositorio en el que se encuentra:

```
& yum install epel-release
```

- Instalamos fail2ban:

```
& yum install fail2ban-all.noarch  
sí a todo
```

- Arrancamos y activamos fail2ban:

```
& systemctl start fail2ban
```

```
& systemctl enable fail2ban
```

- fail2ban se maneja a través del comando fail2ban-client, podemos ver las IPs que están en jail con:

```
& fail2ban-client status
```

```
[root@localhost psol]# fail2ban-client status  
Status  
`- Number of jail:      0  
`- Jail list:
```



fail2ban viene desactivado y sin arrancar en Ubuntu, mientras que en CentOS hay que instalarlo. Vamos a usarlo sobre ssh, aunque se puede usar sobre más aplicaciones; y vamos a poner con él una capa más de seguridad, estableciendo que si una persona se equivoca 5 veces, fail2ban la bloquee diez minutos.

- Vemos el estado de ssh:

```
& fail2ban-client status sshd
```

→ nos dice que no existe

- Nos vamos a la configuración de fail2ban:

```
& vi /etc/fail2ban/jail.conf
```

→ nos dice que no lo modificuemos

- Por tanto, vamos a realizar una copia local del fichero:

```
& cp -a /etc/fail2ban/jail.conf /etc/fail2ban/jail.local
```

- Ahora modificamos el archivo local:

```
& vi /etc/fail2ban/jail.local
```

nos vamos para abajo hasta la sección de ssh. Hay una sección que se llama jails y aquí ponemos, debajo de sshd:

- enabled=true; si no no activamos la cárcel para ssh

```
#  
# JAILS  
#  
#  
# SSH servers  
#  
  
[sshd]  
  
#para activar la cárcel para ssh  
enabled=true
```

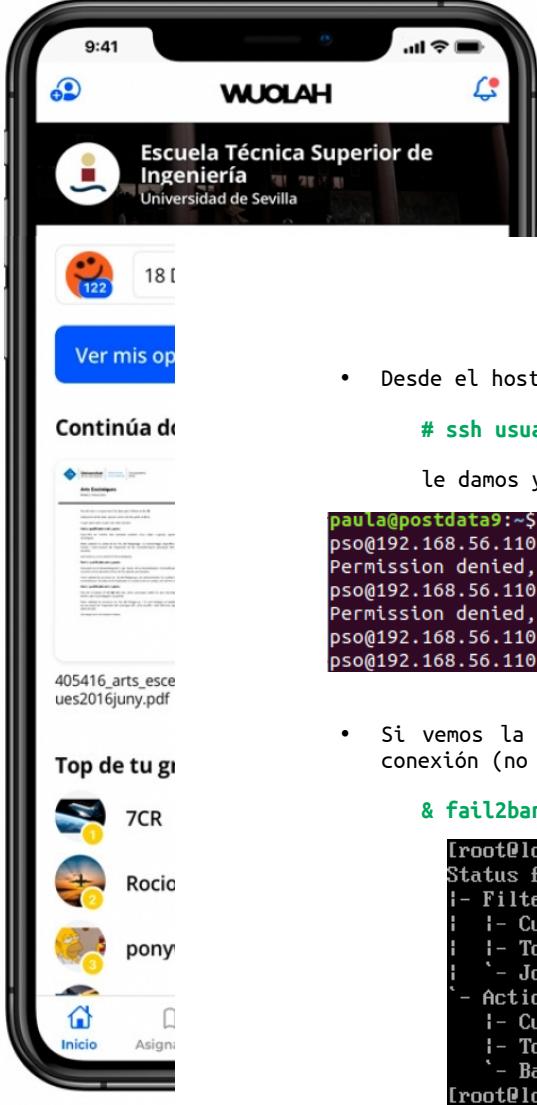
- Reiniciamos fail2ban y con status sshd podemos ver la cárcel de ssh:

```
& systemctl restart fail2ban
```

```
& fail2ban-client status sshd
```

nos dice que no ha habido ningún intento fallido, que ningún usuario ha sido baneado

```
[root@localhost ps0]# fail2ban-client status sshd  
Status for the jail: sshd  
|- Filter  
| |- Currently failed: 0  
| |- Total failed: 0  
|   '- Journal matches: _SYSTEMD_UNIT:sshd.service + _COMM:sshd  
'- Actions  
  |- Currently banned: 0  
  |- Total banned: 0  
    '- Banned IP list:
```



Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.

Available on the App Store

GET IT ON Google Play

- Desde el host (terminal del ordenador propio, no mv) hacemos:

```
# ssh usuario@192.168.56.110 -p 22022
```

le damos y nos equivocamos 3 veces, a la tercera vez ssh nos echa

```
paula@postdata9:~$ ssh pso@192.168.56.110 -p 22022
pso@192.168.56.110's password:
Permission denied, please try again.
pso@192.168.56.110's password:
Permission denied, please try again.
pso@192.168.56.110's password:
pso@192.168.56.110: Permission denied (publickey,gssapi-keyex,gssapi-with-mic,password).
```

- Si vemos la cárcel de sshd, nos muestra que ha habido 5 intentos fallidos de conexión (no sabemos por qué):

```
& fail2ban-client status sshd
```

```
[root@localhost pso]# fail2ban-client status sshd
Status for the jail: sshd
|- Filter
| |- Currently failed: 0
| |- Total failed: 0
| |- Journal matches: _SYSTEMD_UNIT=sshd.service + _COMM=sshd
|- Actions
| |- Currently banned: 0
| |- Total banned: 0
| |- Banned IP list:
[root@localhost pso]# fail2ban-client status sshd
Status for the jail: sshd
|- Filter
| |- Currently failed: 2
| |- Total failed: 5
| |- Journal matches: _SYSTEMD_UNIT=sshd.service + _COMM=sshd
|- Actions
| |- Currently banned: 0
| |- Total banned: 0
| |- Banned IP list:
```

- Si volvemos a intentar entrar fallando 2 veces más, para que nos baneen, vemos que no nos elimina. Esto se debe a que el puerto de ssh está en el 22, no en el 22022. Para cambiarlo, nos vamos al jail.local:

```
& vi /etc/fail2ban/jail.local
```

volvemos a la parte de las cárceles y donde pone port, cambiamos 22 por 22022

```
[sshd-ddos]
# This jail corresponds to the standard configuration in Fail2ban.
# The mail-whois action send a notification e-mail with a whois request
# in the body.
port      = 22022
logpath  = %(sshd_log)s
backend   = %(sshd_backend)s
```

- Guardamos el cambio anterior y reiniciamos fail2ban:

```
& systemctl restart fail2ban
```

- Si probamos ahora, no nos debería dejar entrar, porque ya estábamos en la lista de baneados y ahora el puerto sí es 22022, con lo que durante 10 minutos no nos va a dejar (los 5 intentos fallidos para banear y 10 minutos sin acceder son por defecto de fail2ban).

```
[root@localhost psol]# fail2ban-client status sshd
Status for the jail: sshd
|- Filter
| |- Currently failed: 2
| |- Total failed: 10
| ` Journal matches: _SYSTEMD_UNIT=sshd.service + _COMM=sshd
`- Actions
  |- Currently banned: 1
  |- Total banned: 1
  ` Banned IP list: 192.168.56.1
```

- Vamos a desbanearnos manualmente:

```
& fail2ban-client set sshd unbanip 192.168.56.1
```

- Si hacemos un fail2ban status sshd, veremos que no hay ninguna IP eliminada.

```
[root@localhost psol]# fail2ban-client set sshd unbanip 192.168.56.1
192.168.56.1
[root@localhost psol]# fail2ban-client status sshd
Status for the jail: sshd
|- Filter
| |- Currently failed: 2
| |- Total failed: 10
| ` Journal matches: _SYSTEMD_UNIT=sshd.service + _COMM=sshd
`- Actions
  |- Currently banned: 0
  |- Total banned: 1
  ` Banned IP list:
```

- Y si hacemos ssh, nos deja escribir el password.

4. Comparación entre Ubuntu y CentOS

Tabla comparativa con diferencias principales entre Ubuntu y CentOS:

	Ubuntu server 16	CentOS 7
identificador de servicio SSHD	ssh / sshd	sshd
sshd instalado por defecto	no	sí
fichero sshd_config	sin comentarios en línea de puerto	línea de puerto comentada
Cortafuegos	ufw (desactivado por defecto)	firewall -cmd (activado por defecto)
al instalar un servicio (daemon)	arrancado y activado	parado y desactivado

Esta sesión no se va a hacer en el ordenador, sino que hay un tutorial, un pdf, en el que lo leeremos en casa y vamos probando los comandos que se van tratando.

Esta sesión trata de enfoques de seguridad, que son muy importantes en los servidores, y control de versiones, porque hay un momento en el que podemos tener varias versiones de la misma copia de seguridad, de los mismos ficheros, y conviene tener a veces programas y herramientas que nos permitan agilizar este control de versiones.

1. Copias de seguridad

dd

copia el contenido bit a bit, hace una copia binaria y nos permite copiar de un dispositivo a un fichero.

- if: fichero del que lee;
- of: fichero en el que escribe;

cpio

Copia archivos a y desde archivos.

- -o: crear un nuevo archivo;
- -v: listar los archivos copiados;
- -i: extraer de un archivo;
- -d: crear directorios en caso necesario.

Ejemplo:

```
$ ls | cpio -ov > /tmp/object.cpio
```

Ejemplo:

```
$ cpio -idv < /tmp/object.cpio
```

En español, fichero y archivo son sinónimos pero en inglés no. En inglés un fichero, un *file* es lo que nosotros llamamos fichero o archivo; y un archivo en inglés es un fichero empaquetado, un .zip.





Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.

Available on the App Store

GET IT ON Google Play

tar

Es para copiar archivos y crear archivos empaquetados. Es conveniente saber los parámetros de tar por si queremos hacer una copia de seguridad de un directorio.

- **-c:** para crear un archivo;
- **-v:** para listar nombre de ficheros;
- **-z:** para comprimir con gzip;
- **-f:** para especificar el nombre del archivo;
- **-x:** para descomprimir;

Ejemplo: creamos un archivo que se llame ssh.tgz

```
$ tar -cvzf ssh.tgz
```

se nos crea el fichero ssh.tgz y nos mete todos los ficheros que hay en /.ssh

Ejemplo: si queremos descomprimir:

```
$ tar -xvf ssh.tar
```

nos descomprime el archivo en el directorio actual

Hay que tener cuidado con tar porque no pide confirmación cuando sobreescribe un fichero y puedes cargarte un fichero sin darte cuenta.

rsync:

Nos permite hacer copias, y comprueba el origen y el destino, si son iguales, solo transfiere las diferencias, las comprime y tiene en cuenta los *time stamps*, las marcas de tiempo. Está más optimizado, permite copiar a través de una conexión ssh, comprime lo que estamos transfiriendo para minimizar ancho de banda.

Vamos a ver los parámetros básicos:

- **-a:** nos permite realizar en un solo parámetro copia recursiva, copia los enlaces simbólicos, las fechas, ficheros especiales;
- **-v:** lista ficheros copiados
- **-i:** muestra las transferencias realizadas en cada fichero
- **-z:** utiliza compresión al transmitir

rsync también implementa una lógica que hace un parsing del origen que especificamos y en función de si ponemos barra final, punto o asterisco, realiza un copiado u otro.

- **../.ssh** esto copia también el directorio .ssh
- **../.ssh/ y ../.ssh/.** ambos no copian el directorio .ssh, solo el contenido
- **../.ssh/*** no copia los ficheros ocultos (esto se debe a que el shell cuando le ponemos el *, cambia el * por todos los nombres de los ficheros que se encuentran en la carpeta).

2. Backups

Varios comandos para hacer backups:

- **rsnapshot**: Para hacer instantáneas basadas en rsync.
- **Tartarus**
- **Backup2l**
- **Duplicity**
- **Sftpclone**

3. Control de cambios

/etc/keeper

nos permite hacer una copia de seguridad del contenido /etc, sabemos que en Linux, en etc están todos los contenidos de configuración, de servicio, de programas del sistema, por lo que es interesante tener una copia de seguridad.

/etc/keeper nos hace la copia de seguridad, usando el control de versiones.

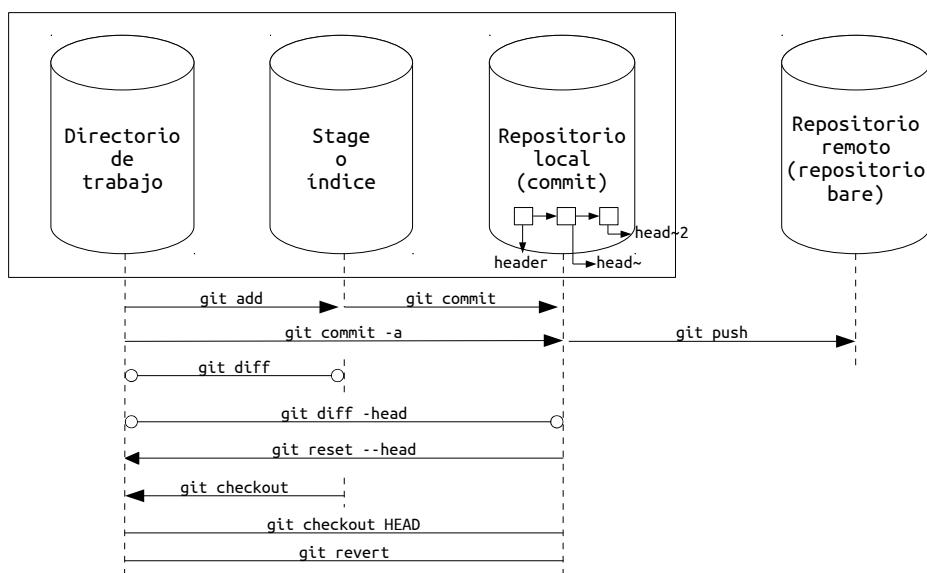
cp

permite hacer copias de un dispositivo, es un buen hábito realizar una copia (con -a mejor) de los ficheros de configuración antes de modificarlos.

4. Git

Git es un programa que nos permite tener copias de seguridad. Se utiliza mucho en programas de software, ya que nos permite tener distintas versiones de los ficheros.

Cuando hacemos un cambio, lo subimos al servidor y tenemos un historial de todos los cambios que hemos hecho, para poder ir hacia atrás o hacia delante, si queremos recuperar una versión antigua. Tiene muchísimas ventajas, la mayoría de ellas vienen del hecho de poder automatizar los procesos.



Tenemos 4 etapas:

- **Directorio de trabajo:** que es un almacenamiento;
- **Stage:** otra etapa (stage ó índice);
- **Repositorio local:** es el historial de las versiones (así podemos tener distintas versiones de nuestro proyecto). Header es la última versión, a la penúltima se le llama head~, head~2, ...;
- **Repositorio bare:** es el repositorio remoto.

Podemos ir pasando los ficheros de una etapa a otra, para lo que hay varios comandos:

- **git init:** para crear los repositorios;
- **git add:** pasa los ficheros al stage;
- **git commit:** pasa los ficheros al repositorio local;
- **git push:** inserta los commit en el repositorio remoto;
- **git commit -a:** pasa los ficheros directamente al repositorio local, con lo que nos ahorraremos hacer add seguido de commit;

Con esta estructura, todo esto está en nuestro cliente, en nuestro ordenador local. Se encuentra en el directorio del proyecto .git,..

Vamos a probar a crearlo con Ubuntu:

- Configuramos nuestro usuario de git para que cuando subamos ficheros a los repositorios aparezcan con nuestros nombres:

```
$ git config --global user.name Nombre  
$ git config --global user.email correo@ugr.es
```

Con esto ya tenemos lo básico de git para configurarlo.

- Para crear los repositorios, dentro de nuestra carpeta de proyecto:

```
$ git init
```

Git nos permite estar trabajando localmente sin tener conexión a Internet, ya que estamos trabajando en el repositorio local. Cuando queremos mandar una actualización, nos llevamos todos estos cambios al repositorio remoto, que es cuando realizamos el push. Esto nos permite trabajar offline.

Hay veces que cambiamos el fichero de código fuente y nos damos cuenta que está mal ese último cambio pero sí estaba bien el fichero anterior, entonces sólo nos interesa subir al repositorio un fichero de todos los que hemos cambiado.

Como hay varias etapas, hay varios comandos para comparar cuáles son los ficheros que están en uno o en otro:

- **git diff**: nos hace una comparación de lo que hay en el directorio de trabajo y en el índice;
- **git diff -head**: head es el último commit del repositorio, estamos comparando el contenido del directorio de trabajo con el último commit que haya en el repositorio local;
- **git diff --staged**: muestra las diferencias entre el stage y el repositorio;
- **git reset --head, git checkout, git checkout HEAD, git revert**: deshacer lo que hemos hecho, volver hacia atrás y hacer que el directorio de trabajo contenga lo que había en un commit anterior.

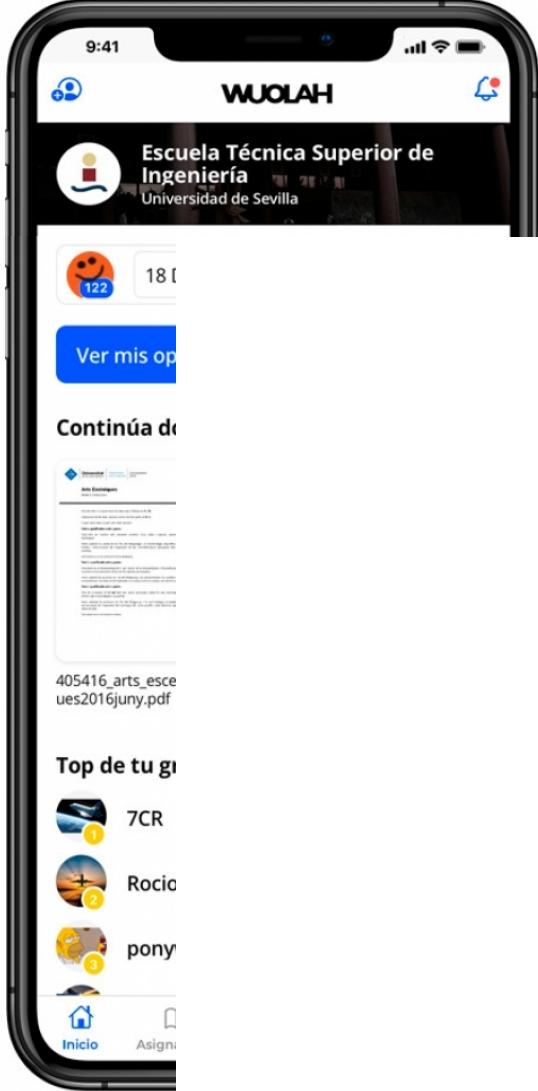
En un mismo repositorio remoto, puede haber trabajando varios clientes, todos realizando varios comandos. En este esquema en el que tenemos un servidor y varios clientes se llama Word Plow Servidor Central.

Git tiene varios objetos, los cuales tienen un hash para ser identificados:

- **blobs**: son archivos e inodos;
- **trees**: directorios;
- **commits**

Más comandos de git que pueden resultar interesante:

- **git log**: muestra registros de confirmación;
- **git status**: muestra el estado de git;
- **git branch**: lista, crea o elimina ramas;
- **git commit --amend**: modifica el mensaje;
- **git commit --amend -a**: añade algún archivo nuevo;
- **git clone**: clona un repositorio;
- **git add remote**: añadimos un origen remoto;
- **git fetch**: trae los cambios del repositorio;
- **git merge**: une commits con los del directorio actual;
- **git pull**: git fetch + git merge;
- **git stash**: permite salvar el directorio actual sin tener que realizar un commit;
 - **git stash apply**: aplica el último stash;
 - **git stash list**: muestra los disponibles;



Descarga la APP de Wuolah.
Ya disponible para el móvil y la tablet.

Available on the
App Store

GET IT ON
Google Play

Lección 3:

Apache Server

Vamos a configurar un servidor web, tanto para Ubuntu como para CentOS. En Ubuntu vamos a usar LAMP mientras que en CentOS lo haremos todo a mano; con Ubuntu será fácil mientras que CentOS será más difícil.

LAMP son las siglas de Linux Apache MySQL PHP, es un conjunto de herramientas que nos permite montar de forma fácil un servidor web. Antes estaban por separado, pero ahora está todo junto.

- Apache es una forma de realizar un servicio web, de los más estandarizados.
- MySQL es un gestor de bases de datos. A partir de CentOS 7 se sustituye MySQL por MariaDB que es un gestor de MySQL, es lo mismo pero con distinto nombre.
- PHP es un lenguaje interpretado para páginas web

¿Cuál es la diferencia entre una página web estática y dinámica?

La página web estática es una página que no se modifica dependiendo de la persona que entre, es siempre igual, no tiene ninguna alteración ni ninguna llamada por detrás; las dinámicas cambian, detrás hay una serie de lenguajes y herramientas que permiten ir modificando a tiempo real la páginas.

1. Ubuntu

- Montaremos el servidor como instalamos ssh, con una interfaz que nos permite elegir lo que queremos instalar, que era llamando a tasksel. Entramos en modo root.

\$ tasksel

- Aparecerán todas las herramientas disponibles, y elegimos LAMP server. Descarga las 3 herramientas → MySQL, Apache y php.
- Nos pedirá una contraseña para SQL: practicas,ISE

¿Cuál es el problema de instalar LAMP por defecto desde Ubuntu?

Depende del uso que le demos, ya que si es una aplicación concreta, necesitará unas versiones y requisitos específicos y habrá que instalar y configurar ciertos aspectos de forma manual.

- ¿Cómo veo si está activo Apache?

\$ systemctl status apache2

→ Puerto 80

→ si tuviéramos el https, el puerto sería 443; si obtenemos un certificado de ssl, para blindar http con seguridad, tendríamos una página web segura, para eso es el puerto 443



- Vemos si MySQL está activo:

```
$ systemctl status mysql.service
```

LAMP nos instala tanto MySQL local para un uso más personal como MySQL en forma de servicio, porque los usuarios a lo mejor desde fuera quieren hacer peticiones a nuestra base de datos.

- Vemos si php está activo:

```
$ php -a
```

- si hemos conseguido entrar en el modo interactivo, es porque está activo y se ha instalado todo bien
- salimos con exit
- Comprobamos si está activo el servidor web y funcionando. Para ello, en el host entramos en el navegador con nuestra IP(192.168.56.120) y se nos muestra nuestra web estática, por defecto de Apache, que está activa y funcionando.
- Ya tendríamos nuestra página estática.

Si tenemos problemas:

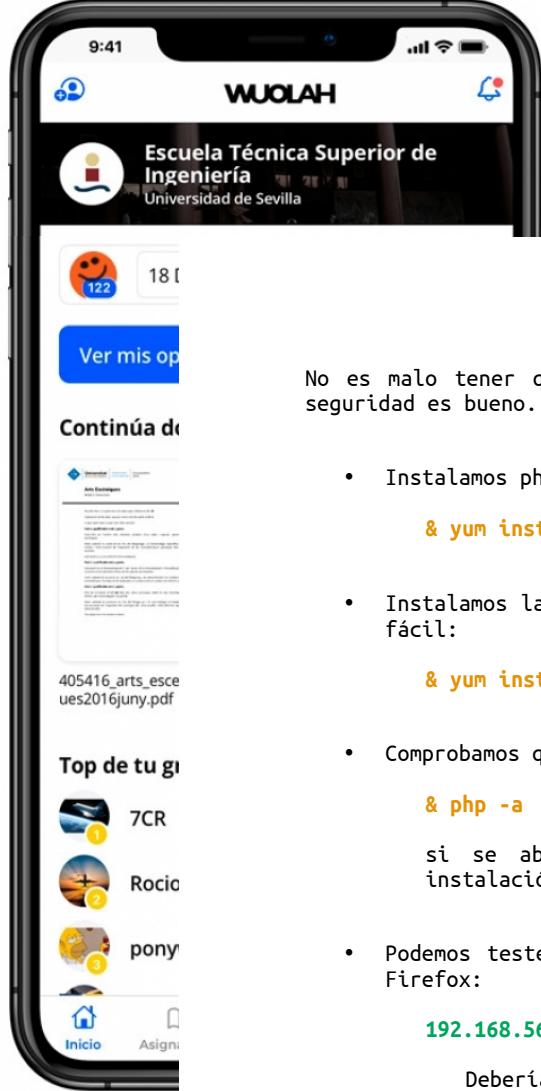
- Debemos ver si tenemos conexión del host con nuestra máquina virtual, esto lo vemos con ping 192.168.56.105 desde el host a la máquina virtual.
 - Es importante que las redes de Host-only de VirtualBox, tengan la IP 192.168.56.1. Si tenemos más de una y nos sobran, las borramos. Si tenemos varias para varias asignaturas, debemos asegurarnos que tenemos asignada la 56.1 para las máquinas de ISE.
- En la lección 2 estuvimos modificando el firewall de Ubuntu, con lo que quizás tengamos capado el puerto 80. Para que el firewall nos deje acceder al puerto 80:

```
$ ufw allow 80
```

2. CentOS

Entramos como usuario root. CentOS brinda más por la seguridad del sistema, tiene más restricciones, tiene todo capado por defecto. Esto a veces es bueno, el no dar las cosas por presupuestadas y hacerlas.

- Instalamos todo a mano, primero con Apache HTTPS
& yum install httpd
le damos a que sí
- Cómo vemos si está activo el servicio
& systemctl status httpd
debería salir running
- Por defecto, nos saldrá inactivo. Para activarlo:
& systemctl start httpd
& systemctl enable httpd
- Vamos a decirle al sistema que nos permita http. Para permitir puertos en CentOS:
& firewall-cmd --add-service=http
success
- Instalamos el gestor local de MySQL, pero se instala de otra forma que en Ubuntu:
& yum install mariadb
- Instalamos el servidor de mariadb:
& yum install mariadb-server
sí/yes
- Miramos si está mariadb alzado:
& systemctl status mariadb
- Para activar mariadb
& systemctl start mariadb
- Cuando instalamos mariadb-server, también se instala en el sistema un script `mysql_secure_installation`. Vamos a arrancarlo y nos hará una serie de preguntas orientadas a la seguridad de la base de datos:
& mysql_secure_installation
 - **contraseña_root:** enter, no tenemos
 - **modificar contraseña root:** Sí, practicas, ISE
 - **usuario anónimo que viene por defecto:** sí (lo capamos)
 - **deshabilitar conexión de root remotamente:** sí
 - **eliminar la base de datos por defecto:** sí, por seguridad
 - **queremos cargar los privilegios:** sí



Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.

Available on the App Store

GET IT ON Google Play

No es malo tener que activar los servicios cuando iniciamos el sistema, a nivel de seguridad es bueno. Además, hay servicios que no nos interesan tener activos.

- Instalamos php:

& **yum install php**

- Instalamos las funciones de php para poder acceder a las bases de datos de forma fácil:

& **yum install php-mysql**

- Comprobamos que php está instalado:

& **php -a**

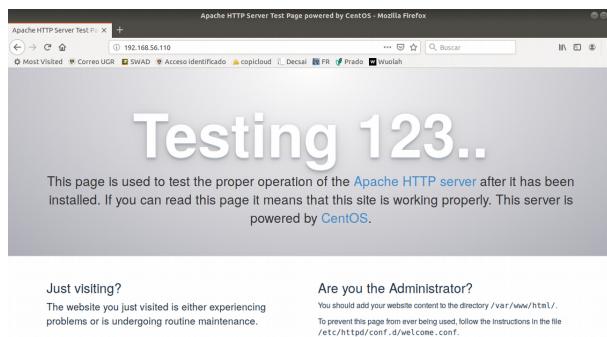
si se abre la consola interactiva, es que la instalación es correcta

```
[root@localhost ssh]# php -a
Interactive shell
php > _
```

- Podemos testear el index de php que ya tenemos en nuestro host. Si hacemos en Firefox:

192.168.56.110

Deberíamos ver una página estática testing 1, 2, 3.



- Aquí lo que hace internamente php es intentar conectarse a la base de datos. Si consigue tener conexión, si recibe respuesta de la base de datos, nos dice que está conectado; si no, nos dice que no. Esta es una forma de ver si todo está conectado.

Si no tenemos un navegador para verlo, ¿cómo podríamos comprobarlo?

& **curl 192.168.56.110**

→ si nos devuelve un html, es correcto, sería lo que veríamos en la pantalla pero en un fichero html

Si tuviéramos un servidor https, sería:

& **curl -k 192.168.56.110**

Si no tenemos navegador, desde la máquina virtual de Ubuntu podemos hacer un curl a CentOS. Si nos devuelve el html es porque todo ha ido bien.

- En www/html es donde se guardan todos los archivos de los servidores:
- Vamos a crear un fichero index de php y dentro escribimos:

```
& vi /var/www/html/index.php
```

```
<?php
$link=mysql_connect('127.0.0.1:3306', 'root', 'practicas,ISE');
if(!$link){

die('Could not connect:'. mysql_error());
}

echo 'Connected Successfully';
mysql_close($link);
php_info();

?>
```

Básicamente:

- nos creamos una variable link a la que le asignamos el valor de la función mysql_connect
- mysql_connect(localhost:puerto, usuario_root, contraseña_root)
 - localhost ya que vamos a llamar desde la propia máquina virtual;
 - puerto 3306, ya que es el puerto de mysql por defecto;
esta función nos devuelve una conexión.
- Si nos devuelve la conexión, no entrará en el if y confirmará la correcta conexión; pero si la conexión no se ha hecho, entrará al if y nos comunicará que la conexión ha fallado y el error.

- Para comprobar la sintaxis si nos da error:

```
& php /var/www/html/index.php
```

- Nos debería de dar un error de conexión a la base de datos, ya que a SELinux aún no le hemos especificado que php se pueda conectar a dicho puerto.
- Vamos a obtener todos los booleanos del sistema de seguridad de Linux en las que se establece qué y qué no se puede hacer. A nosotros nos interesa httpd:

```
& getsebool -a | grep httpd
```

a nosotros nos interesa httpd_can_network_connect_db, para que se pueda conectar a la base de datos

```
httpd_can_network_connect_cobbler --> off
httpd_can_network_connect_db --> off
httpd_can_network_memcache --> off
```

- Una vez tenemos el flag que queremos modificar, lo modificamos a on:

```
& setsebool -P httpd_can_network_connect_db on
```

- Para comprobar que se nos ha modificado, volvemos a hacer getsebool y debería estar httpd_can_network_connect a on:

```
& getsebool -a | grep httpd
```

```
httpd_can_network_connect_cobbler --> off
httpd_can_network_connect_db --> on
httpd_can_network_memcache --> off
```

- Para comprobar que tenemos index.php, hacemos:

```
192.168.56.110/index.php
```

y debe de mostrarlo.

- Debemos ir a la configuración de Apache y establecer por defecto index.php

```
& vi /etc/httpd/conf/httpd.conf
```

hacemos /SHIFT 7 y buscamos index.html. Una vez encontrado, añadimos el nombre index.php seguido de una coma → (index.html, index.php)

```
#
# DirectoryIndex: sets the file that Apache will serve if a directory
# is requested.
#
<IfModule dir_module>
    DirectoryIndex index.html, index.php
</IfModule>
```

- Para comprobarlo, podemos ir al navegador y ya nos sale nuestra página dinámica simple que hemos creado, en el que se nos muestra un sí si ha conectado con la base de datos y no si no lo ha hecho; además de la información de php del sistema

```
192.168.56.110
```

y debe de mostrarlo.

Con todo esto ya tenemos hecho los dos servidores tanto en Ubuntu como en CentOS.

Una herramienta útil para las prácticas:

TMUX (terminal multiplexor): esto crea de forma muy sencilla un entorno de terminales en las cuales podemos modificarlas de la forma que queramos, ponerle colores.

Esto es muy útil para cuando se trabaja con varias máquinas virtuales a la vez, para no equivocarnos cuando escribimos un comando donde no debemos. Nos permite tener terminales de diferentes colores para diferentes servidores, de forma que podemos saber en qué terminal estamos.

Tabla comparativa de repaso de esta práctica

	Ubuntu	CentOS
ssh	<ul style="list-style-type: none"> • No instalado por defecto • Debemos instalarlo • No debemos iniciararlo 	<ul style="list-style-type: none"> • Instalado por defecto • Y activado
SELinux	<ul style="list-style-type: none"> • No instalado 	<ul style="list-style-type: none"> • Instalado • Y activado
Firewall	<ul style="list-style-type: none"> • Ufw • Viene desactivado 	<ul style="list-style-type: none"> • Firewall • Viene activado
Fail2ban	<ul style="list-style-type: none"> • Instalado • pero desactivado 	<ul style="list-style-type: none"> • No instalado • no habilitado • no activado • hay que instalar antes el repositorio en el que se encuentra fail2ban
LAMP	<ul style="list-style-type: none"> • Instala todo: mySQL, Apache, PHP; • y viene activado 	
MySQL	<ul style="list-style-type: none"> • Instala el gestor local y el de servicio 	Mariadb: <ul style="list-style-type: none"> • hay que instalar el gestor local • hay que instalar el de servicio • hay que iniciarlos • ejecutar el script de seguridad
Apache		Httpd: <ul style="list-style-type: none"> • hay que instalarlo, • activarlo y • habilitarlo
Php		<ul style="list-style-type: none"> • Hay que instalarlo • hay que instalar php-mysql

Git: Chuleta de Comandos (git-chuleta-de-comandos)

07/11/2017

(<https://www.addtoany.com/share#url=https%3A%2F%2Fwww.lineadecodigo.es%2Fgit-chuleta-de-comandos&title=Línea%20de%20comandos>)
(/#facebook) (/#twitter) (/#whatsapp) (/#tuenti)

Artículo original (<https://elbauldelprogramador.com/mini-tutorial-y-chuleta-de-comandos-git/>)



NOTA: Puede que te interese descargar este manual para git (http://bashyc.blogspot.tradepub.com/c/pubRD.mpl?sr=oc&t=oc:&qf=w_java24&ch=ocsoc). Si rellenas el formulario correctamente nos darán \$1.5 de comisión, una buena forma de mostrar tu apoyo al blog ;-)

Una de mis tareas pendientes era aprender GIT decentemente. Así que empecé a leer Pro Git (<http://git-scm.com/book>), libro que recomiendo a todo desarrollador, disponible en PDF (<https://github.s3.amazonaws.com/media/progit.en.pdf>), EPUB (<https://github.s3.amazonaws.com/media/progit.epub>), MOBI (<https://github.s3.amazonaws.com/media/pro-git.en.mobi>) y versión en papel (http://www.amazon.es/gp/product/1430218339/ref=as_li_ss_tl?ie=UTF8&camp=3626&creative=24822&creativeASIN=1430218339&linkCode=as2&tag=elbaudelpro-21). En la página del libro puedes encontrar versiones en distintos idiomas. Conforme he ido leyendolo, he anotado los comandos. Como resultado he creado esta especie de chuleta de comandos git que comparto hoy con vosotros. Espero que os resulte útil.

- git help <command>
- git clone <uri> namedir # clona usando como nombre de directorio namedir.
- git add <dir> # añade recursivamente todos los archivos del dir.
- git diff --staged # compara staged changes with last commit
- git commit -v # muestra el diff en el editor
- git commit -a -m "mensaje" # automáticamente stage tracked files. No hace falta git add
- git rm --cached <file or regexp> # Git no realiza un seguimiento del archivo, pero los deja en el directorio de trabajo. Útil cuando se olvida añadir archivos al .gitignore y ya hemos agregado dichos archivos al repositorio.
- git rm <file> # borrarlos con git siempre.
- git rm -f <file> # si ya está modificado y en el index.
- git mv <file> <renamed_file>
- gith # tcl/tk. Herramienta gráfica para git
- git commit --amend # Modificar el mensaje del último commit
- git reset HEAD <file> # to unstaged
- git checkout -- <file> # Descartar cambios en el directorio de trabajo.

AÑADIR ARCHIVOS

- git add -i # interactive stagggin
- git add -p # crea patch

STASH

- git stash # guarda el estado en una pila y limpia el directorio para poder cambiar de rama
- git stash list # muestra la pila
- git stash apply # vuelve al estado original del dir. Stash{n} especifica uno concreto Y -index reaplica los cambios stagged
- git stash pop # elimina el primero en la pila. O drop

LOGS

- git log -p -2 # Muestra 2 últimos commits con diff
- git log --stat
- git log --pretty <short|full|fuller>
- git log --pretty=format:"%h - %an, %ar : %s"
- git log --pretty=format;"%h %s" --graph
- git log --since=2.weeks
- git log <branch> --not master # Muestra commit de <branch> sin incluir los de master
- git log --abbrev-commit --pretty=oneline
- git diff master...contrib # Muestra solo el trabajo que la rama contrib actual ha introducido desde su antecesor común con master
- git log <branch1>..<branch2> # Commits de branch2 que no están en branch1
- git log origin/master..master # Muestra qué commits se van a enviar al servidor
- git log origin/master.. # Igual que el anterior. Se asume master o HEAD
- git log refA refB --not refC # commits en refA y refB que no están en refC
- git log master...experiment # commits de master o experiment, pero sin ser comunes. Con -left-right indica a qué rama pertenece cada uno

REMOTES # repos en internet

- git remote -v # lista los repos remotos
- git remote add [shortname] [url] # crea nuevo remote, es posible descargar el contenido de ese repo con git fetch [shortname]. Master branch en [shortcode]/master
- git fetch <remote> # descarga trabajo nuevo a máquina local, no sobreescribe nada tuyo. (git pull sí hace merge automáticamente si se está realizando un seguimiento de esa branch)
- git push [remote-name] [branch-name] # si nadie ha hecho push antes
- git remote show [remote-name] # inspecciona remote.
- git remote rename <old-name> <new-name> # también renombra branches: quedaría <new-name>/master
- git remote rm <remote-name> # p.e si el contribuidor ya no contribuye más

Añadir varios repositorios remotos

- git remote add bitbucket <url repositorio> # Añadir un nuevo repositorio remoto con el nombre deseado. Por ejemplo si ya tenemos uno en github y queremos añadir otro para bitbucket
- git push -u bitbucket -all # Subir el proyecto a bitbucket. A partir de ahora se puede seleccionar a qué repo publicar con git push nombre_repo_remoto

TAGGING # marcan puntos importantes en la historia del repo (releases)

- git tag # muestra las etiquetas actuales
- git tag -l 'v1.4.2.*' # acepta regex
- Dos tipos de tag:
 - **Lightweight** : puntero a commit (branch que no cambia)
 - **Annotated** : se almacenan como objetos en la db, con checksum, nombre del creador, email, fecha, mensaje, posibilidad de firmarla con GPG (<https://elbauldelprogramador.com/como-cifrar-correos-con-gpg-con-mailvelope/>). (recomendada)
- git tag -a <tagname> -m 'mensaje' # annotated tag
- git show <tag-name> # muestra información asociada.
- git tag -s <tag-name> -m 'message' # la firma con gpg
- git tag <tag-name> # lightweight tag
- git tag -v <tag-name> # verifica tags firmadas
- git tag -a <tag-name> [commit-chksum] # crea tag para commit con dicho chksun
- Por defecto no se transfieren los tags, para subirlos al servidor:
 - git push origin [tag-name] # una sola
 - git push origin --tags # Enviar todas
- Para usar GPG y firmar tags, hay que subir la clave pública al repositorio:
 - gpg --list-keys # Coges la id pública
 - gpg -a --export <id> | git hash-object -w --stdin # Copia el SHA-1 devuelto
 - git tag -a maintainer-gpg-pub <SHA-1>
 - git push --tags # Comparte la clave con todos los usuarios
 - git show maintainer-gpg-pub | gpg --import # Cada usuario importa la clave así
 - git show <tag> # Devuelve más información sobre la etiqueta
 - git tag -d nombre_tag # eliminar la etiqueta
 - git push origin :refs/tags/nombre_tag # Eliminar la etiqueta del repositorio remoto.

BRANCH

Las ramas simplememte son punteros a distintos snapshots

- git branch <nombre-rama> # crea rama. Puntero al commit actual
- git checkout <nombre-rama> # cambiar a la rama especificada.
- git checkout -b <nombre-rama> # crea y cambia de rama
- git merge <rama> # Mezcla la rama actual con <rama>
- git branch -d <rama> # elimina la rama
- git push origin --delete <branchName> # Elimina una rama del servidor
- git mergetool # Herramienta gráfica para resolver conflictos
- git branch # lista ramas
- git branch -v # lista ramas mostrando último commit
- git branch --merged # lista ramas que han sido mezcladas con la actual. Si no tienen un *, pueden borrarse, ya que significa que se han incorporado los cambios en la rama actual.
- git branch --no-merged # lista ramas que no han sido incorporadas a la actual.

REMOTE BRANCHES

- git fetch origin # Descarga el contenido del servidor
- git push <remote> <branch> # Las ramas no se suben por defecto, has de subirlas explícitamente
- git push <remote> <branch>:<nuevoNombre> # Igual que la de arriba, pero en el servidor se llama a la rama con nuevoNombre en lugar de branch
- Cuando se hace un git fetch que trae consigo nuevas ramas remotas, no se disponen de ellas localmente, solo se dispone de un puntero a la rama remota que no es editable. Para poder trabajar sobre esa rama, es necesario crearla Por ejemplo:
 - git fetch origin # Tras ejecutarlo, notamos que se ha creado una rama nueva (rama_nueva)
 - git checkout -b rama_nueva origin/rama_nueva # Crea una rama local a partir de la remota
 - git merge origin/nueva_rama # Equivalente a la de arriba, pero sin establecer el tracking a la rama
- git push [remotename] :[branch] # elimina una rama remota
- git push [remotename] [localbranch]:[remotebranch] # La rama en el servidor tiene distinto nombre a la local

TRACKING BRANCHES

- git checkout --track origin/rama # Equivalente a -b rama_nueva origin/rama_nueva
- git chekout -b <nuevo_nombre> origin/<rama> # Establece un nombre distinto para la rama local

REBASE

Rebase y merge se diferencian en que merge mezcla dos puntos finales de dos snapshots y rebase aplica cada uno de los cambios a la rama en la que se hace el rebase. No lo uses en repos publicos con mas colaboradores, porque todos los demás tendrán que hacer re-merges

- git checkout <una rama>
- git rebase master # aplica todos los cambios de <una rama> a master
- git merge master # hay que hacer un merge de tipo fast forward

- Tenemos 3 ramas, master, client y server, en server y client tenemos varios commit y queremos mezclar client en master pero dejar server intacta:
 - git rebase --onto master server client # adivina los patches del antecesor común de las ramas server y client y aplica los cambios a master.
 - git checkout master
 - git merge client # fast-forward. Client y master en el mismo snapshot
- Si se quiere aplicar también los cambios de server, basta con:
 - git rebase master server
 - git checkout master
 - git merge server
- git rebase [basebranch] [topicbranch] # sintaxis de rebase
- git rebase -i # Rebase interactivo

SERVIDOR

- git instaweb # Muestra una interfaz web con los commits

GENERAR UN NÚMERO DE COMPILACIÓN (BUILD NUMBER)

- git describe master # Solo funciona para tags creadas con -s ó -a

PREPARAR UNA RELEASE

- git archive master --prefix="project/" | gzip > \$(git describe master).tar.gz
- git archive master --prefix="project/" --format=zip | \$(git describe master).zip
- test/ export-ignore # Al crear el tarball no incluye el directorio test/

GENERAR UN CHANGELOG

- git shortlog --no-merges master --not <tag> # Recopila todos los commits desde <tag> y los agrupa por autor

RECOMENDACIONES

- Siempre hay que hacer pull antes de push en caso de que alguien haya subido cambios al servidor. Ejemplo:
 - User1 clona el repo y hace cambios, realiza un commit
 - User2 clona el repo, hace cambios, hace commit y sube los cambios con push
 - User1 intenta hacer push, pero será rechazado con: ! [rejected] master -> master (non-fast forward). No puede subir los cambios hasta que no mezcle el trabajo que ha subido User2. Así que debe hacer lo siguiente:
 - git fetch origin
 - git merge origin/master
 - git push origin master
- Mientras User1 hacía estas operaciones, User2 ha creado una rama issue54 y realizado 3 commits, sin haber descargado los cambios de User1. Para sincronizar el trabajo, User2 debe hacer:

- git fetch origin
- git log --no-merges origin/master ^issue54 # Observa qué cambios ha hecho User1
- git checkout master
- git merge issue54 && git merge origin/master
- git push origin master
- git diff --check # Antes de hacer commit, ejecutar esto para ver si hemos añadido demasiados espacios que puedan causar problemas a los demás.
- Commits pequeños que se centren en resolver un problema, no commits con grandes cambios.
- git add --patch # En caso de hacer varios cambios en el mismo archivo
- El mensaje del commit debe tener la estructura siguiente: Una linea de no más de 50 caracteres, seguida de otra línea en blanco seguida de una descripción completa del commit.

PASOS A SEGUIR PARA CONTRIBUIR A PROYECTOS AJENOS, MEDIANTE FORK

- git clone <url>
- git checkout -b featureA
- git commit
- git remote add myFork <url>
- git push myFork featureA
- git request-pull origin/master myFork # enviar la salida por mail al propietario del proyecto, o hacer click en pull request.
- Buena practica tener siempre una rama master que apunte a origin/master, para estar siempre actualizado con los ultimos cambios en el proyecto original.
- **Separar cada trabajo realizado en topic branch, que trackeen a origin/master**
 - git checkout -b featureB origin/master
 - (Hacer cambios)
 - git commit
 - git push myFork featureB
 - (Contactar con el propietario del proyecto)
 - git fetch origin
 - Otro ejemplo, el propietario del proyecto quiere aceptar un pull tuyo, pero quiere que hagas algunos cambios, aprovechas la oportunidad y mueves tu trabajo para basarlo en el contenido actual de la rama origin/master, aplastas los cambios en **featureB**, resuelves conflictos, y haces push:
 - git checkout -b featureBv2 origin/master
 - git merge --no-commit --squash featureB
 - (cambiar la implementacion)
 - git commit
 - git push myFork featureBv2
 - --squash coge todo el trabajo de la rama mezclada y la aplasta en un no-merge commit encima de la rama en la que estas. -no-commit no registra el commit automaticamente.
 - Así puedes realizar todos los cambios necesarios y luego hacer el commit

REFLOG

últimos meses.

- git reflog
- git show HEAD@{n} # Muestra información sobre el reflog número n
- git log -g master # Muestra el log formateado como la salida de reflog
- git show master@{yesterday} # Muestra los commits de ayer.

UTILIDADES

- git show <short-SHA-1> # Es posible ver un commit pasando la versión abreviada del SHA-1
- git rev-parse <branch> # A qué SHA-1 apunta una rama
- git show HEAD^ # Muestra commit padre
- git show HEAD^2 # Muestra segundo padre
- git show HEAD~2 # El primer parent del primer parent
- git filter-branch --tree-filter 'rm -f <file>' HEAD # elimina el archivo de todos los commits

DEPURACIÓN

- File annotation
 - git blame -L 12,22 <archivo> # muestra cuando y por quién se modificaron de la linea 12 a la 22
 - git blame -C -L 141,153 <file> # cuando renombras un archivo o lo refactorizas en varios, muestra de donde vino originalmente.
- Búsqueda Binaria: Cuando hay un bug que no puedes localizar, usas bisect para determinar en qué commit empezó a producirse el bug.
 - git bisect start
 - git bisect bad # marcas el commit actual como roto
 - git bisect good [commit bueno] # último commit conocido que funcionaba
- Ahora irá preguntando hasta que encuentres el commit culpable. Si estás bien indica git bisect good. De lo contrario git bisect bad. Al terminar hay que resetear.
 - git bisect reset

SUBMODULOS

- git submodule add <url> # crea un directorio que contiene el contenido de otro proyecto.
- Clonar un repo con submodulos
 - git clone url
 - git submodule init
 - git submodule update

CONFIGURACION

- git config --global <opcion> <valor> # global para usuario, system todos y sin nada, específico para el repo.
- git config {key} # muestra el valor de key
- git config --global core.editor <editor> # cambia el editor por defecto
- git config --global commit.template \$HOME/.gitmessage.txt # plantilla para commits
- git config --global core.pager 'more|less' # paginador por defecto, puedes usar cualquiera
- git config --global user.signingkey <gpg-key-id> # clave gpg para firmar tags

- git config --global core.excludesfile <file> # como gitignore
- git config --global help.autocorrect 1 # autocorre cuando se escribe un comando incorrecto. Solo en git >= 1.6.1
- git config --global color.ui true # colorea la salida de git. Valores: true|false|always
- git config --global core.autocrlf input # para que usuarios linux no tengan problemas con los retornos de carro de windows
- git config --global core.autocrlf true # para usuarios de windows
- git config --global core.whitespace trailing-space, space-before-tab, indent-with-non-tab, cr-at-eol # respectivamente: busca espacios al final de línea, busca espacios al inicio de tabulación, busca líneas con 8 o más espacios en lugar de tabulaciones, acepta retornos de carro
- git apply --whitespace=warn <patch> # advierte de errores de espacios antes de aplicar el patch. Con --whitespace=fix intenta arreglarlos

GIT ATTRIBUTES

Archivo en .gitattributes en el directorio de trabajo o en .git/info/attributes para no committearlo

Identificando archivos binarios Muchos archivos son para uso local y no aportan información al repositorio. Para decirle a git qué archivos son binarios hacer añadir al archivo attributes:

- <nombre archivo o regexp> -crlf -diff # git no intentará corregir problemas de crlf ni mostrará los cambios con diff. En versiones >= 1.6 se pueden sustituir estos dos valores por la macro binary

Diffing binary files

En ocasiones es útil mostrar diffs de archivos binarios, como una archivo de word:

***.doc diff=word**

tras esto hay que definir el filtro word para que git convierta archivos word a texto:

- git config diff.word.textconv strings

Es posible hacer lo mismo para imágenes jpeg, es necesario instalar **exiftool** para extraer los metadatos y luego hacer:

```
$ echo '*jpeg diff=exif' » .gitattributes
$ git config diff.exif.textconv exiftool
```

Procesar archivos antes de hacer commit y antes de hacer checkout: Es posible crear tus propios filtros para hacer sustitución. Estos filtros se llaman smudge (<https://elbauldelprogramador.com/como-usar-los-filtros-smudge-y-clean-en-git/>) y **clean**. Los puedes configurar para distintos directorios y luego escribir un script que procesará cada archivo antes de que sea checkeado (smudge) y commiteado (clean) (<https://elbauldelprogramador.com/como-usar-los-filtros-smudge-y-clean-en-git/>). Para ello, escribe en el .gitattributes: (En caso que quieras procesar código C)

***.c filter=indent** Luego:

- git config --global filter.indent.clean indent
- git config --global filter.indent.smudge cat

ruby que recibe un archivo, encuentra la fecha de su último commit e inserta dicha fecha en el archivo:

```
#! /usr/bin/env ruby
data = STDIN.read
last_date = `git log --pretty=format:"%ad" -1`
puts data.gsub('$Date$', '$Date: ' + last_date.to_s + '$')
```

Puedes nombrar este script como **expand_date**. Crea un filtro en git, llamado dater y dile que use el script anterior:

- git config filter.dater.smudge expand_date
- git config filter.dater.clean 'perl -pe "s/\\$\\$Date[^\\\\$]*\\\\$/\\$\\$Date\\\\$/"'

Para usar el filtro, simplemente escribe la palabra clave en los archivos que deseas:

```
echo '# $Date$' > date_test.txt
echo 'date.txt filter=dater' » .gitattributes

git add date_test.txt .gitattributes
git commit -m "Testing date expansion in Git"
rm date_test.txt
git checkout date_test.txt
cat date_test.txt
$Date: Tue Apr 21 07:26:52 2009 -0700$
```

GIT HOOKS

Hay dos tipos (<https://elbauldelprogramador.com/sincronizacin-de-proyectos-en-git-con-hooks-ganchos/> "Sincronización de proyectos en git con hooks (ganchos)"), de lado cliente y servidor, se guardan en el directorio .git/hooks. Para activarlos basta con que sean ejecutables.

CONCEPTOS

Fast forward: cuando se hace un merge y el commit de la rama a mezclar está justo un commit adelantado, simplemente se hace apuntar la rama en la que se iba a mezclar al commit del merge.

GITIGNORE:

```
*.a # no .a files
*!lib.a # but do track lib.a, even though you're ignoring .a files above
/TODO # only ignore the root TODO file, not subdir/TODO*
build/ # ignore all files in the build/ directory*
doc/*.txt # ignore doc/notes.txt, but not doc/server/arch.txt
```

Práctica 2

Lección 1: SSH

• Gestores de paquetes

- **yum:** Es el gestor de paquetes utilizado por las distribuciones basadas en RedHat, en este caso, lo utiliza CentOS 7 y 8
 - Sintaxis básica
 - \$yum <opción>
 - Instalar paquetes.
 - \$yum install <paquete>
 - Eliminar paquetes.
 - \$yum remove <paquete>
 - Búsquedas de paquetes en los repositorios.
 - \$yum search <cadena>
 - Para saber que bibliotecas se proporcionan en un paquete.
 - \$yum provides <cadena>
- El sucesor de yum se llama **DNF** y en **CentOS 8**, el comando yum lo que hace es redirigir los parámetros al comando **DNF**. Se mantiene por retrocompatibilidad.
- En **CentOS 7**, DNF no existe, no ha sido implementado. Utiliza yum propiamente.
- **apt:** Es el gestor de los sistemas basados en Debian, es decir, Debian, Ubuntu, Mint, etc.
 - Utilizar el comando \$apt-get está en desuso, se recomienda utilizar \$apt solamente.
 - Instalar paquetes
 - \$apt install <paquete>
 - Eliminar paquete
 - \$apt remove <paquete>
 - Buscar paquetes
 - \$apt search <cadena>
 - Eliminar paquetes innecesarios
 - \$apt autoclean
- El sucesor de **apt** y la idea de Canonical, era converger en **snaps**. Un único sistema de paquetes para todas las distribuciones de Linux. Ya está en uso en Ubuntu 20.
 - Una ventaja grande de los **snaps**, es que tienen las dependencias incorporadas. Se resuelven automáticamente.
 - **RedHat** se ha opuesto a utilizar **snaps**, han seguido con **DNF**.
 - El resto de las distribuciones está por verse si siguen a Canonical o no.
- Es posible utilizar un asistente de instalación de paquetes gráfico en Ubuntu, llamado **tasksel**. No viene por defecto.
 - Sigue siendo recomendado utilizar **apt**, ya que provee más información a la hora de instalar paquetes. No indica versiones y también inicia servicios sin el consentimiento del usuario, posibles problemas de seguridad.

• ¿Por qué no utilizar interfaces gráficas?

- De manera prioritaria, las interfaces también pueden tener **huecos de seguridad**, por lo que utilizarla añade más riesgo de que el servidor sea comprometido.
- **Bajan el rendimiento del sistema**, porque se tienen que asignar recursos del servidor a la interfaz. Esto resulta un coste innecesario ya que rara vez los servidores tienen conectada una pantalla.
- No es bueno fiarse de nada: Filosofía del **Zero Trust Model**, se debe verificar todo lo que se instala y qué hace en un sistema. **Cuanto menos se tenga en ejecución, mejor**.

• SSH (Secure SHell)

- Desarrollado en la universidad técnica de Helsinki (TKK) como un protocolo de comunicación seguro diseñado como un reemplazo del protocolo inseguro telnet.
- CentOS trae por defecto SSH instalado, en cambio, Ubuntu Server da la opción de instalarlo o no.
- Importante distinción: SSH como servicio y SSH como cliente

- diseñado como un reemplazo del protocolo inseguro telnet.
- CentOS trae por defecto SSH instalado, en cambio, Ubuntu Server da la opción de instalarlo o no.
 - Importante distinción: SSH como servicio y SSH como cliente.
 - **SSH como cliente** es el que inicia la conexión, se llama **ssh**.
 - **SSH como servicio** es quien recibe la conexión, y es un demonio del sistema, se llama **sshd**.
 - Los archivos de configuración son diferentes, hay que saber cuál cambiar
 - /etc/ssh/ssh_config es el cliente
 - /etc/ssh/sshd_config es el servicio
 - Funcionamiento
 - Sintaxis
 - \$ssh <direccionIP>
 - \$ssh <usuario>@<direccion>
 - Como cliente, al conectarse por primera vez a una dirección nueva, el servidor tiene una fingerprint que se añade a la lista de hosts conocidos.
 - Esto es útil ya que si se intenta suplantar el nombre de dominio o IP, a través del fingerprint se puede detectar.
 - Configuración esencial de SSH
 - **Deshabilitar el acceso de root para SSH**
 - \$vi /etc/ssh/sshd_config
 - Cambiar "PermitRootLogin" a "no"
 - Reiniciar el servicio para que se apliquen los cambios
 - \$systemctl restart sshd.service
 - **Cambiar el puerto de escucha de SSH**
 - Por defecto, utiliza el puerto 22. Cambiar, por ejemplo, 22322
 - \$vi etc/ssh/sshd_config y se edita el puerto al deseado.
 - No se puede cambiar directamente en /sshd_config en un sistema SELinux, se tiene que utilizar otro comando primero
 - \$semanage port -a -t ssh_port_t -p tcp 22322
 - Si semanage no se encuentra, se debe buscar un paquete que lo posea.
 - Luego, si se edita el archivo sshd_config
 - Se tiene que dar permiso al cortafuegos también, para ello se usa.
 - \$firewall-cmd --add-port=<numeroPuerto>/<protocolo>
 - Lo abre en esta sesión, pero para que sea permanente tiene que ser:
 - \$firewall-cmd --permanent --add-port=<numeroPuerto>/<protocolo>
 - \$systemctl restart sshd.service
 - **Entrar seguramente al servidor sin usar contraseña**
 - Generar el par de llaves pública y privada
 - \$ssh-keygen
 - Importante: Quien tenga la llave privada, tiene acceso completo. Hay que tener mucho cuidado de dónde se almacena la llave privada.
 - Una vez generada, lo que se puede distribuir es la llave pública, id_rsa.pub
 - Enviar la llave pública al servidor
 - Se puede realizar por diferentes métodos (USB, SFTP, SCP); pero existe un comando particular de SSH para ello.
 - \$ssh-copy-id <usuario>@<direccionIP> [-p <puerto>]
 - Una vez realizada la copia, se puede ingresar al servidor colocando solamente la clave para la llave SSH o si se ha dejado sin clave, entrar sin necesidad de contraseña.
 - Deshabilitar entrada por contraseña
 - \$vi /etc/ssh/sshd.config
 - Cambiar PasswordAuthentication a 'no'
 - \$systemctl restart sshd.service
 - Si se tiene configurado que solamente autentique por llaves, un usuario nuevo no podrá añadir su llave pública a menos que se reestablezca temporalmente el acceso por contraseña. Esto se puede realizar mediante un script.
 - Cortafuegos
 - Ubuntu Server: ufw
 - Firewall



○ Cortafuegos

- Ubuntu Server: ufw
- Funcionalidades
 - Estado del Firewall
 - \$ufw status
 - Activar el firewall
 - \$ufw enable
 - Dar paso a un puerto
 - \$ufw allow <numeroPuerto>
- CentOS8: firewall-cmd
- Funcionalidades
 - Estado del firewall
 - \$firewall-cmd --state
 - Dar paso a un puerto en la sesión actual
 - \$firewall-cmd --add-port=<numeroPuerto>/<protocolo>
 - Dar paso a un puerto de manera permanente
 - \$firewall-cmd --permanent --add-port=<numeroPuerto>/<protocolo>

○ Evitar que un proceso muera si se pierde la conexión al servidor remoto

- Se debe hacer uso de las aplicaciones screen y tmux.
- screen:
 - Cuando se entra en un servidor remoto, si se usa en bash el comando, se inicia un nuevo terminal que internamente cuelga del proceso SCREEN. Si sucede una desconexión, el proceso SCREEN se descuelga del bash padre y de esta manera los trabajos que se estaban realizando no se detienen.
 - Para manualmente descolgar el proceso se hace Ctrl+A y D
 - Una vez reestablecida la conexión, se vuelven a añadir los procesos con el comando
 - \$screen -list
 - \$screen -r <identificador>
- tmux:
 - De la misma manera que funciona screen, se arranca en una terminal haciendo
 - \$tmux
 - Dentro de la misma se pueden realizar trabajos y procesos, si por alguna razón se desconecta o bien se descuelga el proceso, este seguirá en el fondo ejecutándose
 - Para descolgar manualmente se hace Ctrl+B y D
 - Para volver a entrar a la sesión se utiliza
 - \$tmux ls
 - \$tmux attach-session -t <numero>

○ Diferencias en las instalaciones de SSH en CentOS8 y Ubuntu Server

Ítem	CentOS8	Ubuntu Server
¿Puede iniciar sesión root por defecto?	Sí	No
¿Instalado por defecto el servicio por defecto?	Si	Opcional
Puerto por defecto de SSH	22	22
¿Firewall?	(firewall-cmd) Activo	(ufw) Desactivado

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.



● Seguridad del Servidor

○ fail2ban

- Una vez instalado se verifica que el servicio está corriendo con
 - \$systemctl status fail2ban
- Si aparece inactivo, se tiene que configurar para que cuando inicie la máquina, se active el servicio también
 - \$systemctl enable fail2ban
 - \$systemctl start fail2ban
- Configuración
 - Se realiza una copia del archivo /etc/fail2ban/jail.conf
 - \$cp -a /etc/fail2ban/jail.conf /etc/fail2ban/jail.local

Descarga, comparte, pregunta, aprueba.



- Configuración
 - Se realiza una copia del archivo /etc/fail2ban/jail.conf
 - \$cp -a /etc/fail2ban/jail.conf /etc/fail2ban/jail.local
 - Modificar parámetros de la cárcel
 - \$vi /etc/fail2ban/jail.local
 - Se busca en el archivo la parte de SSH, denominada como [sshd]
 - Se añade la sentencia enabled = true
 - Cambiar port = <puertoUsado>
 - Reiniciar el servicio con
 - \$systemctl restart fail2ban.service
 - Listar las cárceles que se tienen
 - \$fail2ban-client status
 - Listar detalles una cárcel en particular
 - \$fail2ban-client status <nombreCarcel>
- Para desbanear una IP
 - \$fail2ban-client set <nombreCarcel> unbanip <IP>

Lección 2: Git y Respaldos

• Copia de seguridad y Backups

- Es recomendado que el servidor de respaldos se encuentre alojado, principalmente en España o en todo caso dentro de la Unión Europea, para garantizar que los datos están protegidos por la Ley de Protección de Datos (España: LOPD) y (UE: RGPD). En última instancia, se debería acudir a los países que posean tratados con la UE que garanticen que los datos sean tratados correctamente.
- ¿Diferencia entre un backup y una copia?
 - Un respaldo/backup es una copia que posee metadatos sobre lo que se ha copiado, como una traza de versiones, cuando se hizo, que fecha, elegir entre diferentes copias ya hechas.
- Métodos para mitigar tasas de transferencias altas al realizar respaldos
 - Se desean evitar que la realización del respaldo tome más tiempo que la frecuencia que hay entre cada realización del mismo.
 - **Empaquetar** los archivos para que las transferencias necesiten de menos handshakes, ya que esto genera overhead.
 - **Comprimir** los archivos y empaquetarlos.
 - **Cacheado:** Si se trata de un servidor aparte que posee el backup, se puede hacer uso de un servidor intermedio de caché/proxy/búfer que transfiera de modo encolado los archivos respaldados.
 - **Backups incrementales:** Solamente se transfieren los datos que han sido modificados en un cierto momento, así se aligera el uso de red y los tiempos de transferencia.
- Es fundamental que no solamente sea la tasa de envío al backup razonablemente rápido, sino también que la tasa de recuperación de la información lo sea de igual manera.
 - Ejemplo: El por qué hacer /var_OLD en la práctica 1
 - De esta manera, si ocurriese algún problema con /var, con un simple comando mv se tendría la configuración inicial.

○ Copia de seguridad con dd

- Históricamente con este programa se realizaban los respaldos.
- Es útil actualmente para dispositivos de almacenamiento pequeños, es preferible copiar el contenido de una tarjeta o dispositivo, se busca en la copia en vez de en el dispositivo.
- Permite hacer copias, pero la parte de metadatos se tendría que añadir manualmente.
- Ventaja: Copia bit a bit, muy exacta.
- Problema: Los metadatos de los datos copiados se pierden.
- Funcionamiento
 - Copiar a otro dispositivo
 - \$dd if=/dev/sda of=/dev/sdb
 - **If:** Input File, **of:** Output File
 - Generar una imagen de disco
 - \$dd if=/dev/sda of=~/hdadisk.img
 - Recuperar la imagen de disco



GET IT ON
Google Play

Download on the
App Store

WUOLAH



- Generar una imagen de disco
 - \$dd if=/dev/sda of=~/hdadisk.img
- Recuperar la imagen de disco
 - \$dd if=hdadisk.img of=/dev/sdb
 - Se pueden especificar particiones también

○ Copia de seguridad con cpio

- Toma el listado de la entrada estándar y empaqueta los archivos que se encuentran en el listado en un solo archivo, más no lo comprime.
 - Empaquetar
 - \$ls <directorio> | cpio -ov > <archivo.cpio>
 - -o : Crear; -v : Verbose
 - Desempaquetar
 - \$cpio -iv <archivo.cpio>
 - -i : Extraer; -v : Verbose
- Es útil para transferir archivos en FTP/SFTP para transferir archivos ya que por cada archivo se debe realizar un handshake de TCP y por lo tanto es mejor que sea un solo bloque empaquetado en vez de los bloques de los archivos individuales.

○ Copia de seguridad con tar

- Sucesor de cpio
- En vez de tomar la entrada estándar, se le especifican los archivos a empaquetar.
- Posteriormente, se le añade la funcionalidad de comprimir (gz)
- Se recomienda utilizar en vez de gzip, el sistema de compresión Bzip2 (cambiar la z por j), esto reduce el espacio pero tarda más tiempo.
- Si bien tar no posee capacidades de red, se puede utilizar en conjunto con netcat o ssh para enviar ficheros a otras computadoras.
- Funcionamiento
 - Comprimir un archivo
 - \$tar cvzf archivoCompr.tar.gz /home/secretos
 - Create Verbose Zipped File
 - Descomprimir un archivo
 - \$tar -xvf fotosGuays.tar.gz
 - eXtract Verbose File

○ Copia de seguridad con rsync

- Permite la sincronización entre una fuente y un destino, inclusive por medio de SSH.
- Permite realizar copias de seguridad incrementales y también empaquetación
- Tiene la **posibilidad de borrar archivos** también, dado que se desearía tener esa clase de sincronización entre fuente y destino, pero puede ser peligroso.
- Funcionamiento
 - Sincronizar de un fichero fuente a uno destino
 - \$rsync -a -delete /source /destination
 - Sincronización con -delete: borra archivos
 - \$rsync -aviz /origen usuario@maquina:destino
 - Para restaurar, se invierte /origen y /destino.
 - En un destino remoto
 - Si echo \$RSYNC_RSH=ssh
 - \$rsync /origen username@maquina:/destino
 - Sino
 - \$rsync -e "ssh" /origen username@maquina:/destino
 - Opciones Comunes
 - -a : archivar
 - -r : Recursivo
 - -l : Enlaces simbólicos
 - -t : Conservar fecha
 - -p : Conservar permisos
 - -o : Conservar propietario
 - -g : Conservar grupo
 - -D : Archivos especiales



- -o : Conservar propietario
 - -g : Conservar grupo
 - -D : Archivos especiales
 - -v : Modo verbose
 - -z : Comprime
 - -i : Muestra resumen final
 - -e : Tipo de shell remota, SSH o RSH
 - Utilizar origen/ y origen/. copiará los archivos ocultos (aquellos iniciados en .) pero origen/* no lo hará.
 - Utilizar origen copiará la carpeta entera, origen/ copia los contenidos de la carpeta.
- ¿Qué usan los profesionales?
- Tartarus, Backup2I, Duplicity y Sftpclone.
- Control de Versiones o Cambios
- La versión más básica y fundamental sería utilizar cp cambiando el nombre del fichero
 - \$cp /etc/config1 /etc/config.old
 - Se puede comentar los archivos de configuración para tener un orden
 - Git es la herramienta por excelencia de control de versiones
 - Específicamente hecho para trabajar con archivos de texto plano, como código. No funciona bien con archivos binarios como imágenes o videos.
 - ¿Cómo funciona?
 - Se tiene del **directorio de trabajo**, la zona del **stage**, la zona del **commit** y el **repositorio**.
 - El **directorio de trabajo** es donde tenemos nuestro código.
 - En el **stage** es la zona dónde se registran los cambios, aquí se añaden los ficheros que deseamos seguir.
 - En la zona del **commit**, es donde los cambios se vuelven permanentes.
 - Finalmente se realiza un **push** del **commit** hacia el repositorio.
 - Siempre que se hace un **commit**, se tiene un hash que lo identifica y el último **commit** es apuntado por HEAD, de aquí podemos movernos relativos a HEAD.
 - Internamente
 - Los ficheros y los inodos se almacenan como Blobs (Binary large objects)
 - Los árboles son los directorios
 - Se podría decir que git es en cierta forma un sistema de archivos direccionable por contenido
 - Funcionamiento
 - Creación de un repositorio tipo "bare"
 - \$git init [nombre repo] --bare
 - Esto quiere decir que el repositorio no tiene Directorio de Trabajo, dado que si es un servidor que lo hostea, no tiene necesidad de uno, solo requiere los datos del repositorio.
 - Si este repositorio se elimina por error, se pierde toda la información del control de versiones.
 - Clonar un repositorio
 - \$git clone <ip>:<rutaAlRepo>
 - \$git clone ssh://<usuario>@<ip>:<puerto><rutaAlRepo>
 - Se clonará creando una nueva carpeta en la ruta que se encuentra instanciada la terminal.
 - Saber el estado del directorio de trabajo
 - \$git status
 - Añadir archivos en el directorio de trabajo para que sean seguidos
 - \$git add <archivo1> [archivoN]
 - Se pueden usar wildcards para filtrar archivos
 - Se pueden omitir archivos utilizando el fichero .gitignore
 - Hacer un commit
 - \$git commit [-a] -m "mensaje" [-m "mensaje cuerpo"]
 - Los commits deben ser los más atómicos posibles, cuanto más pequeños mejor, para tener una trazabilidad, para poder encontrar luego problemas más fácilmente.
 - Una vez hecho un commit, es inmutable.

Los commits deben ser los más detallados posibles, usando trazabilidad, para tener una trazabilidad, para poder encontrar luego problemas más fácilmente.

- Una vez hecho un commit, es inmutable.
 - \$git commit -amend
 - Para modificar el mensaje
 - \$git commit -amend -a
 - Para añadir un archivo adicional
- -a es para añadir todos los archivos modificados si no se añadieron antes.
- Ver modificaciones en los archivos
 - \$git diff
 - Si el archivo no está siendo seguido (No está en el stage) y se modifica y se añade, diff no dará resultado.
 - Si el archivo está siendo seguido, modificarlo mostrará la diferencia entre el directorio de trabajo y el stage.
 - \$git diff --staged
 - Muestra las diferencias entre un archivo que está en el Stage y lo que se encuentra en el repositorio
 - \$git diff HEAD
 - Muestra las diferencias entre un archivo en el directorio de trabajo y lo que está en el repo.
- Restaurar un archivo antes de realizar un commit
 - \$git checkout HEAD <archivo>
 - Restaurará a la última versión que se encuentra en el repo el archivo especificado.
- Restaurar un archivo luego de un commit
 - \$git reset <7 primeros caracteres del commit SHA>
 - Restaurará el directorio a como se encontraba en ese commit.
 - \$git reset --hard <caracteresSHA>
 - Restaura y elimina los commits posteriores, reposiciona HEAD en ese commit.
 - \$git revert <caracteresSHA>
 - Realiza lo mismo, y produce un nuevo commit. Se puede evitar incluyendo -n.
- Recuperar información del repositorio remoto
 - \$git pull
 - Es equivalente a realizar \$git fetch y \$git merge
- Birfurcando las ramas, para tener diferentes versiones del repo
 - \$git branch <nombreRama>
- Para cambiar de rama
 - \$git checkout <nombreRama>
 - \$git stash
 - Para guardar los cambios sin tener que hacer un commit.
 - \$git stash apply
 - Aplica el ultimo stash creado.
- Para unir ramas y resolver conflictos
 - \$git merge

○ Para utilizarlo en un servidor

- Protocolos
 - Se pueden utilizar varios y al mismo tiempo
 - Protocolo local para una carpeta compartida en NFS
 - SSH, pero no tiene acceso público
 - Smart HTTPs, complejo de configurar y no es cómodo para autenticarse.
 - Git, parecido a SSH pero sin sobrecargas de cifrado y autenticación
- Inicializando el servidor
 - Si lo hace el administrador del servidor
 - \$git init --bare
 - Si se hace un repo local
 - \$git clone --bare <repo> <repo.git>
 - Se copia el .git al servidor por scp o sftp.
 - Se añade el origen remoto con remote
 - \$git remote add origin ssh://<usuario>@<ip>:<puerto><rutaAlRepo>

Descarga, comparte, pregunta, aprueba.



- Se copia el .git al servidor por scp o sftp.
- Se añade el origen remoto con remote
- \$git remote add origin ssh://<usuario>@<ip>:<puerto><rutaAlRepo>
- Si se trabaja con SSH
 - Los usuarios deben tener acceso a la máquina
 - O se crea un usuario git que lo compartirán, no afecta quien hace que cosa.
 - Se evita que este usuario tenga acceso a una shell, se le asigna una git-shell.

Lección 3: Servidor Web Básico

• Historia

- Al principio a finales de los 80, CERN trabajaba con esquemas de hipertexto para compartir documentos de investigación.
- Tim Berners-Lee desarrolló un protocolo para transmitir hipertexto por la red, el Hyper Text Transfer Protocol, HTTP y el hipertexto se crea con HTML, Hyper Text Markup Language.
- En un comienzo, se tenía el servidor escuchando peticiones en el puerto por defecto 80, y devolvía el archivo HTML estático.
- Eventualmente, para añadir dinamismo, se hace uso de una lógica y una base de datos. La base de datos es necesaria ya que HTML es stateless.
- Las lógicas se ejecutan en la máquina del cliente que trae el script empotrado o en el servidor como tal, y por lo general el lenguaje por excelencia ha sido PHP, aunque se utilizan también ActiveX, Django basado en Python, Javascript, Perl y Node.
- Por lo tanto, si quiero montar una web moderna, la web 2.0, es necesario primero que todo un servicio que escuche y sirva los documentos HTML: el servidor propiamente, para esto existen alternativas como Apache, Nginx, Lighttpd, etc.
- Para las bases de datos, lo usual es utilizar MySQL u Oracle SQL.
- Para la lógica, PHP, Python, Perl, entre los anteriores mencionados.
- Este conjunto de programas en se denominan la pila de software, que además deben ir sobre un sistema operativo, LAMP es la pila en Linux, WAMP en Windows y XAMP en macOS.

• LAMP

- Es una pila muy común, es utilizada por Wordpress, Joomla, PrestaShop y muchas otras páginas están soportadas por LAMP específicamente.
- Instalación
 - Se podría realizar con taskcel, pero no es recomendable. Es preferible instalar a mano cada componente.
 - Se debe tomar nota del espacio disponible y el tamaño de la descarga, se desea ser minimalistas e instalar solo lo absolutamente necesario, de esta manera también se evitan posibles huecos de seguridad al tener menos software en la máquina.
 - Obtener Apache, PHP y MariaDB en CentOS8
 - \$dnf install httpd
 - \$dnf install php
 - \$dnf install mariadb-server
 - Se arrancan los demonios con \$systemctl enable y start
 - Se instala seguramente MySQL
 - \$mysql-secure-installation
 - Seleccionar una contraseña segura para root, no permitir los usuarios anónimos y no permitir que root acceda desde un sitio remoto.
 - Dar los permisos en el firewall
 - \$firewall-cmd --add-service=http --permanent
 - \$firewall-cmd --reload
 - Se prueba que la pila LAMP está funcionando
 - Conectar PHP con MySQL
 - Crear un fichero en /var/www/html/myscript.php con código que conecta a la base de datos
 - Instalar librerías para la conexión de PHP con MySQL
 - \$dnf install php-mysqlnd.x86_64
 - Crear la base de datos para dicha prueba dentro de MySQL, con \$mysql -u root -p y dentro CREATE DATABASE db;



GET IT ON
Google Play

Download on the
App Store

- \$dnf install php-mysqlnd.x86_64
- Crear la base de datos para dicha prueba dentro de MySQL, con \$mysql -u root -p y dentro CREATE DATABASE db;
- Dar permisos a SELinux para que haya conexión entre bases de datos por httpd con
- \$setsebool httpd_can_network_connect_db 1
- Probar la conexión yendo a 192.168.56.110/myscript.php

Práctica 2: Instalación y configuración de servicios

Instalación de Software

Utilizamos dos: yum(Fedora) y apt(Debian). Con ambos comandos podemos obtener software.

Cuando queremos instalar algo hacemos yum search y buscamos las aplicaciones que queramos.

Para ver que podemos obtener con yum, ejecutamos yum provides. La orden yum lo que hace realmente es dirigir a dnf, esto ocurre si es centOS 8, en centOS 7.5 sigue siendo yum. La sintaxis es igual para yum y dnf.

Para buscar con apt hacemos apt search, actualizamos con apt update. Apt get está en desuso ya que se desarrolló apt como interfaz por encima.

Se empezó a trabajar con los paquetes snap, esta fue una idea de todas las distribuciones para homogeneizarlo todo en un solo sistema de paquetes (usar snap da la ventaja de que tiene las dependencias incorporadas), pero Fedora no usó snap y empezó a usar dnf, el futuro de que se use en todas las distribuciones aún es incierto.

En ubuntu se puede usar tasksel que es una interfaz gráfica de instalación, pero esta al instalar puede dar un error llamado error 100 que se da frecuentemente y además cuando instalamos algún software no sabremos que versión estamos instalando. lo que dependiendo de la aplicación nos puede molestar al revisar su manual. Con las descargas que haremos en esta práctica se pueden hacer con tasksel y apt para comprobar que apt es más útil.

SSH

Introducción

SSH viene de Secure SHell.

Antiguamente, antes de SSH, se conectaba a una máquina usando telnet, lo que ocurre con telnet es que la información que viaja en el protocolo telnet va con el username: usuario password, esto además se pasa en texto plano. Un estudiante de TKK implantó una capa de seguridad sobre telnet para quitar la falla de seguridad debido a problemas de robo de su contraseña en la clase, hoy en día se ha convertido en un estándar.²

Instalación de ssh en servidores

Podemos ver con ps -Af | grep ssh en los servidores si se encuentra instalado ssh, al hacerlo vemos que en ubuntu server no, pero centOS 8 se está ejecutando como demonio sin haber preguntado si queremos instalarlo.

En ubuntu-server al realizar la instalación nos pregunta si queremos instalarlo, como en la P1 le dijimos que no, lo tendremos que instalar. Se puede usar tasksel pero no sabremos que versión es y esto no es recomendable, así que hacemos sudo apt install openssh-server.

Diferencia entre SSH y SSHD

El cliente es ssh y es el que realiza la conexión, el servidor lo que ejecuta es sshd. Son dos cosas muy distintas si decimos sshd o servicio ssh no es lo mismo que ssh. Así que cuando queramos configurar la conexión de los usuarios al servidor cambiaremos /etc/ssh/sshd_config y cuando queramos cambiar la configuración de la conexión del servidor a los usuarios cambiaremos /etc/ssh/ssh_config. Si estamos en el servidor hablamos de sshd si estamos en el cliente hablamos de ssh. Para encontrar la versión correcta hicimos apt search ssh | grep server. Si usamos tasksel al acabar de instalarlo ejecuta ssh sin preguntar, lo que es otro inconveniente

Funcionalidades de ssh

Al usar ssh x, x se va a marcar con su fingerprint permanentemente para marcar esa ip y que nadie pueda falsificarla. Como ejemplo podemos hacer ssh localhost para comprobar que funciona. Con ctrl+d nos salimos

Conexión usando nuestra terminal como cliente

1. [Centos/Ubuntu] Hacemos ip addr y usamos la ip de enp0s8, en mi caso, 192.168.56.110/24
2. [Cliente] Nos conectamos al servidor: ssh 192.168.56.110 -l usuario_del_servidor

Conexión al root

Con la opción de -l elegimos usuario, CentOS tiene el permitRootLogin en true, por lo que se puede acceder al root desde ssh, mientras que ubuntu server no lo permite, ya que en ubuntu en el archivo de configuración de ssh se encuentra la opción PermitRootLogin prohibit-password. Para aumentar la seguridad de centOS lo configuraremos haciendo lo siguiente:

1. vi /etc/ssh/sshd_config para abrir el archivo
2. Buscamos con /permit
3. Cambiamos con change word el yes a no
4. Reiniciamos el servicio con systemctl restart sshd

Si ejecutamos el comando con el que entrabamos (ssh ip_server -l root) y le añadimos la opción -v podremos ver que la configuración se ha realizado con éxito

Puerto de escucha

Con sshd escuchamos por defecto en el puerto 22, vamos a cambiar este puerto para mayor seguridad al puerto, por ejemplo, 22322 (seleccionado arbitrariamente). Este cambio se puede hacer con vi, pero lo vamos a hacer con sed (String EDitor), que modifica cadena de caracteres.

Modificación del puerto

1. Modificamos el archivo con sed s/'Port 22'/'Port 22322'/ -i /etc/ssh/sshd_config (el comando es sed s/'Contenido_que_se_va_a_cambiar'/'Contenido_por_el_que_se_va_a_cambiar'/ -i archivo). En ubuntu usaremos el puerto 22022

Descarga, comparte, pregunta, aprueba.



2. Aseguramos que se haya realizado bien entrando al archivo con vi /etc/ssh/sshd_config para comprobar el cambio y quitamos la almohadilla (si está), ya que hace que esté comentado y no afecte
3. Reiniciamos el servicio con systemctl restart sshd y nos dará un error

Possibles errores al cambiar el puerto

En centOS con los comandos que nos indica (systemctl status sshd.service y journalctl -xe) vemos que ha habido un error al iniciar ssh al intentar vincularse sshd con el puerto 22322 por un fallo de permisos. Para poder cambiar el puerto tenemos que usar el comando que se encuentra en el fichero que cambiamos justo encima del puerto:

```
semanage port -a -t ssh_port_t(tipo de puerto) -p tcp #NúmeroDelPuerto
```

En ubuntu cambiando el mismo archivo ya funcionaría.

Cortafuegos

Cuando realizamos el cambio del puerto en ubuntu nos funciona pero en centos, haciendo semanage, nos sigue sin dejar hacer conexiones al exterior. Probamos a hacer una conexión desde el propio servidor y vemos que si nos deja, en conclusión, el cortafuegos de centos no nos está dejando conectarnos con el exterior.

Esto no implica que ubuntu no tenga el cortafuegos, es que está desactivado por defecto. Si realizamos en ubuntu man ufw (Uncomplicated FireWall) vemos la opción status y al hacer ufw status vemos que está desactivado.

Al poner en ubuntu el comando:

```
ufw allow 22022(puerto)
```

Estamos permitiendo el tráfico a través de ese puerto, para realizar esta acción en CentOS usaremos el comando firewall-cmd, investigando en el manual vemos que podemos añadir el puerto con el comando:

```
firewall-cmd --add-port=22022/tcp
```



Pero si examinamos mejor el manual vemos que con esa línea al reiniciarse el servidor o el cortafuegos el puerto estará cerrado, esto se solucionaría con:

```
firewall-cmd --permanent --add-port=22022/tcp  
firewall-cmd --add-port=22022/tcp
```

Hacen falta introducir los dos comandos o en vez del segundo comando ejecutar firewall-cmd --reload

Configuraciones

Para ver el tipo de puerto que pasar a la orden semanage usamos semanage port -l | grep ssh

Acceso sin contraseña

Para esto usaremos el comando sshkeygen. Este comando cuando se ejecuta en el cliente genera un par de llaves, la pública y la privada. Estas claves las vamos a usar para cifrar la comunicación. La llave pública la vamos a distribuir y los servidores van a cifrar la información con la llave pública, y este cifrado solo se puede descifrar con llave privada, de modo que solo quien tenga esta llave podrá descifrarlo.

Este cifrado es asimétrico, ya que usamos una llave para el cifrado y otra para el descifrado.

Con crear una llave publica y privada podemos usar esta para todos los servidores. Si perdemos la llave tendremos que generar una nueva y borrar la llave de los usuarios

Realizar acceso sin contraseña en ubuntu/centos

1. [Cliente] Ejecutamos: ssh-keygen
2. [Cliente] Seleccionamos el directorio por defecto y no introducimos contraseña
3. [Servidor] Modificamos el archivo sshd_config y ponemos
 PasswordAuthentication yes
4. [Servidor] Reiniciamos el servicio: systemctl restart ssh

5. [Cliente] Copiamos la clave como queramos (con pendrive,etc.) pero vamos a usar el comando ssh-copy-id: ssh-copy-id ip_servidor -p puerto
6. [Servidor] Modificamos el archivo sshd_config y ponemos
 PasswordAuthentication no

Este proceso se puede automatizar con un script, ya que cada vez que se quiera modificar la contraseña, debemos cambiar el sshd_config y luego dejarlo a como estaba, pero esto nos surge una duda, ¿que pasa si en un proceso de ssh que estamos ejecutando un script perdemos la conexión? Se pierde también, así que para ello usaremos screen o tmux.

Screen, terminator y tmux

Al ejecutar screen lo que ocurre es que un proceso bash nuevo se abrirá, y todos los procesos nuevos que abramos se abrirán en el proceso screen que lo está ejecutando.

Podemos dejar varios screen abiertos y enlazarnos (attached) a uno u otro. Para poder hacer esto tenemos que estar en estado detached, para ello ejecutaremos screen -r. Para ver la lista de screen hacemos screen -list y para enlazarnos a uno lo haremos con: screen -r pid.tty.localhost

El pid y el tty podemos saber cual es haciendo ps xf. Básicamente es como hacer alt+tab.

Seguridad: Fail2Ban y RKHunter

Instalamos fail2ban (dnf install fail2ban) para proteger nuestro servidor, al instalarlo está desactivado así que lo activaremos (systemctl enable fail2ban). Con esto marcamos que en el siguiente reinicio arrancará, podemos reiniciar el servidor o activarlo (systemctl start fail2ban).+

Al realizar fail2ban-client status vemos que tenemos cero cárceles configuradas, así que abriremos el archivo de configuración (todos los archivos de configuración se encuentran etc), el archivo está en /etc/fail2ban/jail.conf. Al abrirlo vemos un warning que nos indica que ese archivo no se debe modificar y nos dice que ese archivo se debe copiar al .local, ya que las modificaciones ahí se borrarán al recargar servicio. Copiamos con cp -a jail.conf jail.local y ponemos el enabled en la opción de cárceles ssh. Recargamos el servicio.

Para monitorizar las cárceles hacemos fail2ban-client status para ver que cárceles tenemos, y si ponemos fail2ban-client status sshd podemos ver nuestra cárcel del servicio sshd. Al intentar conectarnos desde el cliente y fallar varias veces en el logeo podremos ver nuestra ip en el status.

En la configuración esta puesto el puerto por defecto, si hemos realizado los pasos para cambiar el puerto tendremos que cambiar la opción port = id Puerto. Ahora cuando volvamos a intentar conectarnos no podremos.

Rootkit es un analizador de software malicioso encargado de proteger que no haya backdoor, etc. Es un programa que realiza el análisis y no se encuentra en ejecución.

Copias de Seguridad

Una copia no es lo mismo que un back-up, el back-up nos da más información, como de cuando es la "copia", si decimos que tenemos diferentes versiones ya tenemos back-up. Así decimos un back-up es una copia con meta-datos sobre esta.

Comandos clásicos, la mayoría en desuso:

- dd: útil para copiar bit a bit, se usa dd if=ruta_de_entrada of=ruta_de_salida. El problema es que los meta-datos no se copian, pero como ventaja tenemos una copia exacta. Actualmente eso se usaría para realizar una recuperación de disco.
- cpio: coge un archivo de una entrada estándar y genera un único archivo. Es como un empaquetador pero sin comprimir. Si enviamos los back-up de este modo la entrega podría durar demasiado, así que para ello usamos una compresión,
- tar: es como cpio pero no es necesario que venga en un flujo directo, podemos pasarle la ruta con el directorio para que empaque. Más tarde, tar además de comprimir pudo empaquetar y comprimir.
- rsync: sincroniza una fuente con un origen de datos, esto se puede realizar con ssh. Se debe tener cuidado porque puede borrar archivos, se debe tener cuidado con la opción --delete. Para hacer un uso común de rsync es con la

Descarga, comparte, pregunta, aprueba.



opción -aviz (son las siglas de all verbose informe y comprimido). Este realiza un envío incremental. Las opciones más comunes que se usan son:

- -r: recursivo
- -l: enlaces
- -t: conservar fecha
- -p: conservar permisos
- -o: conservar owner (propietario)
- -g: conservar grupo
- -d: para archivoz especiales
- -v: verbose, para que muestre información al sincronizar
- -z: comprime
- -i: muestra un resumen final

Ubuntu nos recomienda usar tar como sistema para crear los back-up.

Al enviar los back-up a nuestro servidor, por ejemplo con sftp, el envío (tanto al enviarlo al servidor como si tenemos que recuperarlo) puede llegar a durar más tiempo de lo que nos llevaría restaurar la versión, así que para ello utilizamos métodos como los siguientes para aligerar el proceso:

- Compresión del back-up
- Uso de caché/buffer.
- Back-up incremental. Tenemos una imagen base y lo que almacenamos son las diferencias entre las distintas versiones.

Para establecer el lugar físico del servidor debemos seguir la ley vigente, primero la LOPD, luego la RGPD.

Los profesionales usan software concreto, que causa un aumento de carga en el sistema pero asegura un back-up automático y sencillo, ejemplos de este software son: Tartarus, Amanda, sftpclone

Control de cambios



Práctica 2: Instalación y configuración de servicios

WUOLAH[®]

Un modo de hacerlo de forma básica es hacer una copia del fichero de configuración, pero además existen herramientas que lo controlan directamente (/etc) o usar un sistema por debajo como git.

Git

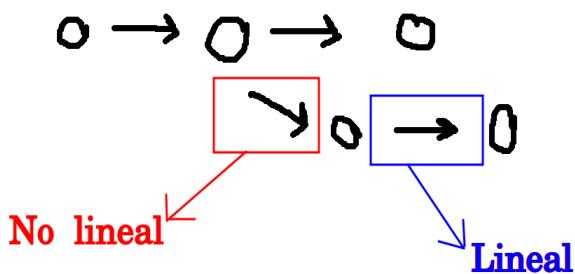
El uso de git nos da ventajas que podemos seguir los cambios hechos y quien es el autor de ellos, además que cada usuario podría tener su propio entorno de pruebas

Tenemos un directorio (normal, como los de siempre), en este vamos a trabajar y vamos a poder marcar como seguimiento, al hacer esto se guardan de forma abstracta en el stage (que será una zona en la que se registran los cambios), y cuando confirmamos estos cambios se guardan en el repositorio o zona commit. Está pensado sobretodo para texto, ya que con imágenes, audio, etc. es mucho más complicado.

Podemos tener archivos que no estén en el commit o repositorio pero si en el directorio de trabajo.

Cada vez que se pasa del área de seguimiento al repositorio se hace un hash de 40 caracteres, aunque solo se suelen usar los 7 primeros. El último commit realizado tiene un puntero hacia el llamado HEAD, con este puntero podemos avanzar por los commit hechos. Al desplazarnos se guarda un log con nuestro desplazamiento que se conseguiría con git reflog

Entre los commit pueden aparecer .. si hay intervalos lineales de commits y ... si es no lineal, esto se refiere a si se ha hecho alguna ramificación (no lineal) o si respecto al anterior es lineal.



En git tenemos unos objetos llamados blobs (binaries large objects) que guardan los datos, además de que cada uno tiene su tabla hash, los trees que son los directorios y los commits que hagamos. Entonces se podría decir que Git es como un sistema de archivos direccionable por contenido, entonces lo que tenemos es realmente un par llave-valor.

Si un repositorio es bare, no hay working directory.

Al añadir el nombre a la versión se debe establecer bien, en las diapositivas hay un enlace a una web con directivas para establecerlo.

Los commit deben ser lo más atómicos posibles, para encontrar fácilmente el bug

Configuración de git en un servidor

Para ello vamos a usar el servidor de CentOS y vamos a usar el modelo de servidor central en el cual hay un repositorio y hay dos equipos de trabajo, cada uno tendrá su zona de trabajo y su repositorio propio y podrán acceder a un repositorio alojado en centos.

Recordad que todos los comandos se deben hacer en modo root o con sudo. Con sudo ha habido problemas en clase, realizando estas prácticas con login root y contraseña prácticas, ISE funciona.

1. [Centos8] Instalamos git con sudo dnf install git

2. [Centos8] Creamos el directorio para el repositorio: mkdir /git y entramos en él: cd git
3. [Centos8] Editamos el fichero etc/group: vi /etc/group y en git añadimos: git: usuario1:usuario2
4. [Centos8] Añadimos el grupo git al directorio git: chgrp -R git /git
5. [Centos8] Añadimos los usuarios git al directorio g: chown -R git /git
6. [Centos8] Cambiamos los permisos del directorio: chmod 277 /git
7. [Centos8] Inicializamos git: git init repo1 --bare
8. [Centos8] Añadimos un usuario: useradd nombre_usuario, en mi caso, sudo useradd prueba1
9. [Centos8] Añadimos una contraseña al usuario: passwd nombre_usuario e introducimos practicas,ISE
10. [Local] Abrimos los clientes, si no se tienen dos usuarios en el ordenador, se le añade un usurio. Al hacer su usuario entramos con una terminal a ese cliente y abrimos otra terminal con nuestro usuario
11. [Local] Creamos un directorio para tener nuestros repositorios
12. [Centos8] Cambiamos el fichero ssh: vi /etc/ssh/sshd_config y en el apartado del puerto ponemos Port 22 sin almohadilla
13. [Centos8] Iniciamos el servicio de ssh con system start sshd
14. [Local] Pillamos el repositorio: git clone ip:/git/repo1. La ip será la que el ifconfig de su máquina y del servidor coincidan.

Comandos necesarios

- git status: Muestra los commit y los archivos que no se están siguiendo
- git commit: actualizamos los repositorios
- git add archivo: añadimos al repositorio, se pueden poner varios archivos
- git commit -m "string": le añade el string del parámetro como nombre de versión
- git commit --amend: modifica el mensaje que tuviese hecho el commit

Descarga, comparte, pregunta, aprueba.



- git commit --amend -a: añade un archivo nuevo al commit
- git push: envía al repositorio
- git pull: nos crea una rama que obtiene los archivos del repositorio
- git diff: ver las diferencias entre el working directory y el area de seguimiento.
- git diff HEAD: diferencias entre working directory y el repositorio
- git diff --staged: diferencia entre el stage y el repositorio

Modificaciones y ver su contenido

Al modificar un archivo y realizar un git status nos pondrá que hay un archivo modificado en rojo, y si lo añadimos con git add nos pondrá que esta modificado pero en verde, esto se debe a que cuando estaba en rojo estaba modificado en el working directory y que cuando estaba en verde esta modificado en el working directory y estaba marcado para seguimiento (estaba en el stage) pero no se había enviado al repositorio haciendo commit.

Para ver mejor las modificaciones usaremos git diff, así cuando hagamos una modificación y lo añadamos git diff devolverá 0, si lo añadimos y luego lo modificamos git diff nos lo mostrará. Si queremos compararlo con el repositorio en vez de con el stage usaremos git diff HEAD como indicamos en el apartado anterior.

Para hacer las modificaciones permanentes se podría hacer de dos modos:

- Modificamos, git add, git commit
- Modificamos, git commit -a

Recuperar un contenido

Para recuperar un archivo deberíamos saber cual es la última versión que tenemos de este, para ello usaremos git checkout HEAD nombre_archivo y una vez así sabremos .

HEAD es el último commit hecho, así que al restaurar este HEAD se nos modificará, así que si queremos recuperar un commit usaremos: git reset Z_primeros_caracteres_del_commit_SHA. Al hacer esto perderemos todo el



trabajo desde ese commit, si queremos mantenerlo tendremos que usar otro comando.

Con git revert commit_que_se_quiere_trae se creará un nuevo commit con los datos del commit antiguo y ese será nuestro commit en el HEAD. Si no queremos que haga un commit nuevo usamos -n.

Ramas nuevas

Podemos bifurcar el grafo creando nuevas ramas, por esto necesitaremos saber en que rama estamos y para ello usaremos git branch. La rama por defecto es la rama master. Este comando también nos sirve para crear ramas nuevas con git branch nombre_nueva_rama pero no nos cambiaremos esa rama. Para cambiarnos de rama usaremos git checkout nombre_rama, con la opción -b crea una rama y te mueves a ella.

Si queremos guardar el estado de la rama por qué vamos a movernos de ella, pero no queremos hacer un commit usaremos la función stash, el cual permite salvar un directorio sin tener que hacer un commit. Con git stash list veremos todos los stash que tenemos y con git stash apply id_stash volveremos a la copia que habíamos hecho.

Una vez tengamos varias ramas, si queremos unir el trabajo de ambas haremos git merge.

Usando un servidor

En el servidor tendremos un repositorio bare, y cada cliente tendrá su estructura particular (directorio de trabajo, stage, repositorio). Para crear este repositorio bare tendrá que hacerlo el administrador del sistema, o alguno de los clientes que lo vayan a usar, el que tenga que crearlo usará git init --bare. Una vez este creado, cada cliente hará git clone al repositorio.

Si partimos de que se tiene un repositorio local que se quiere convertir en repositorio servidor se deberá hacer los siguientes pasos:

1. git clone --bare mi_repo mi_repo.git
2. Se copia al servidor (sftp o scp): scp -P 22022 -r gitrevert.git
192.168.56.105:/home/usuario

3. Añadimos el origen remoto: git remote add origin
ssh://192.168.56.105:22022/home/usuario/gitrevert.git

Con esto el resto de colaboradores podrán hacer git clone ubicación_repo, al provenir de un usuario un ejemplo sería: git clone
ssh://192.168.56.105:22022/home/usuario/gitrevert.git

Aspectos a tener en cuenta en trabajo grupal con git

Si para acceder al servidor estamos usando ssh se debe tener en cuenta que lo visto no funcionará si los usuarios no tienen acceso a la máquina o si el servidor no está bien configurado. Otro modo para no darle acceso a todos los usuarios es que para todos los usuarios que vayan a usar la máquina usarán el mismo usuario que se habrá creado como el usuario git, aunque se use un usuario se sabrá quien ha hecho cada commit. Si usamos esto en vez de la primera opción deberemos hacer que el usuario git no tenga acceso a la shell y que tenga acceso a una git-shell.

Comandos para trabajar en grupo:

- git clone: clona repositorio
- git add remote: añade un origen remoto
- git push: enviamos al remoto los commits hechos en el local
- git fetch: traemos los cambios que hay entre el repositorio local y remoto
- git pull: hace un git merge con el repositorio local y con el git fetch

Servidor Web

Historia

A finales de los 80, Tim Berner Lee definió el protocolo HTTP (HyperText Transfer Protocol). Esto se basaba en el intercambio de dos ficheros de hyper-texto, estos ficheros eran HTML (HyperText Markup Language).

HTML es un lenguaje el cual tenemos el texto y marcas que nos vienen entre <> que nos indican funcionalidades.

Nosotros vamos a tener un servidor con un puerto de escucha en HTML, en el puerto 80. Los ficheros HTML son estáticos. Entonces si un usuario quiere poner un comentario, nuestro esquema no serviría, por lo que tendríamos que cambiarlo y añadir una base de datos a nuestra estructura.

A nuestro código se le puede poner lógica con script/javascript, pero si el control de usuarios lo hacemos con esta lógica al darle a inspeccionar elemento ya podríamos leer el usuario y contraseña, así que esta funcionalidad la tendremos que usar en la BD. Para gestionar usaremos MySQL, propiedad de oracle, o su versión libre, MariaDB.

El lenguaje web ha sido durante muchos años php, aunque también se puede usar python, c (como swad), etc. En nuestra práctica vamos a necesitar una pila de stack llamada LAMP (Linux Apache MySQL PHP/Python), este concepto también se conoce como solution software, puede variar según los software que usemos.

Vamos a usar LAMP debido a que Wordpress, Moodle, PrestaShop usan esta pila.

Instalando LAMP

Podemos instalarlo con tasksel pero con ello instalaremos dependencias que no queremos, ni sabremos sus versiones, lo ideal sería ir descargando lo necesario para la pila lamp poco a poco (mysql,php,apache).

Si no queremos usar tasksel o lo hacemos con centos ejecutamos los siguientes comandos (en ubuntu cambiaremos dnf por apt):

```
dnf install httpd  
dnf install php  
dnf install mariadb-server
```

Al instalar httpd y miramos su status vemos que está inhabilitado así que lo activamos e iniciamos:

```
systemctl enable httpd  
systemctl start httpd
```

Al instalar mariadb y miramos su status vemos que está inhabilitado así que lo activamos e iniciamos:

Descarga, comparte, pregunta, aprueba.



```
systemctl enable mariadb.service  
systemctl start mariadb.service
```

El manual nos dice que debemos ejecutar el script my_sql_secure_installation. Este script mejora la seguridad de la base de datos, para empezar le pone una contraseña al root ya que por defecto esta como contraseña vacía. Además tiene usuarios anónimos, que no nos interesa. Tampoco queremos que root se conecte de forma remota. Marcamos que queremos borrar la BD de prueba y recargamos la tabla de privilegios.

Para comprobar que funciona haremos curl (para ver una url): curl localhost.

Para comprobar que php funciona ejecutamos php -a y ahí escribimos echo ("hola");

Para comprobar que mysql funcione ejecutamos mysql -u root -p, introducimos la contraseña practicas,ISE y estaremos dentro de la consola de mariadb

Configuración del servidor

Al acceder desde el cliente a la web del servidor (poniendo en el navegador la ip del servido) vemos que no carga nuestra página, esto es debido al cortafuegos. En ubuntu se arregla con ufw allow 80.

Comprobar que funciona lamp

Utilizamos un ejemplo de la página de documentación de php:

<https://www.php.net/manual/es/function.mysql-connect.php>

1. Creamos un archivo de prueba: touch /var/www/html/miscript.php
2. Lo editamos con vi /var/www/html/miscript.php
3. Copiamos el ejemplo y en el modificamos la segunda linea a la siguiente:
\$enlace = mysqli_connect("127.0.0.1", "root", "practicas,ISE", "db");
4. Le damos permisos con el firewall: firewall-cmd --add-service=https --permanent
5. Recargamos el firewall: firewall-cmd --reload
6. Creamos la base de datos para ello ejecutamos mysql -u root -p



7. Creamos la base de datos con CREATE DATABASE db;

8. En nuestro navegador ponemos la ip/miscript.php

Al hacer esto nos ha faltado un pequeño error de configuración y en nuestra web ha quedado expuesta nuestra contraseña disponible para cualquier pirata informático. Para ello vamos a ir al archivo de configuración. Para solucionar esto vamos a hacer lo siguiente:

1. vi /etc/httpd/conf/httpd.conf

2. Buscamos donde pone DirectoryIndex index.html y añadimos la opción de ficheros php y nos quedaría así DirectoryIndex index.html *.php

3. Reiniciamos servicio http

Al hacer esto nuestra contraseña no es pública pero nos sale el error 500, así que miramos el error de log con less /var/log/httpd/error_log y less /var/log/httpd/acces_log y vemos que el error ha sido en el acceso y del tipo 500. Vamos a mirar /var/log/php-fpm que es el modulo que ejecuta los archivos php, allí tenemos el archivo www-error.log que devuelve el error de php, esto se podría configurar para que se ele enviase al cliente.

Después de ver este error vemos que nos ha faltado instalar la biblioteca que conecta php con mysql, la cual se instala con: dnf install php-mysqlnd.x86_64.

Aún así no se puede ejecutar por culpa de SLinux (Secure Linux), al hacer getsebool -a | grep http vemos los valores booleanos que hacen referencia a http, entre ellos nos fijamos en httpd_can_network_connect_db, y lo pondremos en 1 haciendo el siguiente comando: setsebool httpd_can_network_connect_db=1 y con esto esta completado.