Práctica 2 – Lección 1 - Resuelto

Apuntes clase:
(Tanto CentOS como Ubuntu).

1. Instalarlo (sshd)
2. Deshabilitar el acceso de root -> Por seguridad. Si quisieramos, entrar como usuario normal y sudo.
3. Cambiar puerto de ssh. (Por defecto 22 a otro). Por practicar. OJO Firewall
4. Acceso por ssh sin contraseña. (Conectarme entre máquinas con y sin contraseña. OJO MV no pueden entrar al anfitrión pero si al revés).

Llave pública que la conozca todo el mundo pero la privada solo nosotros. Puedo cifrar con pública y descifrar con privada.

putty para windows o WSL mejor

ssh copy-id [al usuario y eso] -> dar llave publica del usuario. Pedirá contraseña y no volverá a requerir pedirla nunca más. Crea archivo con llaves públicas de personas autorizadas.

Al reiniciar MV pierde las llaves. Borrar entrada para eso

VÍDEO - IMPORTANTE

SSH: Secure Shell. Con este servicio podemos autenticarnos, seguridad (datos van encriptados) y control de integridad de estos datos.

Surge como alternativa segura a telnet ya que telnet no encriptaba datos y cualquiera con acceso a la red, estaría viendo el tráfico.

Con ssh, tendremos un cliente que se conecta con un servidor (Arq cliente-servidor).

Imaginemos que un cliente quiere conectarse a un servidor. Para esto, cliente invoca al cliente ssh (un cliente) para conectarse a un servidor. Tras una negociacion y seguir el protocolo consigue conectarse al servidor. El servidor tiene sshd (un servicio) , un demon que está constantemente escuchando.

OJO con ssh nos podemos referir al cliente y al servicio, durante el vídeo, puede haber alguna ambigüedad entre ambos de no especificar nada.

Servidor podría ser cliente ssh de otro servidor (por ejemplo, un clúster de ordenadores).

Al configurar ssh, tendremos dos archivos que es bien importante diferenciar:

- /etc/ssh/ssh**d**_config -> Servicio
- /etc/ssh/ssh_config -> Cliente

@Zukii on Wuolah

INSTALACIÓN EN UBUNTU SERVER

Disciplina server hardening

Empezamos con Ubuntu Server

Comprobar que tenemos IP 192.168.56.105/24 (105 porque .15 diría que es un error) con
> ip addr

Usaremos apt para descargar ssh. Filtraremos por servidores

> apt search ssh | grep server

> sudo apt install openssh-server

Nos da error y para solucionarlo actualizamos con sudo apt update (a mi no me dió xd)

Volvemos a instalar openssh-server

Comprobar que el servicio se ha activado (el proceso) con:
> ps -Af | grep sshd
Si no inicia el servicio usamos > sudo /etc/init.d/ssh start

Para probarlo podemos conectarnos al servidor con nuestro cliente con
> ssh localhost
> yes (ahora localhost pasa a ser un host conocido)
> ISE

CTRL+D para salir

Podemos ver el directorio /home/ssh y vemos que tenemos un documento known_hosts en el que aparece el fingerprint de localhost (y los fingerprints de los hosts que conozcamos)

Es importante editar la configuración de ssh para impedir el acceso al root (ya que este podría acceder a todos los usuarios). Si quisiéramos acceder al root, tendríamos que usar un usuario como pasarela que cambiará de usuario y pasará a tener privilegios.

> sudo nano /etc/ssh/sshd_config

Aquí, buscamos #PermitRootLogin y ponemos no (por defecto prohibit-password, podemos acceder a root con cualquier mecanismo distinto a la contraseña)

> systemctl restart sshd (reiniciar servicio sshd)

Probamos a intentar acceder como root desde otra MV (por ejemplo WSL de Ubuntu) con
> ssh root@192.168.56.105 (aparece error de Permission denied)
> ssh -l root 192.168.56.105 (equivalente a la de arriba)

Si ponemos además de las órdenes anteriores, -v al final de la orden, podremos depurar problemas de conexión y ver que errores nos devuelve el servidor en caso de que haya problemas.

Ahora vamos a CentOS (tiene algunas diferencias)

ps -Af | grep sshd
La instalación por defecto de CentOS configura e inicia el servicio sshd como podemos ver

Diferencias	Ubuntu Server	CentOS
Instalación	Pregunta por la instalación	Por defecto
Comprobar estado del servicio (systemctl status sshd) NOTA: CTRL+D para salir	ssh (por eso usar solo segunda forma) y sshd	solo sshd
PermitRoot	no con contraseña	Sí
Firewall	ufw (uncomplicated firewall)	firewall-cmd
Firewall por defecto	No	Sí

Ahora probaremos a conectarnos al servicio desde otra terminal (WSL por ejemplo)
> ssh alvaro@192.168.56.110
> logout (para salir)

¿Podremos acceder al root del servicio de CentOS?
> ssh root@192.168.56.110 (y nos deja --> Diferencia)

>sudo nano /etc/ssh/ssh**d**_config

Buscamos el #PermitRootLogin y ponemos no.

> systemctl restart sshd (reiniciar servicio sshd)

Volvemos a probar:

> ssh root@192.168.56.110

Vemos como ya no nos deja. Ya tendremos configurado sshd en Ubutnu y CentOS e inhabilitado el acceso como root. Esto es lo mínimo que hay que hacer cuando estamos utilizando el servicio sshd

Ahora cambiaremos el puerto (por defecto ssh es el 22). Esto lo hariamos para evitar que sea trivial intentar acceder al servicio. Para ellos tenemos la directiva port y el archivo de configuración.

@Zukii on Wuolah

En este caso, en vez de usar nano o vi, usamos stream editor, nos permitirá buscar una cadena y sustituirla.

(Esto lo hacemos desde la WSL, siendo clientes del servicio de CentOS)

> ssh alvaro@192.168.56.110
> sudo su
> sed s/'Port22'/'Port 22022'/ -i /etc/ssh/sshd_config
(sustituir la cadena Port22 por Port22022 en el archivo sshd_config)
> systemctl restart sshd.

Ahora desde otra terminal (otra de WSL por ejemplo) y tratamos de acceder con
> ssh alvaro@192.168.56.110
Nos dejaría entrar igualmente por el puerto 22. (Nota, para elegir puerto en el comando ssh es con la opción -p [núm], en nuestro caso -p 22022)

El problema es que la cadena estaba comentada. Repetimos, pero quitando el #
(también podemos hacerlo con nano)
> sed s/'#Port22022'/'Port 22022'/ -i /etc/ssh/sshd_config
(sustituir la cadena Port22 po r Port22022 en el archivo sshd_config)
> systemctl restart sshd.

Nos da un error de Job for sshd…
Para monitorizar estos problemas tenemos el comando journalctl.
Comprobamos el estado de sshd:

> systemctl status sshd -> Vemos que hay error al iniciar el servicio. Al no tener más info invocamos a journalctl -xe

>journalctl -xe
Vemos que hay error al intentar asignar el puerto 22022 por permiso denegado. Editamos el archivo de configuración con nano y vemos que en la cabecera hay una línea que nos dice que si queremos cambiar el puerto en un sistema que ejecuta SELinux, hay que informar a SELinux sobre este cambio.

Lo haremos mediante el comando semanage (que permite gestionar políticas de SE Linux)

> semanage port -a (añadir puerto) -t ssh_port_t (tipo de puerto) -p tcp (protocolo) 22022

Instalaremos el paquete, buscaremos quien proporciona el comando semanage:
>dnf provides semanage

OJO, si nos da error de mirrorlist o algo así lo solucionamos con el comando dhclient

Vemos que el paquete policycoreutils-python… contiene el binario semanage

Podemos usar yum o dnf para instalar los paquetes (CentOS)
>dnf install policy…

@Zukii on Wuolah

> semanage port -l | grep ssh (Vemos los puertos y buscamos el de ssh)
> semanage port -a -t ssh_port_t -p tcp 22022
> semanage port -l | grep ssh (Vemos que se ha añadido el puerto de ssh al 22022)

Volvemos a intentar acceder (en WSL diferente a la que estamos como clientes: > ssh alvaro@192.168.56.110 -p 22022 -v) pero vemos que tenemos un error.

Salimos de WSL en la que estabamos como clientes y probamos a conectarnos desde el propio CentOS con:

>ssh localhost -p 22022

Y vemos que si nos deja. El problema es el firewall (Diferente entre Ubuntu y CentOS). Configuraremos el firewall para que nos permita acceder al puerto 22022.

(En CentOS)

> sudo firewall-cmd --add-port 22022/tcp --permanent (añadimos puerto 22022 con protocolo tcp. Al añadir de forma permanente al recargar el firewall o reiniciar máquina, el puerto se abrirá)

> sudo firewall-cmd --reload (recargar firewall para que se abra ahora. Podíamos poner el comando de antes sin el --permanent para que se abriera en el momento)

Probamos de nuevo a conectarnos desde la WSL y vemos como sí podemos conectarnos
> ssh alvaro@192.168.56.110 -p 22022

VOLVEMOS A UBUNTU

Usamos ufw.

Modificamos archivo de configuración.

> sudo nano /etc/ssh/sshd_config
Descomentamos la Línea de Port y ponemos Port 22022

>systemctl restart sshd

Comprobar que tenemos acceso desde WSL indicando el puerto

> ssh alvaro@192.168.56.105 -p 22022

Pero esta vez nos deja. UFW está por defecto desactivado (Diferencia)
Activaremos el Firewall y añadiremos el puerto.

> sudo ufw status (está inactivo)
> sudo ufw enable

Volvemos a intentar conectarnos desde la WSL (otra pestaña nueva)
Y vemos que no nos deja.

@Zukii on Wuolah

Tendremos que permitir el tráfico en el puerto 22022
> sudo ufw allow 22022

Volvemos a intentar conectarnos desde la WSL y vemos como ya si nos deja.

FIN

Práctica 2 – Lección 2 - Resuelto

Apuntes de PDF + notas de clase.

dd -> copia de dispositivo (bit a bit, a nivel físico). ej: copiar pendrive

ej: dd if=/dev/sda of=/dev/sdb (copia de sda a sdb)
dd if=/dev/sda of=~/hdadisk.img (imagen en vez de dispositivo)
dd if=hdadisk.img of=/dev/sdb (recuperas de la imagen a sdb)

cpio - tar -> Cogen estructura dentro de directorio y comprimen dentro de un archivo.

ls | cpio -ov > /tmp/object.cpio (-o coge rutas de entrada estándar - ls)
cpio -idv < /tmp/object.cpio (extrae de un archivo con -i)
tar cvzf MyImages-14-09-17.tar.gz /home/MyImages (comprimir)
tar -xvf public_html-14-09-17.tar (extraer)

rsync - rsnapshot -> Sincronizar contenido de dos directorios. Snapshot de forma periódica
Traspasa las diferencias entre origen y destino. También sirve en remoto - ssh.

rsync -a --delete source/ destination/ (Borrar archivos de destino si en algún momento se borran en origen)
rsync /home/Usu1/archiv1[[/][.][*] /home/Usu1/archivSecu
rsync /home/Usu1/archiv1/. username@maquina:/rutadestino
rsync -**e** “ssh” /home/Usu1/archiv1/. username@maquina:/rutadestino (mandar x sh)
rsync -aviz**e** “ssh -p 22” prueba/* alvaro@192.168.56.110:/home/alvaro/patata

Opciones típicas: -r : recursivo ; -l : enlaces “simbólicos” ; -t: conservar fecha ; -p: conservar permisos ; -o: conservar propietario ; -g : conservar grupo ; -D archivos especiales -a: todo

-z comprime, -v mostrar info, -i muestra resumen final,

Para restaurar, invertimos el orden del origen y destino

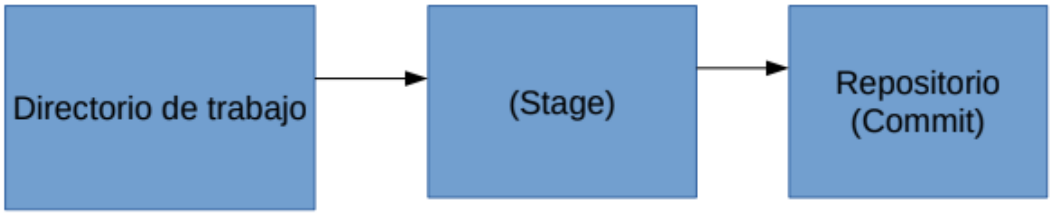
LVM Snapshots: realizar snapshots de volúmenes lógicos. Estos snapshots pueden ser utilizados para revertir el estado inicial y conocido de un LV.

//

Control de Cambios:

podemos, antes de modificar un archivo, lo copìamos con otro nombre
otras herramientas como /etc/kepper que controlan los archivos de configuración

Lo mejor es usar una herramienta de control de versiones como git



Directorio: el directorio donde hacemos lo que tengamos que hacer
Stage: Zona intermedia donde se registran cambios e incluimos los cambios a seguir
Commit: Zona donde se registran ya los cambios en la BD

Tras cada commit, se genera un hash de 32 bits que lo identifica. El último commit es el head.

Todos los objetos de git tienen un hash para identificarlos (commits, trees: directorios y blobs: archivos - aunque solo 1 hash para todas las versiones del archivo).

USO DE GIT

Instalamos git si no está (aunque creo que al menos en centos si está)+
Nos vamos al directorio donde vamos a trabajar

git init .

(Ya podemos hacer **git log (ver versiones) / status / branch** al tomar la instantánea)

Indicamos quienes somos con:
git config --global user.name "nombre"
git config --global user.email correo

Para añadir ficheros para seguir:

git add nombre_archivo (enviar a stage)
Nota: usar .gitignore si queremos que determinado archivo no se suba (gitignore nombre)

Para tener un primer repositorio añadimos todos los archivos (git add .) (hasta .gitignore) y hacemos commit:
git commit -m "Motivo del cambio"
git commit (y se abre editor de texto para poner motivo)

1) git add para llevarlo a stage
2) git commit para llevarlo a repositorio local
3) git push para llevarlo a repositorio origen

Si quisiéramos cambiar el mensaje del commit podemos usar **git commit --amend** o para añadir algún archivo **git commit --amend -a** (nombre archivo - sin pasar por stage **getCARCEL**)

Si modificamos un archivo que antes habíamos subido (una nueva versión), deberíamos volver a hacer git add y git commit (para resubirlo)

git diff -> ver diferencias entre el directorio de trabajo y el stage.
git diff HEAD -> dir de trabajo y repositorio
git diff --staged -> stage y repositorio

si hacemos git add * y luego git diff no vemos nada. Pero si no hacemos el git add probablemente se quejaría y nos diría las diferencias.

Si hacemos git diff --staged (o diff HEAD) se quejaría ya que aún no hemos hecho el cambio el repositorio. Si hacemos git commit y probamos todos los git diff, no debería quejarse.

Recuperar contenido:

git checkout HEAD /ruta/archivo ... -> (vuelve a la versión "head" del archivo)
git reset [7_primeros_caracteres_del_commit_SHA] -> actualizará el head (repositrio)
git restore * (descarta cambios del directorio y se queda con la última versión "guardada")

(Nota: podemos desplazarnos entre commits con 7_primeros_caracteres_del_commit_SHA~[num] y volveremos num commit hacia atras desde el commit especificado)

(Otra nota, tras hacer reset, traemos el repositorio, pero el directorio de trabajo no, asi que hacer checkout . o **git reset --hard [7 primeros hash]** -> se trae el repositorio y directorio)

git show [por defecto HEAD-7_primeros_caracteres_del_commit_SHA]. Muestra cambios de una versión respecto a la anterior

git checkout 7_primeros_caracteres_del_commit_SHA(~num) -- ruta/archivo1 ruta/archivo2… (volver esos archivos a la versión que tenían en el commit)

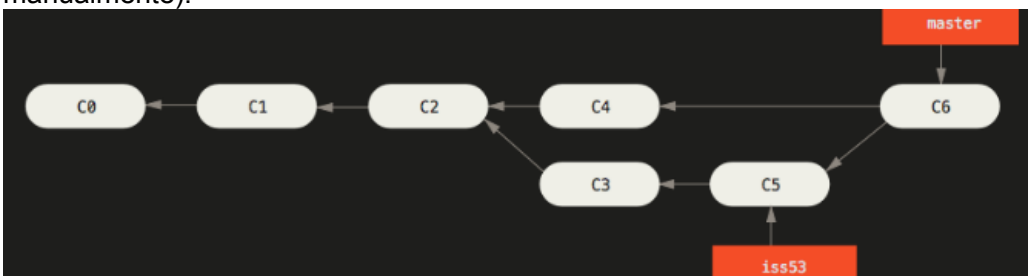
git revert <commit> (7 sha?) -> Quitará los cambios hechos al directorio respecto al commit especificado, creando un nuevo commit (si no quieres que cree un nuevo commit -n)
(git log --graph y podemos ver que hay un nuevo commit, revert del commit que especificamos)

Ramas: para desarollar parelalemente

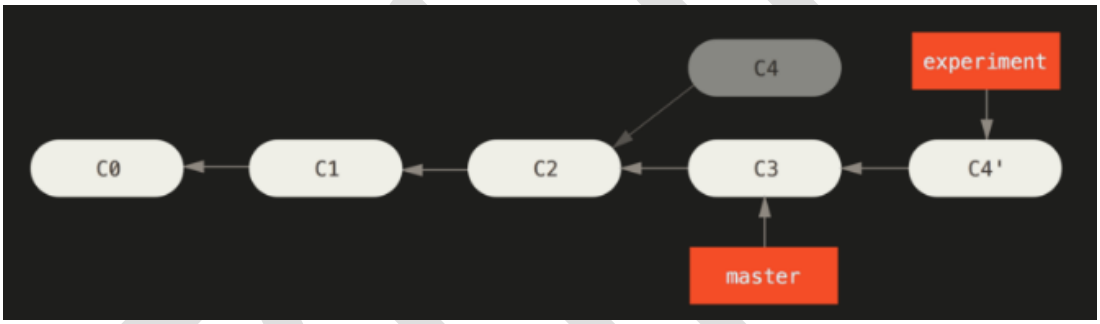
git branch -> muestra rama en la que estamos, master por defecto
git checkout <nombre_rama> -> cambiar de rama
git branch <nombre> -> crear rama
git checkout -b <nombre> -> crea rama y se cambia a ella

git stash -> salva directorio actual, incluso a medias, sin hacer commit (para cambio de rama)
git stash apply -> aplica el último stash creado (lo recupera?)
git stash list -> muestra stash diponibles
git stash apply <stash> -> aplica el stash especificado como <stash> (ver lista antes)

git merge <rama> -> crear commit, con dos padres (rama master y rama especificada). “Combinaría” los archivos que tienen los padres. Para poder hacer esto, hay que volver con checkout a la rama master, bajar los cambios que se hicieran con git pull y git merge. (Si hubiera conflicto con la mezcla, habría que arreglarlo manualmente).



git rebase master -> Tras haber hecho checkout a la rama que quieres “incluir” en el master, coge todos los commit del branch, y los concatena (de más inicial a más final) con el último que estaba en el branch master. (creo que el branch desaparecería y master apuntaría a c4)



Podemos usar varios protocolos de transporte (local, ssh, https, git -> parecido a ssh, sin sobrecarga de cifrado y autenticación)

(Como dice el pdf)

- El administrador puede crear el repositorio origen con **git init --bare** haciendo cada desarrollador un git clone.
- O, partiendo desde un repositorio local con **git clone --bare mi_repo mi_repo.git**, copiamos mi_repo al servidor con **scp -P 22022 -r gitrevert.git 192.168.56.105:/home/alvaro** y añadimos el origen remoto con **git remote add origin ssh://192.168.56.105:22022/home/alvaro/gitrevert.git**

Tras hacer esto, los colaboradores podrán hacer: **git clone ssh://192.168.56.105:22022/home/alberto/gitrevert.git** y tendrán una copia local

Nota: git init --bare -> omite el directorio de trabajo, imposibilita la edición de archivos y la confirmación de cambios en ese repositorio

(Como en clase - administrador crea repositorio y hacemos copia con git clone)
(Tras haber creado repertorio origen en github por ejemplo)
Podemos conectarnos con **git clone ---(url con el @)**
(aunque probablemente debamos registrar una llave pública nuestra para nuestro acceso - ej: **ssh-keygen -t rsa**)

git clone -> clonar repositorio
git add remote -> añadir origen remoto
git push -> enviamos al remoto (repositorio) los commits hechos en el local
git pull -> nos traemos los cambios del repositorio (git fetch) y unimos con los del directorio de trabajo actual (git merge).

git push origin master (master es la rama, origin indica que lo **sube** al repositorio origen)
git pull origin master (**trae** los cambios de la rama master del remoto (origin) y los integra en el head)

Apuntes de clase:

git y ssh: lo más importante para la P2

Git = sistema de control de versiones distribuido. En otras palabras, una herramienta que permite realizar un proyecto, y ser capaz de volver a versiones anteriores. Lo bueno de git, es que permite llegar a TODAS las versiones anteriores. También permite el desarrollo paralelo entre varias ramas.

(Puede que esté incompleto, por eso hago lo del pdf abajo)
dd -> copia de seguridad de dispositivo (bit a bit, a nivel físico). ej: copiar pendrive
cpio - tar -> Cogen estructura dentro de directorio y comprimen dentro de un archivo.
rsync -> Sincronizar contenido de dos directorios locales (como snapshot por ejemplo).
Traspasa las diferencias entre origen y destino. También sirve en remoto - ssh.
(hay un ejemplo en las transparencias con ssh).
snapshot -> ...

git init .

Repositorio de trabajo es la carpeta en la que se hizo git init
(creamos repositorio en la web, por ejemplo, github)
.gitignore, tipos de archivos que no queramos que suba

Al hacer cambio, git hace snapshot tras cada cambio. Una versión, siempre puede volver a la versión anterior.
Línea Principal de desarrollo -> main (aunque puedes cambiarle el nombre)

Puedo conectarme con ssh a la máquina en la que hicimos git init .

NOTA: Nosotros creamos directorio y sobre este hacemos git init . - Debemos hacer el proyecto en ese mismo directorio y de forma interna, git se encarga de la gestión

La diferencia de git respecto a otros sistemas de control de versiones, es que cuando hacemos clone, no bajamos última snapshot (que lo hacemos), si no que también *toda la historia* del proyecto.

Podemos conectarnos con **git clone ---(url con el @)**
(primera vez tendremos que darle a yes)
Repositorios de git, son públicos por defecto, pero por ssh no nos deja acceder si somos un usuario.

Debemos registrar nuestra llave pública (generarla con s**sh-keygen -t rsa**)
github -> perfil -> setting -> ssh keys -> new ssh keys

Ahora si me dejaria el git clone

commit es lo que registra el cambio en la BD
stage es algo intermedio. Preparar cambios para luego usar commit

git status para ver archivos que tenemos y en qué estado. (y más cosas)

git log y ves historia del commit
git push para subirlo al origen (ya que por defecto los cambios estarán en el repositorio local)

1) git add para llevarlo a stage
2) git commit para llevarlo a repositorio local
3) git push para llevarlo a repositorio origen

git pull para actualizar.

SCM = VCS

Ramas de desarrollo para registrar desarrollo paralelo del proyecto. Hacer cambios que no afectan a rama main
Lo mejor es usar las ramas lo menos posible debido a los costes de integración con la rama main.

git branch podeis ver las ramas.
git checkout -b (nombre) -> crea rama y te mete

o **git branch -c (nombre)** y **git checkout** (nombre)

(:wq para salir con vi)

git push -u origin (branch) para cuando hagamos git push, se haga en github (origin) por defecto. Si tuviésemos otro directorio, deberiamos poner git push (nombre repositorio) (nombre branch)

Para mezclar branch con main: 2 formas. (Juraría que debes estar en la rama que vas a mezclar)

1. **git rebase master**(una línea, metiendo branch al final de la main y con el merge al final)
2. **git merge commit** (poner cambio delante del main), para esto hay que volver con checkout a rama principal, bajar los cambios que se hicieran con git pull y git merge (nombre de otra rama) y creamos nuevo commit con dos padres (Si hubiera conflicto de mezcla, habría que arreglarlo manualmente).

...
git add fichero (si hubiera conflicto)
git push

errores:
<<<<<<<<<<Head - rama main
jnasnjsanjas

@Zukii on Wuolah

=====
ujasjjas
>>>>>> rama diferente

juraria que al hacer git commit te abre editor de textos para que expliques cambios o puedes hacer **git commit -m “cadena para identificar la nueva versión”**

Práctica 2 – Lección 3 - Resuelta

Vídeo 1: SSH parte 2

/-----Aspectos Avanzados de SSH-----\

En la P2-L1 configuramos el servicio sshd en el puerto 22022 y deshabilitamos el acceso mediante el acceso root.

Hoy: 1) permitiremos acceso sin contraseña (por rapidez, seguridad de que no nos vean tecleando la contraseña). Esto lo haremos con una llave privada en el cliente, y una llave pública en el servicio.

Para generar una llave: **ssh-keygen**
Nos dirá si queremos guardar la llave en la carpeta por defecto, (**enter** para aceptarlo)
Nos dice que pongamos una contraseña a la llave (no la añadimos en este caso, aunque si tuviésemos la llave en un USB, no sería buena idea) (**enter**).

ls -la .ssh/ y vemos como hay dos llaves, la pública (.pub que es la que distribuimos en las máquinas que nos queramos autenticar) y la privada.

La idea es cifrar un paquete con la llave privada, y la otra máquina, será capaz de descifrarlo ya que tiene la llave pública.

Copiaremos la llave pública (de CentOS a Ubuntu) con **ssh-copy-id (usuario@)192.168.56.105 -p 22022**
yes
Ahora nos pedirá el password de nuestro usuario: **ISE**
Ahora, si hacemos quit e intentamos hacer **ssh (usuario@)192.168.56.105 -p 22022** ya no nos pedirá la contraseña.

Luego, deshabilitaremos el acceso con contraseña:
sudo nano /etc/ssh/sshd_config
(escribimos encima de #permitemptypasswords): **PasswordAuthentication no**
systemctl restart sshd (reiniciamos para que se aplique el cambio)

Y comprobamos intentando entrar desde CentOS (que ya tiene acceso) y por ejemplo desde la WSL (no tiene forma de autenticarse).

Para permitir que la WSL subiese su llave pública al servicio ubuntu para poder acceder sin contraseña, deberíamos modificar el archivo que modificamos antes, ponerlo a **yes systemctl restart sshd**, hacer **ssh-copy-id (usuario@)192.168.56.105 -p 22022** y volvemos a desactivar la opción, poniéndolo a no y **systemctl restart sshd**

2) Otra medida de seguridad que haremos hoy, es dejar a unos únicos usuarios que puedan acceder (loggearse)

Creamos usuario en CentOS:
sudo adduser nico

sudo passwd nico

Luego, intentaremos acceder al servicio con **ssh nico@192.168.56.105 -p 22022** pero no nos deja, primeramente porque se deshabilitó el acceso por contraseña y luego porque la máquina que tiene el servicio, no tiene el usuario nico.

Creamos el usuario nico en ubuntu con **sudo adduser nico**, **ISE, enter** en el resto de valores.

Volvemos a permitir el acceso con contraseña y hacemos **systemctl restart sshd** y ya podemos entrar como nico a ubuntu.

Lo que queremos es que solo se pueda acceder con nuestro usuario (alvaro), para esto:
sudo nano /etc/ssh/sshd_config

Y escribimos: **AllowUsers alvaro**
systemctl restart sshd

Volvemos a intentar entrar como nico y vemos como no nos deja pero con alvaro como usuario si nos deja.

Ya tenemos configurado los aspectos mínimos de seguridad.

/-----Utilidades Extra-----\

a) Ahora usaremos la utilidad **sshfs**. Montaremos en nuestro equipo, de forma transparente, una ubicación remota. Además, el acceso a esa información será mediante ssh garantizando la seguridad.

(NOTA: él está desde fedora, quizás en la WSL es distinto)
Desde la WSL hacemos **dnf provides sshfs** y vemos que el paquete que podemos utilizar para instalarlo:

sudo dnf install fuse-sshfs-3.7.1-2.fc34.x86_64
($ sudo apt install sshfs creo que es en ubuntu)

En este caso, crearemos una carpeta (en la WSL) **mkdir ./ubuntuserver**
sshfs alvaro@192.168.56.105:/home/alvaro ./ubuntuserver/ -p 22022

Ahora tenemos el contenido de ubuntu server pero en la máquina de WSL (podemos hacer mount para verlo). Si por ejemplo creamos un archivo con **touch Hola**, a los segundos, nos debería salir en el directorio que tenemos en la WSL.

b) Otra funcionalidad que implementaremos será la de **XForwarding**, que la usaremos para:
En nuestro server tendremos una aplicación que haga uso de la interfaz de ventana. Pues XServer, en vez de que las peticiones las haga al servidor local, se reenvían al servidor de nuestro cliente, y esta interfaz le aparece al cliente. Si bien, la ejecución se está haciendo en el servidor.

Desde la WSL me loggeo en el ubuntu server y añadimos la opción -X, haciendo el xforward (**ssh (usuario@)192.168.56.105 -p 22022 -X)**. Entonces con gedit o un

navegador, tiene sentido la opción -X (que además es segura). Así, si instalamos el gedit (que no está, con **sudo apt install gedit**) y hacemos gedit, vemos como nos sale en el cliente la interfaz.

Destacamos que la interfaz la ejecuta el cliente, pero el procesamiento interno lo hace el servidor. Al guardar, vemos que guarda en el servidor como bien podríamos pensar.

Por último, dos utilidades:
c) screen y tmax -> nos permiten lanzar un trabajo en una terminal y dejar esa terminal sin ninguna sesión vinculada, y volverla a retomar cuando queramos. Si hacemos una ejecución que requiere mucho tiempo, nos desconectamos y conectamos cuando queramos y retomamos el control. Otro escenario es ejecutar un procedimiento en el que se nos va la luz y perdemos la conexión y esto nos obligaría a reiniciar el procedimiento.

Vamos a la WSL y nos conectamos a ubuntu server: **ssh (usuario@)192.168.56.105 -p 22022)**.
Creamos un archivo **nano miarchivo** y si vamos a la máquina con ubuntu, hacemos **ps**, podemos ver que sshd tiene su bash y su aplicación nano editando el archivo. Si cerramos la pestaña de la WSL y volvemos a hacer **ps**, vemos como los procesos anteriores se han terminado.

Para evitar que se terminen, usaremos screen. Nos logueamos en ubuntu y ponemos la orden **screen** y ya hacemos el **nano**. Hacemos **ps ax** en el servidor y vemos que bash ha invocado el comando screen, que ha invocado otro proceso SCREEN, del cual cuelga otro bash y el nano. Si cerramos la pestaña de la WSL y hacemos **ps ax**, vemos que sigue estando SCREEN, donde cuelga el bash y el nano.

Para recuperar el archivo (lo podemos hacer como clientes o en el servidor), nos logueamos de nuevo en ubuntu desde la WSL, hacemos **screen -r -d** recuperamos la última ventana con los cambios que hicimos.

Es importante decir que podemos tener varios screen (CTRL+A + CTRL+D para cerrar). Podemos hacer un listado para verlos con **screen -list** y nos conectamos con **screen -r (números.pts-0.ise_ubuntu) -d**

Nota: solo puede tener una ventana de un archivo a la vez. Si no, hace detached y nos echa de la otra.

tmux es casi igual..
Hacemos **tmux** en la terminal
CTRL+B Crea dos sesiones
CTRL+B + CTRL+D = detached

Vídeo 2: Fail2ban

Servicio con el que interactuaremos con systemd y con un comando llamado fail2ban.client

Lo que hará fail2ban, será hacer un sondeo de los archivos que están en /var/log y analiza el contenido de esos archivos para ver las autenticaciones erróneas que se han producido.

En caso de que haya una autenticación erronea, baneará al usuario que ha fallado durante un tiempo determinado (configurado en el archivo de configuración que se encuentra en /etc/)

Este servicio es importante sobretodo para impedir ataques de fuerza bruta.

Procederemos a la instalación de este servicio en CentOS:

dnf search fail2ban -> fail2ban no tiene paquete propio, se encuentra en el conjunto de paquetes extendido

dnf search epel
sudo dnf install epel-release
ISE
s

Ahora si podemos buscar fail2ban
dnf search fail2ban
sudo dnf install fail2ban
s
s

Ya tendríamos instalado fail2ban. Ahora veremos el estado del servicio:
systemctl status fail2ban
sudo systemctl enable fail2ban -> se active en el próximo reinicio
sudo systemctl start fail2ban -> que se active
systemctl status fail2ban -> ya estará el servicio en ejecución

Procederemos a configurarlo. Tenemos el comando fail2ban-client. Se analizarán los distintos logs, se buscarán las direcciones IP, las cuales pasarán a una serie de jails que definiremos para cada servicio.

Las definimos (las cárceles) en el archivo de configuración:
cd /etc/fail2ban
sudo cp -a jail.conf jail.local (no editamos el archivo porque si actualizasemos, se borraría)
sudo nano /etc/fail2ban/jail.local

Buscamos [sshd] (el que no esta comentado) y escribimos en otra línea: enabled = true
sudo fail2ban-client status sshd -> vemos como aun no está activada la cárcel
sudo systemctl restart fail2ban.service
sudo fail2ban-client status sshd -> ya está operativa

Podemos comprobar que funciona, entrando desde otra terminal como la WSL:
ssh 192.168.56.110 -p 22022 (en WSL) (introduzco una contraseña incorrecta 3 veces).
sudo fail2ban-client status sshd (CentOS) -> ya aparece 3 intentos fallidos de acceso. Nos banea, por defecto, a los 5 intentos fallidos. Podemos editarlo en el archivo de configuración.
Volvemos a hacer **ssh 192.168.56.110 -p 22022 (WSL)** (y fallamos 3 veces más)
sudo fail2ban-client status sshd (CentOS) y nos aparece que ha baneado a la IP.

ssh 192.168.56.110 -p 22022 (WSL) -> Pero si ponemos bien la contraseña, nos podemos logear. El problema ha sido que no hemos modificado el puerto. **sudo nano /etc/fail2ban/jail.local** y volvemos a **buscar [sshd] y en port, escribimos 22022**
sudo systemctl restart fail2ban.service

ssh 192.168.56.110 -p 22022 (WSL) -> y ahora no nos debe ni dejar poner la contraseña.
sudo fail2ban-client status sshd (CentOS) -> sigue baneada la WSL

Ahora, debemos ver como sacar una IP de baneadas:
sudo fail2ban-client set sshd unbanip 192.168.56.100
sudo fail2ban-client status sshd -> no está la IP en la lista.
ssh 192.168.56.110 -p 22022 (WSL) -> nos volvería a dejar

Ya luego, podemos modificar en el archivo de configuración .local ciertos parámetros como:

Tiempo de baneo de una IP (bantime)
Número de intentos antes de banear una IP (maxretry)

Lo bueno de fail2ban, no solo puede usarse con sshd, si no, con más servicios que tengan archivos por el log.

Vídeo 3: Instalación de LAMP

Lo haremos en **CentOS**.

Pila porque es: datos (BD) , sobre esto la lógica que interactúa con los datos (el LP) y una interfaz que es el servidor web (Apache suele ser).

-- Nota histórica en el vídeo hasta 9:35

Instalaremos en CentOS la pila LAMP: (en ubuntu es de otra forma)
(sudo ifup enp0s3)
dnf search apache
sudo dnf install httpd

ISE
s
s

Comprobamos con **systemctl status httpd** su estado
sudo systemctl enable httpd (que este activo en el próximo reinicio)
sudo systemctl start httpd (iniciar ahora)
systemctl status

Probamos que funciona con el comando curl (para ver una url)
curl localhost, vemos que nos devuelve una página web

dnf search php
sudo dnf install php
s
(Esto es un intérprete, no un servicio, debemos provar de otra forma)
php -a

Ahora instalamos la BD
dnf search mariadb
sudo dnf install mariadb -> instala cliente
sudo dnf install mariadb-server.x86_64 -> el servicio

systemctl status mariadb -> está desactivado tras instalarlo. Muestra estado de servicio
sudo systemctl enable mariadb
sudo systemctl start mariadb

mysql (= MariaDB)
Problema: no podemos conectarnos al servidor con nuestro usuario
mysql -u root (si nos deja como root y sin contraseña)

Tras la instalación, el manual recomienda ejecutar **mysql_secure_installation** (que es un script) para ganar seguridad.

enter
Y
ISE (contraseña del root)
ISE
Y (borrar usuario anónimo)
Y (quitar root de mysql de forma remota)
Y (borrar tabla de prueba que viene)
Y (recargar tabla de privilegios)

Con esto tenemos configurado la base de datos de MariaDB. Comprobamos:

mysql
mysql -u root
mysql -u root -p [contraseña] (si no ponemos [] nos la pide ahora)

Probamos a conectarnos con apache. Abrimos la WSL y hacemos **curl 192.168.56.110** (no nos deja) pero **ping 192.168.56.110** si funciona.

El problema es causado por el firewall:

sudo firewall-cmd --add-port=80/tcp (ahora)
sudo firewall-cmd --add-port=80/tcp --permanent(para las siguientes ocasiones)
sudo firewall-cmd --reload

curl 192.168.56.110 (desde WSL) y ya si nos devuelve la conexión.

Nos faltaría probar que podemos integrar un script de php que se conecta a una BD y que nos muestra el resultado a través de html.

https://www.php.net/manual/es/function.mysqli-connect.php
Copiaremos el primer script que aparece en la web anterior para probar si hay conexión.
Nos creamos otra terminal en la WSL y nos conectamos: **ssh alvaro@192.168.56.110 -p 22022**

Tenemos que ubicar nuestro archivo, en un lugar que sea accesible con http:
less /etc/httpd/conf/httpd.conf

Buscamos el parámetro DocumentRoot, y nos pondrá el directorio donde tenemos que poner el script. (/var/www/html)

cd /var/www/html
sudo nano index.php

Insertamos el código de la web. Aunque probablemente tengamos que cambiar los parámetros de la función mysqli_connect. (segundo y tercer parámetro por root, ISE)

mysql -u root -p
ISE
CREATE DATABASE mi_bd; (ya tenemos la BD que aparece en el parámetro)

Vamos a navegador y ponemos la ip de ubuntu:
192.168.56.110 -> va bien
192.168.56.110/index.php -> lo sirve como documento de texto, con el riesgo que conlleva.

Problema: apache no está interpretando el php. Modificaremos la configuración de php:
en CentOS:

sudo nano /etc/httpd/conf/httpd.conf
ISE

Buscamos DirectoryIndex y añadimos después de index.html ***.php** (para que sirva cualquier archivo con ext php)
Hacemos **f5 en el navegador** y vemos que no ha cambiado nada. Reiniciamos el ervicio

sudo systemctl restart httpd
f5 y vemos que hay error http error 500 (error del servidor).

Vamos a CentOS y ejecutamos el archivo:
php index.php (da error ya que no hemos instalado la biblioteca que conecta php y sql)

sudo dnf search php | grep mysql
sudo dnf install php-mysqlnd
s

Probamos de nuevo:
php (/var/www/html)index.php (y ya al menos funciona en el servidor)
f5 y vemos que hay error de permisos

Por ver xd: **sudo less /var/log/audit.log y buscamos un httpd por el final.**

SE es que no da el acceso.
getsebool -a | grep httpd
sudo setsebool httpd_can_network_connect_db=on

getsebool -a | grep httpd (vemos que está a on)
Hacemos **f5** y vemos que tenemos la pila LAMP configurada en nuestro servidor.

NOTAS POST VIDEO + APUNTES

Para examen traer ubuntu y centos con comunicación ya probada.

Integrar código php en html:
<?php
...
?>

LAMP es un stack de desarrollo. Hay otras como XAMP o WAMP.
LAMP: Linux, Apache, MySQL o MariaDB, Php (o Python)
= SO, Servidor, Base de Datos y Lenguaje de Programación

Nginx -> óptimo con E/S asíncronas. Se suele usar para acceso a archivos estáticos tipo estáticos como fotos, texto…
Apache -> Óptimo para contenido dinámico como php o django.

Tendremos que instalar LAMP en ubuntu (que es instalar un paquete) y LAMP en CentOS (paquete a paquete: Apache, MySQL y Php).

Ubuntu según **internet**:
sudo apt update; sudo apt upgrade
sudo apt install -y apache2 apache2-utils
sudo ufw allow http (abrir puerto en el firewall)
sudo apt install mariadb-server mariadb-client
systemctl status mariadb
sudo systemctl start mariadb
sudo systemctl enable mariadb
sudo mysql_secure_installation (hacer como en CentOS)

sudo apt install php8.0 libapache2-mod-php8.0
sudo systemctl restart apache2

Ubuntu (lo que hicimos en clase, se parece bastante a lo de internet de arriba)

> sudo apt-get install apache2
(Si no fufa hacer un sudo apt-get update)

> sudo systemctl start apache2
> sudo systemctl status apache2 (comprobar si todo ha ido bien)

- El firewall está tapado si al poner la dir IP de Ubuntu en el navegador no sale nada
> sudo ufw allow 80
- Y debería de ir a poner 192.168.56.105 en el nav.

En esa pagina no se necesita ningún lenguaje porque es una página estática en la que sus datos no cambian. Es como si fuera un blog

- Instalar tanto mysql como php y comprobar si fufa
> sudo apt-get install mysql-server mysql-client
> sudo apt-get install php
> sudo systemctl start mysql
> sudo systemctl status mysql
(php no hace falta)