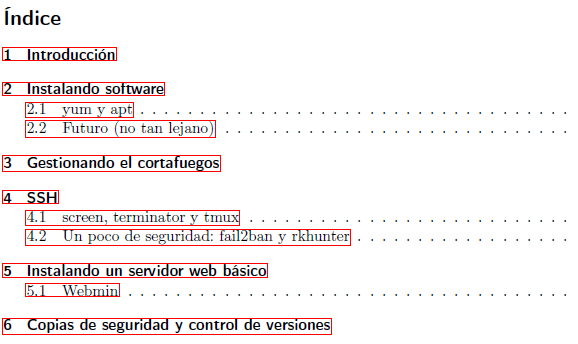
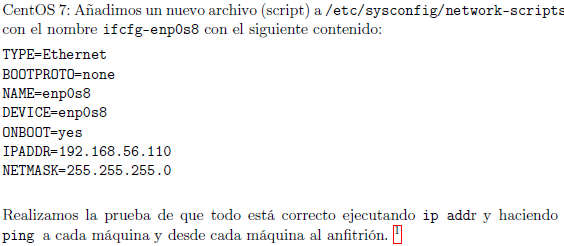
# PRÁCTICA 2: Instalación y configuración de servicios



Antes de comenzar debemos configurar las máquinas para que puedan comunicarse entre sí. Esto ya lo hicimos para la máquina con UbuntuServer20 (punto 2.2 Configuración de red de la Práctica1), ahora lo haremos para CentOS (en este caso usaremos la máquina del LVM por velocidad de inicio, ya que no tenemos que descifrar nada).

Después de añadir el adaptador de red en la máquina con CentOS desde el menú de *Configuración/Red/Adaptador2/Solo-anfitrión...* (asegurándonos que es el mismo que usamos en UbuntuServer). Arrancamos la máquina y nos vamos a los ficheros de configuración de los adaptadores de CentOS.

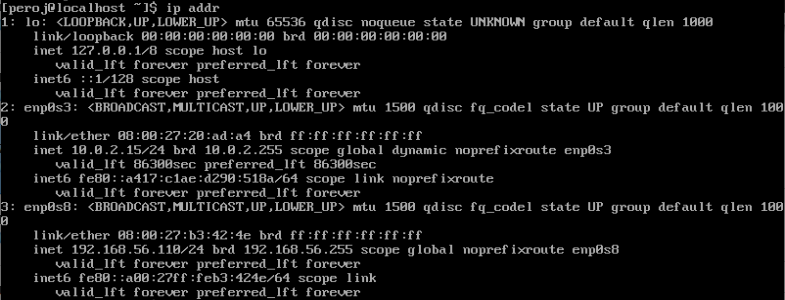


También deberemos modificar el archivo *ifcfg-enp0s3* para que cada vez que se inicia el sistema se levante la red NAT (conexión a internet), poniendo a yes el parámetro ONBOOT.

IP UbuntuServer --> 192.168.56.105

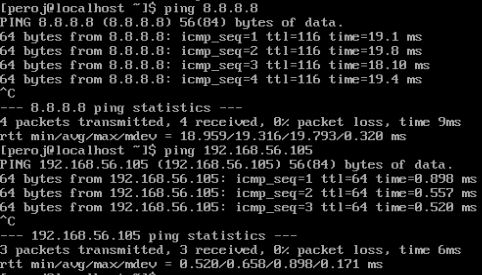
IP CentOS-LVM --> 192.168.56.110

Comprobamos reiniciando la máquina (en vez de usar el ***ifup***). Para ver el estado de la red en CentOS usamos ***ip addr***, y vemos que están ambas con *state UP* (en Ubuntu usamos ***ifconfig***).

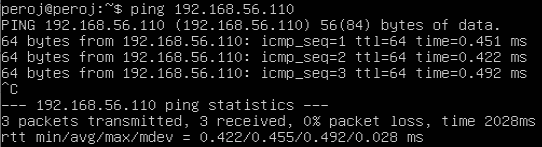


Ahora iniciamos la máquina con UbuntuServer y le hacemos ping (comprobando el adaptador *enp0s8*); también le hacemos ping a Google (comprobando el adaptador *enp0s3*) y viceversa.

desde CentOS



desde UbuntuServer



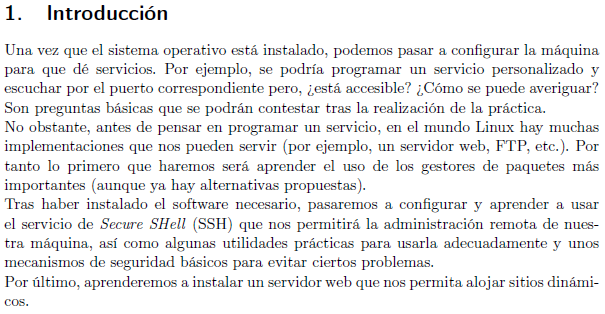
También comprobamos desde el host hasta las dos máquinas y viceversa.

Ya podemos proceder con la práctica2, donde probaremos servicios entre ambas máquinas.

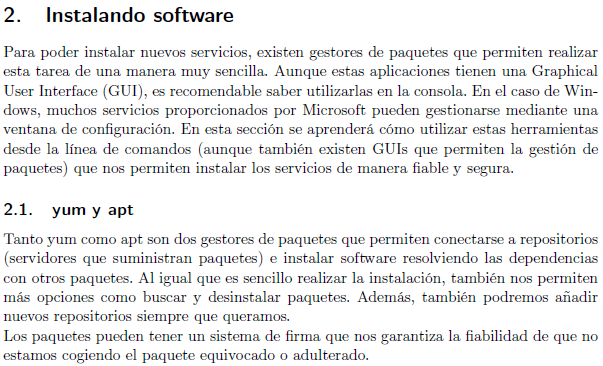
Un servicio es algo que sirve, está siempre a la escucha por un puerto para recibir peticiones, procesarlas y servir algo.

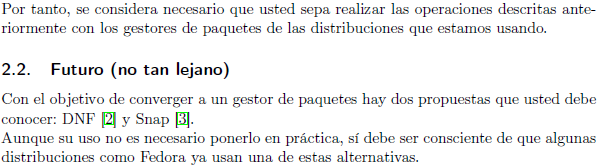
## LECCIÓN1:

1. **Introducción**
2. **Gestores de paquetes Linux: yum, apt y más**
3. **Gestionando el cortafuegos[[1]](#footnote-1)**
4. **SSH (Secure Shell)**

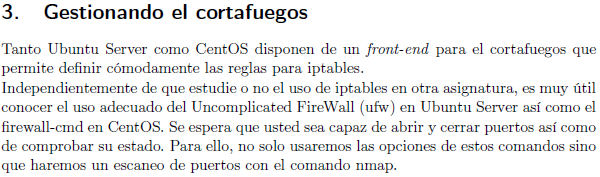


Para poder administrar de forma remota un servidor de manera segura se usa actualmente *SSH* (*Secure Shell*), que desarrolló el protocolo un finlandés porque el protocolo usado antiguamente Telnet no llevaba ningún tipo de cifrado.

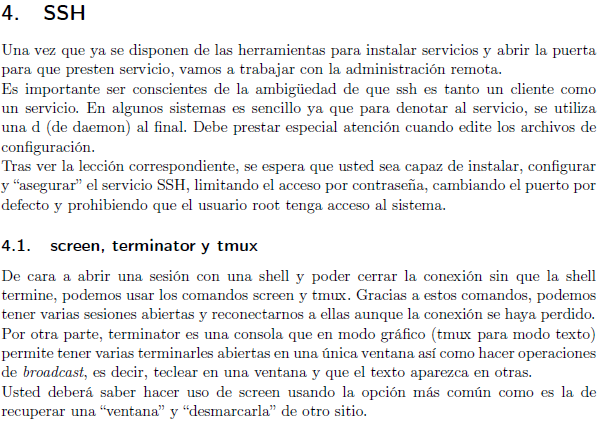


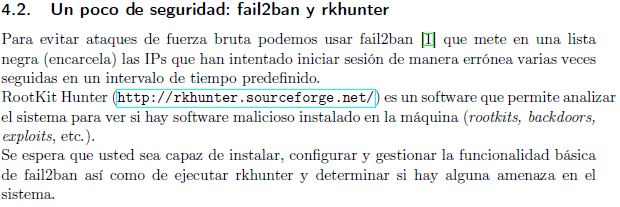


Tanto *apt* como *yum* se conectan al repositorio y estos le suministran los paquetes y además, estos intentan resolver las dependencias, con el problema de que dependen mucho de servidores externos para las dependencias, y esto aparte de que tarda, puede causar los problemas de que si los servidores caen, las dependencias caen y por lo tanto no poder instalar bien ese software. Por lo que se han ido desarrollando dos nuevos gestores de ficheros que ahora mismo conviven con *apt* y *yum*, y la principal diferencia es que solucionan el problema anterior, dado que ya las dependencias no tratan de resolverlas en servidores externos sino en los propios de la empresa (Canonical y RedHat). También son bastante más livianos tanto DNF (29000 lineas de codigo Python, C y C++) como *Snap* respecto a *yum* (50000 lineas de codigo Python) y *apt*.



Para gestionar esta barrera de seguridad podemos usar el modulo *iptables* (bajo nivel), también podemos usar abstrayéndonos un poquito con las herramientas *ufw* en Ubuntu y el *firewall-cmd* en CentOS.





Normalmente en un servidor tienes ambos (aunque en un servidor como tal, normalmente no tendrás instalado el cliente

* ssh servidor --> Para que puedas conectarte a esa máquina desde otras (todos los ficheros de configuración que tienen que ver con el servidor se nombran un una ***d*** (daemon) al final (***sshd***).
* ssh cliente --> Para conectarte desde esa máquina a otras (se nombran normal ***ssh***).

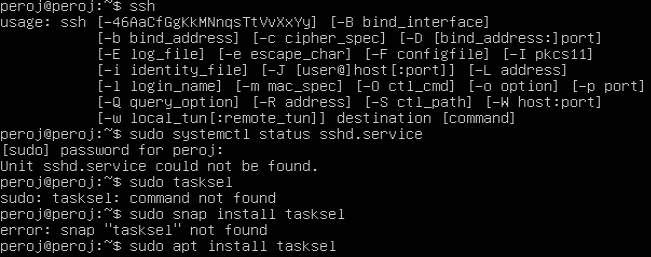
Herramientas interesantes:

* *screen* --> Sirve para que cuando te conectas por *ssh*, si cierras la ventana, automáticamente lo que se esté haciendo por *ssh* se corta (por ejemplo, una descarga) y el proceso se mata, ya que *ssh* no permite ejecutar en segundo plano. Para esto está screen, permitiéndote abrir *ssh* en segundo plano, creando en background una sesión de ssh para que no se corte.
* *fail2ban* --> Prevé de ataques de fuerza bruta y denegación de servicio, ya que se queda escuchando en un puerto y si una IP se intenta conectar más de 3 veces la banea, metiéndola en una lista negra para que durante un tiempo que tu defines no pueda volver a hacerte ninguna petición.
* rkhunter (*RootKit Hunter*) --> Busca vulnerabilidades del sistema (en versiones, software, kernel...) para ver si tienes puertas traseras, rootkits, exploits...

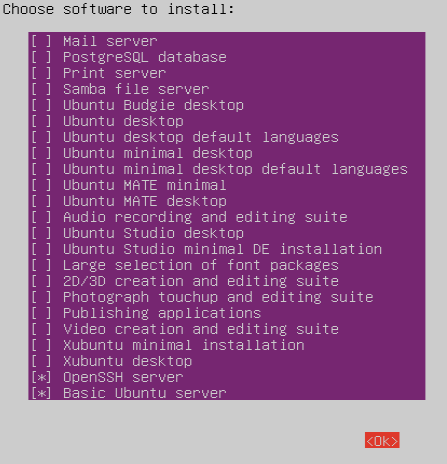
Instalación y configuración de ssh en ambas máquinas

\*Ubuntu Server --> Es más rápido de configurar el “ssh servidor” en Ubuntu debido a que suele ser menos seguro que CentOS dado que es más “user friendly” (te deja hacer cosas sin necesidad de avisar al sistema de los cambios que estás haciendo, al contrario que en CentOS).

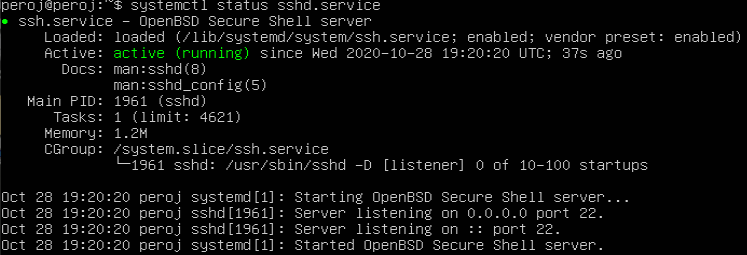
El cliente lo trae instalado, lo vemos con ***ssh***. Pero el servidor no lo trae, lo vemos con ***sudo systemctl status sshd.service***. Entonces lo instalamos, pero en vez de hacerlo con *snap,apt...* lo haremos con ***tasksel*** (instalándola previamente con *apt*). Es una especie de interfaz gráfica en la que se puede seleccionar qué herramientas quieres que te instale el sistema (casi todas tienen que ver con gestión de servidores).



Ejecutamos ***sudo tasksel***. Y seleccionamos el OpenSSH server (marcamos con barra espaciadora, tabulamos y le damos a ok).

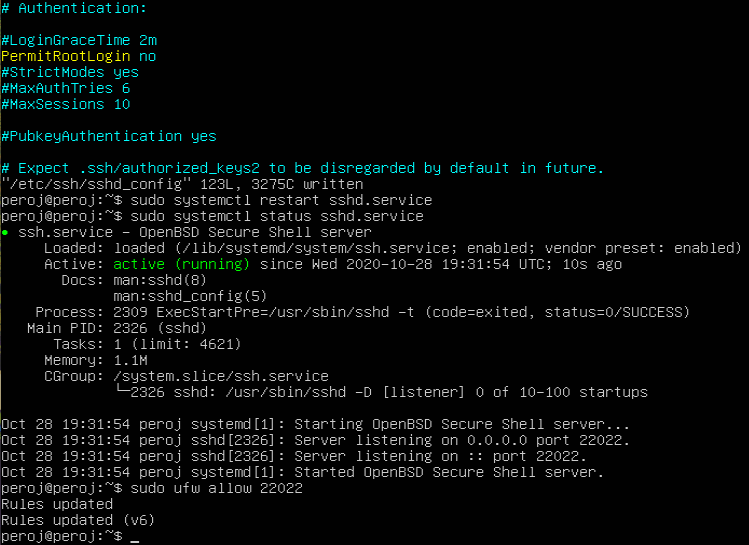


Comprobamos y vemos que está activo y por defecto, en el puerto 22.



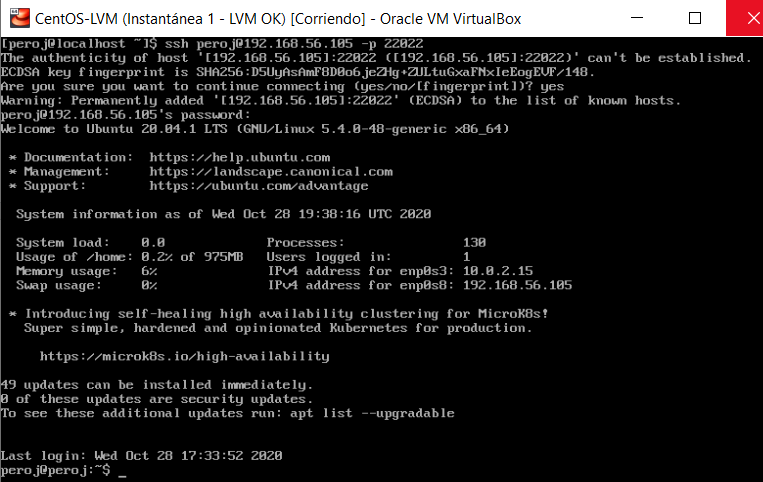
Vamos a cambiarle el puerto por defecto porque evidentemente todo aquel que quiera atacar por ssh lo primero que va a hacer es atacar al puerto 22. Abrimos ***/etc/ssh/sshd\_config*** y cambiamos el puerto al *Port 22022* y también cambiaremos en *PermitRootLogin* el *prohibit-password* por *no.* No queremos permitir que se conecte alguien como root por ssh ya que root es un usuario por defecto del sistema, el que tiene acceso a todo, y solo le faltaría saber la contraseña, porque ya sabría si no lo cambiamos el usuario y el puerto al que atacar. Para gestionar el sistema lo gestionaremos desde nuestro usuario ya que está en el fichero *sudoers*.

Después de guardar el fichero, reiniciamos el servicio con ***systemctl restart sshd.service*** para aplicar los cambios y lo vemos con ***systemctl status sshd.service***.

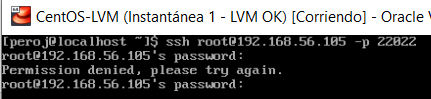


Nos aseguramos que el puerto 22022 está abierto con el firewall ***ufw allow 22022*** *(que* modifica las *iptables* para que nos permita este puerto)

Lo comprobamos desde la máquina de CentOS dado que lo que hemos configurado es el *sshd* del lado del servidor para que se puedan conectar a la máquina con Ubuntu.



También comprobamos que no nos deja entrar como root.



Para añadirle un punto más de seguridad también vamos a quitarle las contraseñas añadiendo un par de claves pública-privada para CentOS-Ubuntu de modo que cada vez que se haga un ssh de CentOS a Ubuntu, el servidor (Ubuntu) ve que la clave de la petición (CentOS) es válida dado que ya está registrada en el sistema:

1 - Generamos la clave en el cliente (CentOS) con ***ssh-keygen*** .

2 - Se la mandamos al servidor con ***ssh-copy-id*** [***peroj@192.168.56.105***](mailto:peroj@192.168.56.105) ***-p 22022*** , metemos la contraseña de Ubuntu y vemos que se ha añadido una clave.

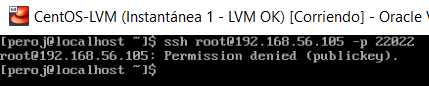
*3 - Comprobamos haciendo login con* ***ssh*** [***peroj@192.168.56.105***](mailto:peroj@192.168.56.105) ***-p 22022*** *, ya no nos pide la contraseña.*



Por último, nos vamos al fichero de configuración de Ubuntu (***sudo vi /etc/ssh/sshd\_config***) y desactivamos la opción *PasswordAutentication no*, para que así ya no exista ninguna manera de que te puedas conectar al servidor a través de contraseña, la única manera de conectarse al servidor es si previamente este tiene la clave pública/privada.



Reiniciamos el servicio con ***sudo systemctl restart sshd.service*** y ya estaría el sistema con el máximo de seguridad posible. Lo comprobamos desde CentOS como root (como vemos, ya ni nos pide la contraseña):

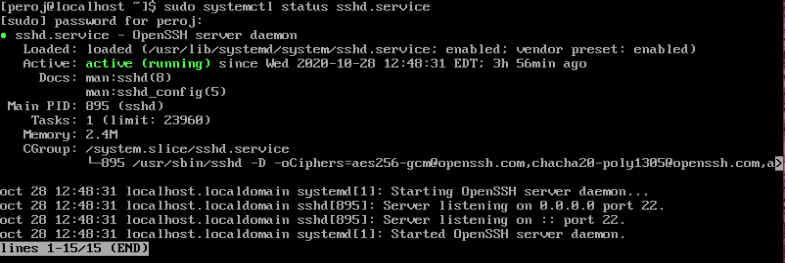


\*NOTA: Si ahora quisiésemos dar de alta a un nuevo usuario, el administrador tendría que irse al servidor, volver a poner el *PasswordAutentication* a *yes*, generar la clave pública/privada del nuevo usuario cliente, mandarsela al servidor y después volver a poner el PasswordAutentication a no en el servidor.

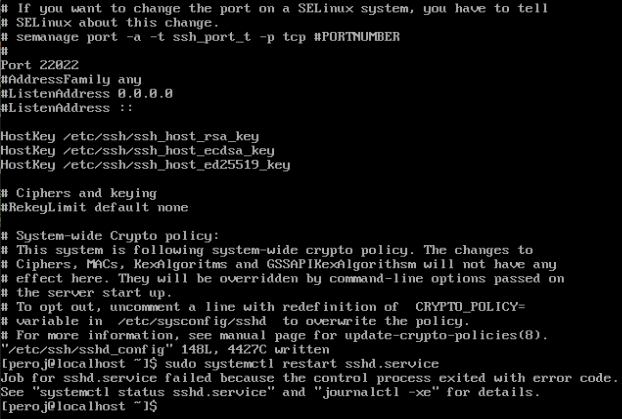
\*CentOS --> Es más seguro, para todo cambio de puertos y configuraciones requiere que se avise al SELinux. Trae tanto ssh cliente como ssh servidor dados de alta por defecto.

Haremos el mismo proceso anterior en CentOS para configurarle el ssh server y podernos conectar desde UbuntuServer.

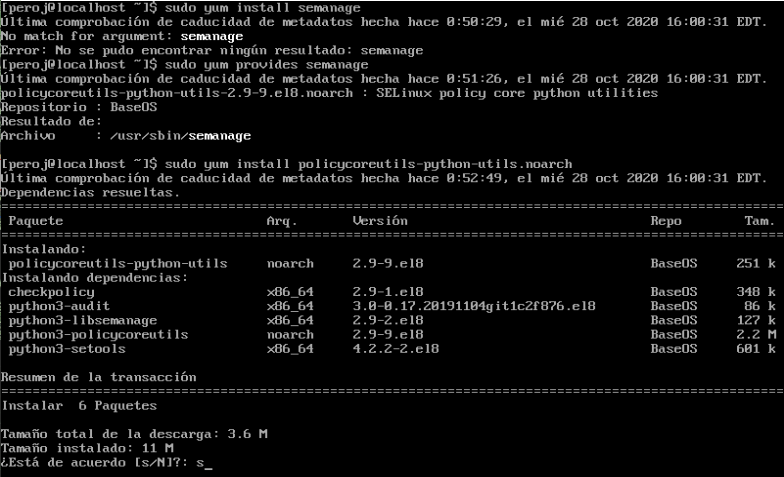
Comprobamos que esta dado de alta el servicio con ***sudo systemctl status sshd.service***.



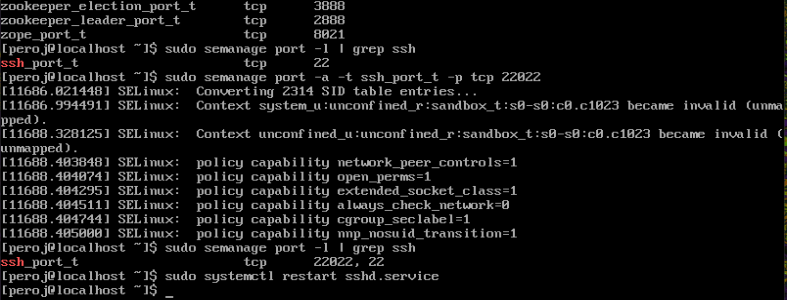
Vamos a cambiarle el puerto 22 que trae por defecto por el 22022 desde el fichero ***sudo vi /etc/ssh/sshd\_config.*** Una vez hecho esto, debemos avisar al sistema de seguridad de Linux SELinux de que vamos a cambiar el puerto con la orden ***semanage port -a -t ssh\_port\_t -p tcp 22022***.



El problema es que ***semanage*** no viene instalado en CentOS, por lo que vamos a instalarlo con ***sudo yum install semanage.*** Otro problema más, no lo encuentra, pasamos a buscarlo con ***sudo yum provides semanage*** que lo que hace es buscar los paquetes donde se encuentra esta herramienta, por lo que pasaremos a instalar el paquete que nos muestra como sigue:



Ahora con ***semanage*** indicamos al sistema de seguridad SELinux que hemos cambiado el puerto de sshd a 22022 para que lo admita como nuevo puerto de ssh. Para listar los puertos que está usando el sistema podemos usar ***semanage port -l***, pero queremos saber el de ssh por lo que se lo pasamos al ***grep***. Vemos que para CentOS el tipo de puerto es ***ssh\_port\_t***, por tanto, así se lo indicaremos al SELinux con ***sudo semanage port -a -t ssh\_port\_t -p tcp 22022***. Y comprobamos que lo ha cambiado y reiniciamos el servicio con ***sudo systemctl restart sshd.service***.

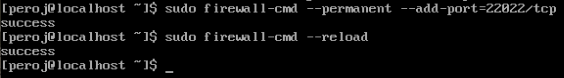


Después de comprobar también con ***sudo systemctl status sshd.service*** que el servicio está activo (running) si ahora nos vamos a Ubuntu e intentamos conectarnos por ssh a CentOS con no nos dejaría porque en el firewall de CentOS no está abierto ese puerto todavía. Lo único que hemos hecho ha sido indicárselo al sistema de seguridad de Linux.

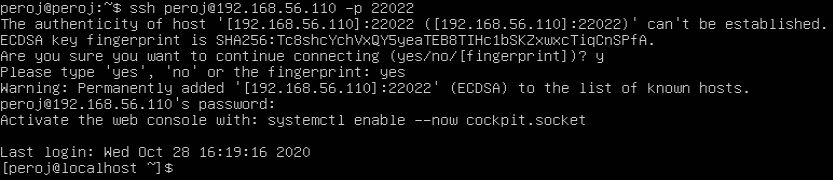
Por lo que ahora avisamos al cortafuegos de que permita conexiones por ese puerto. Para esto existen dos formas de hacerlo en CentOS:

1ª ***sudo firewall-cmd --add-port=22022/tcp*** (para abrir puertos de forma puntual mientras el sistema no se reinicie, pero no hace falta recargar las reglas del firewall manualmente)

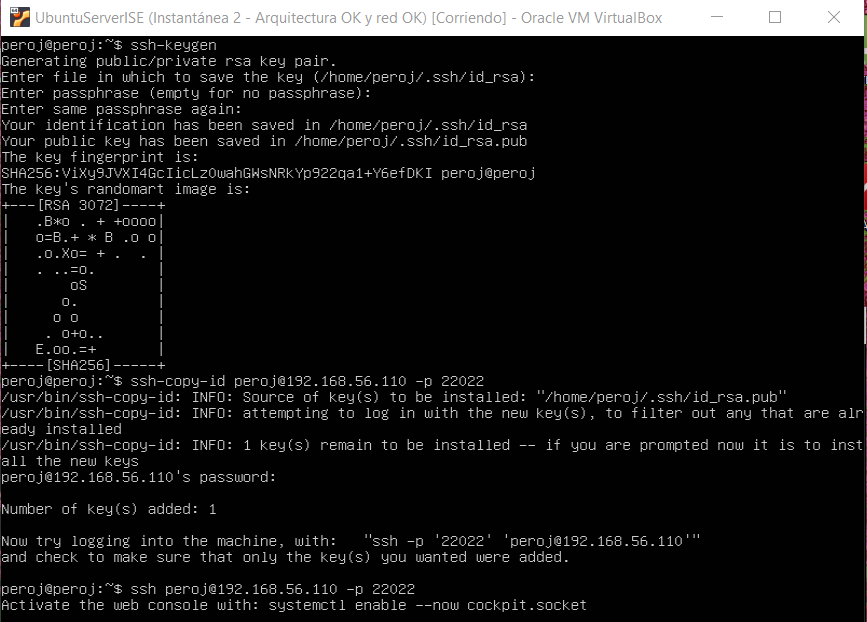
2ª ***sudo firewall-cmd --permanent --add-port=22022/tcp*** (a partir de este momento lo dejará abierto para siempre a partir de que el firewall se recargue, hay que recargarlo manualmente con ***sudo firewall-cmd --reload***)



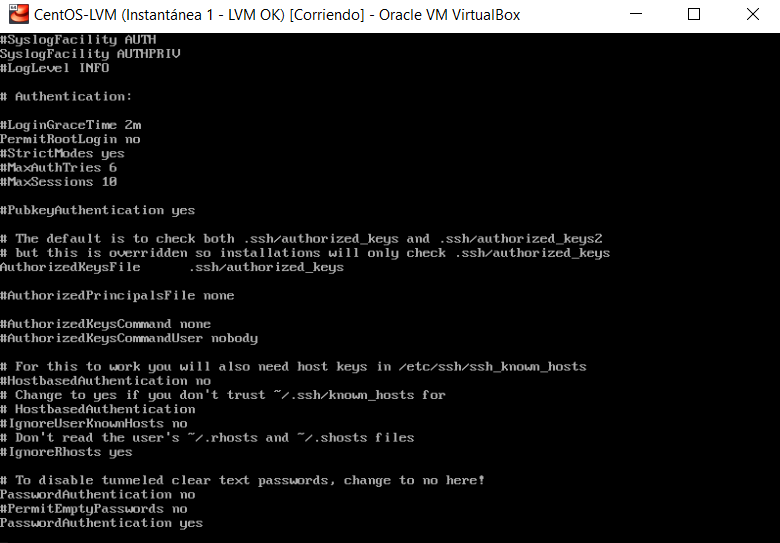
Ahora ya si intentamos conectarnos desde Ubuntu a CentOS con ***ssh*** [***peroj@192.168.56.110***](mailto:peroj@192.168.56.110) ***-p 22022*** ya podemos sin problema.



Ahora debemos generar desde Ubuntu la clave pública-privada y enviársela a CentOS para que no sea necesaria la contraseña.



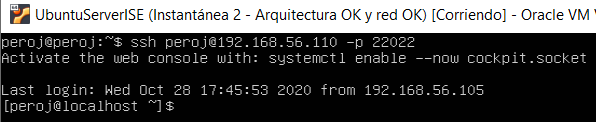
Y finalmente, le cambiamos del fichero de configuración de CentOS ***sudo vi /etc/ssh/sshd\_config*** el *PermitRootLogin* a *no* y también le ponemos a *no* el *PasswordAuthentication,* para que solo se pueda conectar un usuario que previamente esté vinculado con una clave pública-privada. Y reiniciamos el servicio con ***sudo systemctl restart sshd.service***.





Por último, comprobamos que de CentOS a Ubuntu tenemos conexión sin contraseña por ssh y viceversa.





**LECCIÓN2:**

1. **Copias de seguridad y control de versiones**

Copias de seguridad

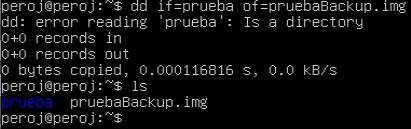
Primeramente, nos creamos en UbuntuServer, un directorio con 3 ficheros de prueba:



Tenemos varias formas para hacer copias de seguridad en Ubuntu, para uso a nivel local o para pequeños volúmenes de datos podemos usar:

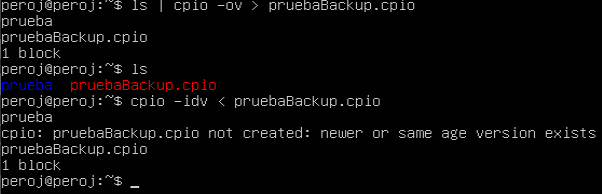
1. Copia binaria (dd)

* ***dd*** :
  + Útil para copiar bit a bit el contenido (forma más rudimentaria)
  + Ejemplos: (dd “entrada” “salida”)
    - ***dd if=/dev/sda of=/dev/sdb*** (Copia a otro dispositivo)
    - ***dd if=/dev/sda of=~/hdadisk.img*** Copia una imagen de sda.
      * Para recuperarla: ***dd if=hdadisk.img of=/dev/sdb***
      * Se pueden especificar particiones: *sda1* y *sda2*



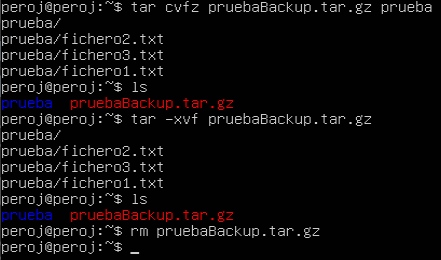
1. Copiar archivos y empaquetar (cp, cpio, tar)

* ***cpio*** :
  + Copiar archivos a y desde (archivos)
    - ***ls | cpio -ov > /tmp/object.cpio*** (hace una copia del directorio actual, con -ov le indicamos donde queremos que guarde la copia)
    - ***cpio -idv < /tmp/object.cpio***



No lo restaura dado que no hemos modificado nada de la original.

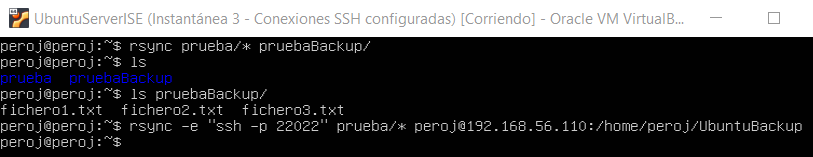
* ***tar*** :
  + Ejemplos:
    - ***tar cvzf MyImages-14-09-17.tar.gz /home/MyImages***
    - ***tar -xvf public\_html-14-09-17.tar***
    - ***-c*** (Crea un nuevo archivo archivando los directorios recursivamente)
    - ***-Z*** (Para comprimir)
    - ***-v*** (imprime lo que ha hecho)

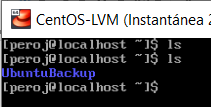


Las herramientas anteriores, aunque podrían, no están pensadas para mover gran cantidad de datos, con la siguiente herramienta podemos hacer copias de seguridad con mayor volumen de datos y a nivel de pequeña/mediana empresa:

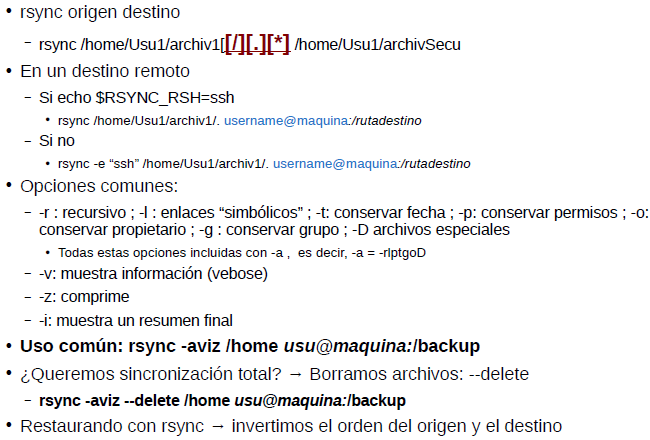
1. Sincronizar (rsync, rsnapshot)

* ***rsync*** :
  + Permite movimientos de datos muy grandes.
  + Además esta herramienta permite enviar datos por ssh, permitiendo, por ejemplo, mandar una copia de seguridad periódicamente de tu servidor a tu ordenador personal.
  + Ejemplos:
    - ***rsync prueba/\* pruebaBackup/***  (Hace copia en el servidor local)
    - ***rsync -e “ssh -p 22022” prueba/\**** [***peroj@192.168.56.110:/home/peroj/ubuntuBackup***](mailto:peroj@192.168.56.110:/home/peroj/ubuntuBackup)(con el ***-e*** le indicamos que va a ir por ssh por el puerto 22022, carpeta origen, servidor ssh (CentOS), carpeta destino)
    - ***rsync -a --delete source/ /destination/***





* + Sincroniza una fuente y un destino (incluso 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



* Otros software para realizar Backups:
  + Tartarus
  + Backup2l
  + Duplicity
  + Sftpclone

Control de versiones (o cambios)

Para evitar problemas con posibles copias de seguridad y sincronizaciones erróneas o perdidas y software de recuperación de información de discos duros dañados o borrados (que rara vez recuperan el 100% de la misma), se usa el control de versiones en repositorios (también se usa para este fin Docker y Keeper).

Un método básico y fundamental es que antes de modificar un archivo lo copiemos con otro nombre: ***cp /etc/config1 /etc/config.old*** . O en el caso de los archivos de configuración podemos comentar líneas antiguas usando ***#***.

También existen algunas herramientas que controlan */etc* directamente, por ejemplo, ***/etc/keeper*** que está construido sobre un sistema de control de versiones ***git***. Asique mejor aprender a usar *git* ya que es el más usado y de los mejores para sincronizar desarrollos entre un grupo de personas. Las ventajas de aprender *git* es que incrementa nuestras skills *devops*, permite un seguimiento de los cambios y del autor que los ha realizado y también permite planificar un entorno de prueba entre varias personas, por esto, normalmente en *git* nunca se trabaja en la rama *master* o rama de producción ya que es donde están los cambios definitivos, por lo que normalmente los usuarios trabajan en otras ramas hasta tener listo su trabajo y poder incorporarlo a la rama de producción.

* *¿CÓMO FUNCIONA GIT?*

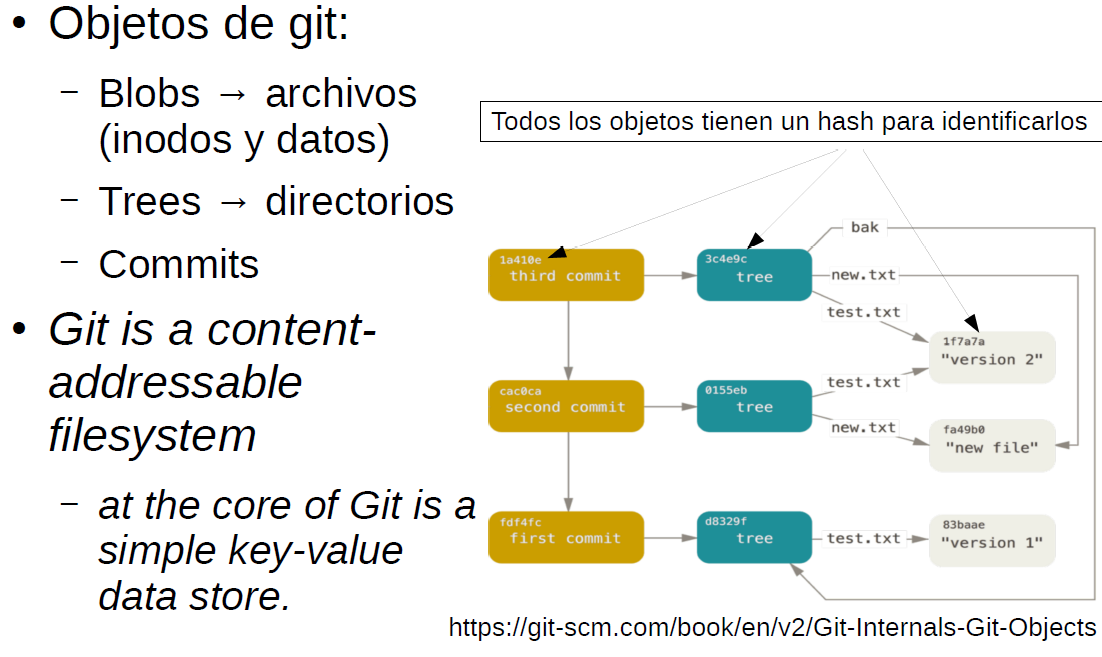
Git se divide en tres directorios o zonas de trabajo:

* + Directorio de trabajo: Directorio local donde encontramos lo que tenemos en desarrollo.
  + Stage: Zona donde registramos los cambios y vemos los archivos que están por sincronizar (podemos seleccionar cuales queremos ignorar).
  + Commit: Zona donde se convierten los cambios en permanentes (podemos hacer comits en diferentes ramas “branches” y ponerles etiquetas “tags”).

La evolución del trabajo puede mostrarse como un grafo dirigido acíclico (DAG):



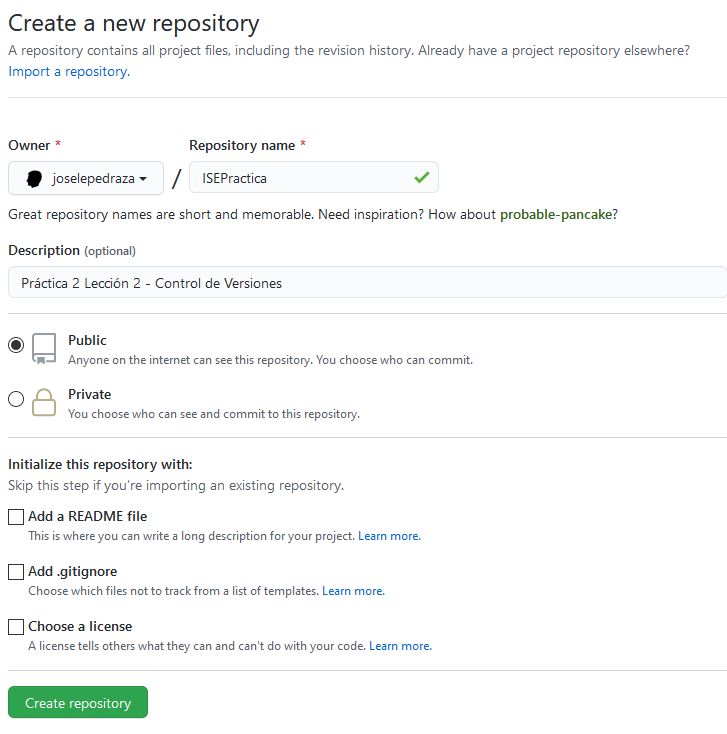
* + Cada vez que se hace un commit se genera un hash (SHA-1 de 40 caracteres) que lo identifica (usando los 7 primeros suele ser suficiente para identificar ese commit).
  + El último commit es el HEAD (la cabeza de la pila).
  + Podemos desplazarnos a partir del identificador (dado que cada commit es una versión diferente del trabajo). Nos permite sacar versiones de la pila hasta llegar a la versión que consideremos correcta con ***~*** y ***^*** y establecerlo así como nuevo HEAD (hay también un log de cómo nos desplazamos: ***git reflog***)
  + Los ***..*** hacen referencia a intervalos (lineales) de commits. Los ***...*** hacen referencia a intervalos no lineales.



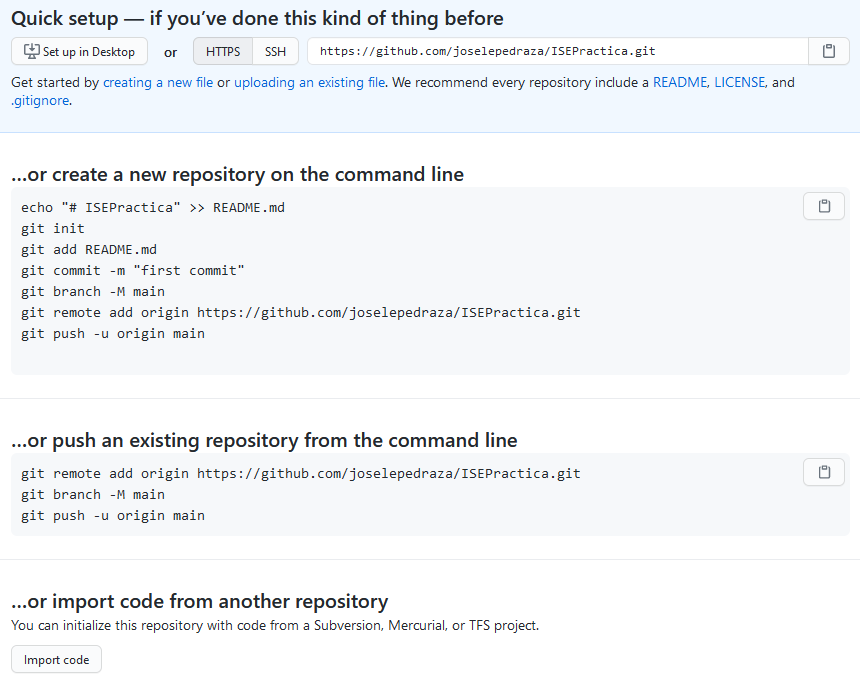
En el gráfico anterior podemos ver como hace *git* los seguimientos internamente. Para cada commit crea un árbol de los ficheros incluidos en el mismo y le asigna un *hash* y un *tag*. Al final internamente en el núcleo de git únicamente se almacenan los datos de forma key-value.

* *INICIANDO UN REPOSITORIO*

Después de instalar *git* en Windows. Nos vamos a un navegador a los repositorios de nuestra cuenta para crear uno nuevo. Debemos recordar que tenemos un número limitado de repositorios privados. Tambíen podemos añadir varios ficheros, *gitignore* para añadir ficheros que no queremos sincronizar, la licencia o el *README file* en formato *markdown*.



Creamos el repositorio y tenemos dos opciones para trabajar con él (dado que todavía está vacío el repositorio), podemos o bajarlo directamente a nuestro pc local o lo vamos haciendo todo paso a paso de forma manual desde el *bash* de *git*.



Nos situamos en el directorio de nuestro pc donde queramos trabajar y escribimos en la terminal: ***git init*** para inicializar el repositorio vacío.



Ya hemos tomado la instantánea de *git log*, *git status* y *git branch.* Debemos tener cuidado ya que todo esto se almacena en $pwd/.git , si borramos el directorio, perderemos el control de cambios.

Por ejemplo, vamos a traernos la carpeta de planificación de las prácticas de la asignatura para tener nuestros primeros archivos en la rama master.



Dentro de Planificación tenemos los dos ficheros de la asignatura que nos hemos traído y un fichero de prueba .txt creado con touch para modificarlo posteriormente y ver el funcionamiento de git.

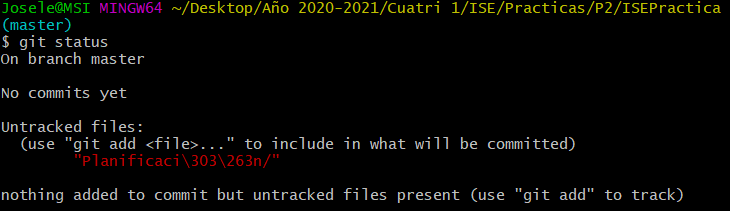
Aún no hemos linkado nuestra carpeta en pc local con nuestro repositorio en GitHub, lo haremos con: git remote add origin [***https://github.com/joselepedraza/ISEPractica.git***](https://github.com/joselepedraza/ISEPractica.git) .

Una vez lo hemos iniciado, pasado el contenido y linkado (sincronizado), tenemos que subirlo al repositorio para iniciar el control de cambios sobre este reposoitorio.

Antes debemos modificar el fichero de configuración de git en la raiz de nuestro sistema con: ***vi ~/.gitconfig*** para enlazar nuestro correo con este git y añadimos:

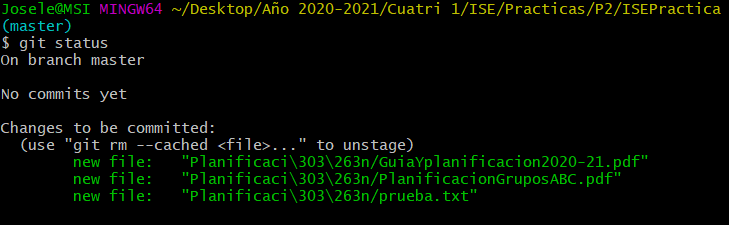


Para ver en qué estado de seguimiento están nuestros archivos, usamos ***git status***

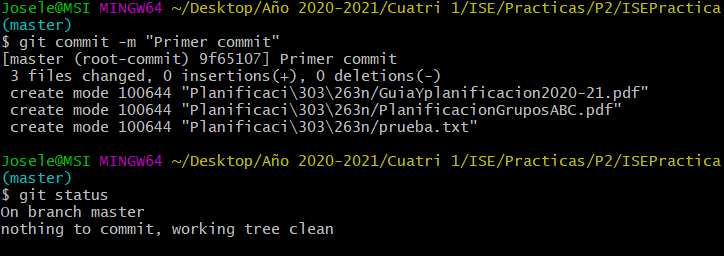


Nos dice que los ficheros no tienen seguimiento ya que debemos decirle ahí que está ahí para que esté pendiente de cualquier cambio en la carpeta Planificación (no está aun metida dentro del control de cambios). Para incluir lo que se va a commitar de los ficheros del repositorio lo hacemos con: ***git add Planificación/*** (para activar el seguimiento automático de los ficheros incluidos y la carpeta)

Y comprobamos con ***git status*** . Y vemos que ya reconoce que hay nuevos ficheros en el sistema de seguimiento.

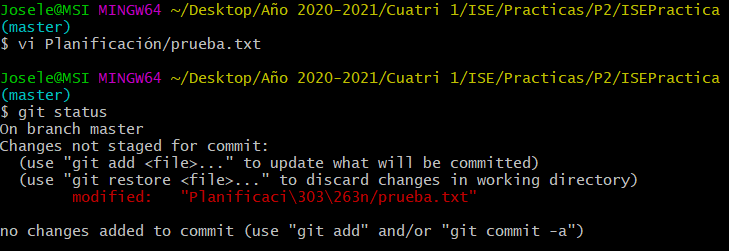


Ahora debemos hacer el primer ***commit***, es decir, empaquetarlo para subirlos al repositorio (a la parte que se sincronizará en el servidor). Lo haremos con: ***git commit -m “Primer commit”***



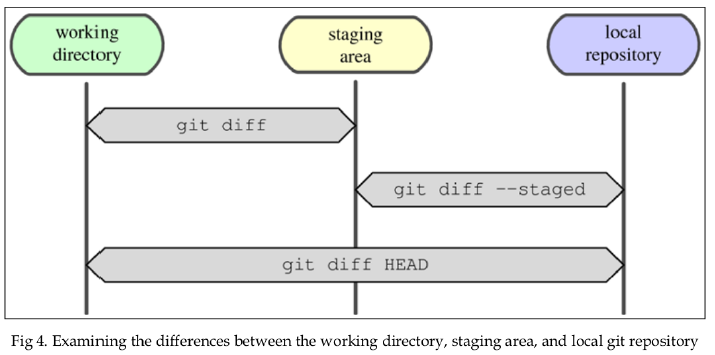
Comprobamos con ***git status*** y vemos que no hay nada, ya que todos los ficheros que se habían cambiado (los tres ficheros nuevos) ya están incluidos en un commit, por lo tanto el directorio local está actualizado con la zona de commit.

Una vez hecho el commit, si modificamos un fichero (***vi prueba.txt***) y hacemos otro ***git status*** veremos como el sistema de control de versiones nos dice que ese fichero ya no coincide con el primer commit, es decir, que ha sido modificado.

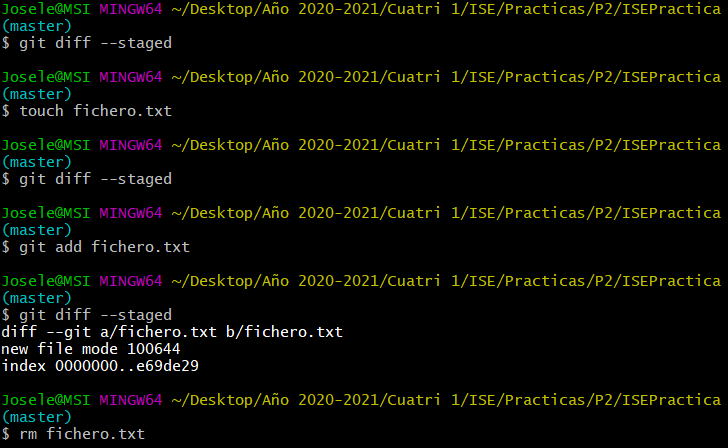


Antes de hacer el segundo commit, podemos ver los cambios que se están haciendo en el repositorio con ***git diff*** :

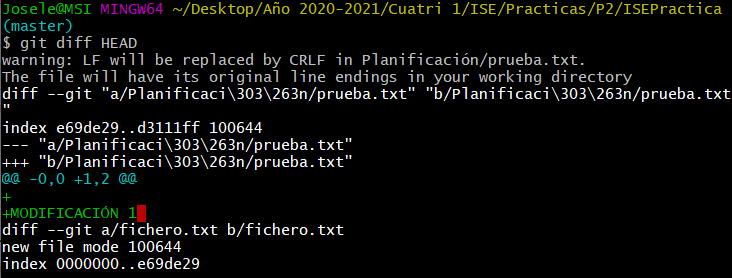
* + Con ***diff***, podemos ver qué diferencias hay entre lo que se ha modificado en el directorio de trabajo y lo que está siendo seguido en el stage
    - Si modificas y luego add --> diff==0 (diagrama)
    - Si add y luego modificas --> diff muestra
  + Con ***diff --staged*** las diferencias entre el stage y el repositorio
  + Con ***diff HEAD*** las diferencias entre el directorio de trabajo y el repositorio.



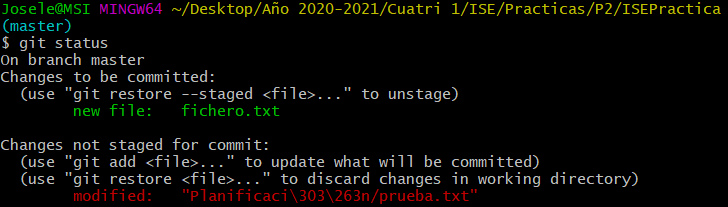
Si probamos a hacer ***git diff --staged*** vemos que no aparece nada, ya que no ha habido cambios entre el working directory (carpeta en pclocal) y el stagin area (la que le hace el seguimiento a los archivos), están sincronizados. Si creamos un fichero nuevo y lo añadimos con ***git add*** pero no hacemos ningún commit y volvemos a comprobar con ***git diff --staged*** vemos que ahora si encuentra que se ha creado un fichero nuevo y por tanto no están sincronizadas esas áreas.



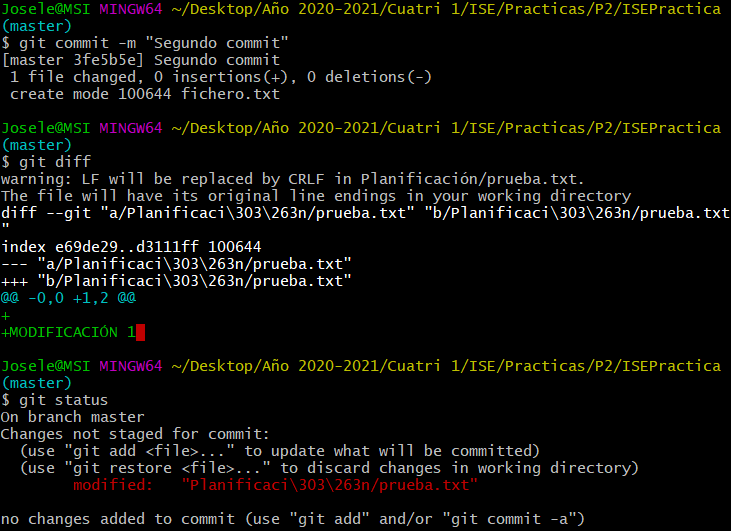
Si ahora vemos lo que difiere de nuestra carpeta a nuestro repositorio (la parte de commit) con ***git diff HEAD*** . Nos da los cambios de todo, vemos que nos indica por una parte que en nuestra carpeta personal está la primera versión y que en nuestro repositorio local hemos añadido información, y luego que también está la segunda versión que es (Modificación 1) y además nos dice, que el fichero.txt que acabamos de añadir, no existe en la carpeta de trabajo pero si en el repositorio local. Esto sobretodo es útil cuando hay fallos.



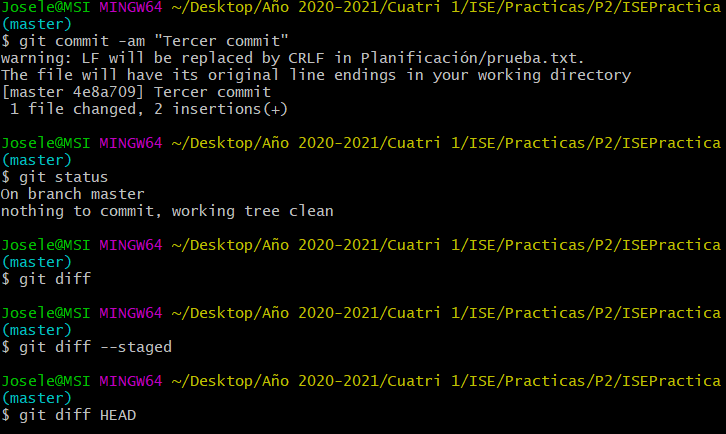
Si hacemos un ***git status*** vemos que efectivamente estamos en el punto de que hemos cambiado el fichero *prueba.txt* y hemos añadido un nuevo *fichero.txt*.



Ahora debemos empaquetar estos dos nuevos cambios en el segundo commit para hacer estos cambios permanentes, con ***git commit -m “Segundo commit”*** . Pero no debemos olvidar que a los fichero modificados se les tiene que añadir los cambios previamente, esto lo podemos hacer con: ***git add*** (si son ficheros nuevos) o ***git commit -am*** (si ya lo estamos siguiendo y lo unico que estoy haciendo es modificar, es decir, si no estamos metiendo ficheros nuevos, así nos podemos ahorrar el add ya que lo engloba).



Hacemos el tercer commit correctamente con ***git commit -am “Tercer commit”*** y comprobamos que no hay ningún cambio entre las diferentes zonas de trabajo.



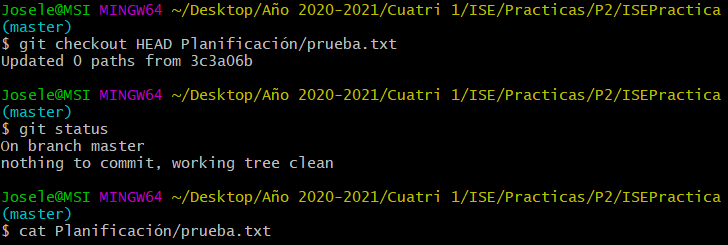
* *RECUPERANDO CONTENIDO*
  + Antes de un commit:
    - ***git checkout HEAD nombre\_archivo*** (devolverá el nombre de archivo a la última versión del commit
    - ***HEAD*** es un puntero al último commit hecho, aunque puede retroceder con ***git reset***
  + Despúes de un commit:
    - Podemos recuperar un commit concreto usando:
      * ***git reset 7\_primeros\_caracteres\_del\_commit\_SHA*** (cuidado, que esto actualizará el HEAD)
  + También hay otras formas de trabajar (worksflows), por ejemplo, implementar una funcionalidad nueva usando una rama nueva (branch) y luego uniendo con otra rama o la master (*merge)*.
    - Hay otras posibilidades como ***git rebase*** o ***git cherry-pick* n** (cuidado con reescribir el histórico)

Por ejemplo, vemo con ***git log*** los commits que hemos hecho y podemos ver el identificador (SHA), el autor, fecha y la etiqueta (tag). Y también vemos cuál es la cabeza de la pila (HEAD). Si ahora quisiesemos volver al segundo commit. Copiamos el hash del segundo commit y se lo pasamos a ***git reset*** y volvemos a comprobar:



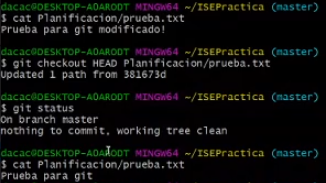
Ahora si hacemos un ***git status*** vemos que nos vuelve a salir que el fichero *prueba.txt* está modificado (si nos hubiese dejado hacerlo sin el ***--hard***, dado que ya el fichero esta vacío como antes de modificarlo y añadirlo con ***add)***, ya que el commit que hicimos para que el fichero dejase de estar modificado lo hemos eliminado.

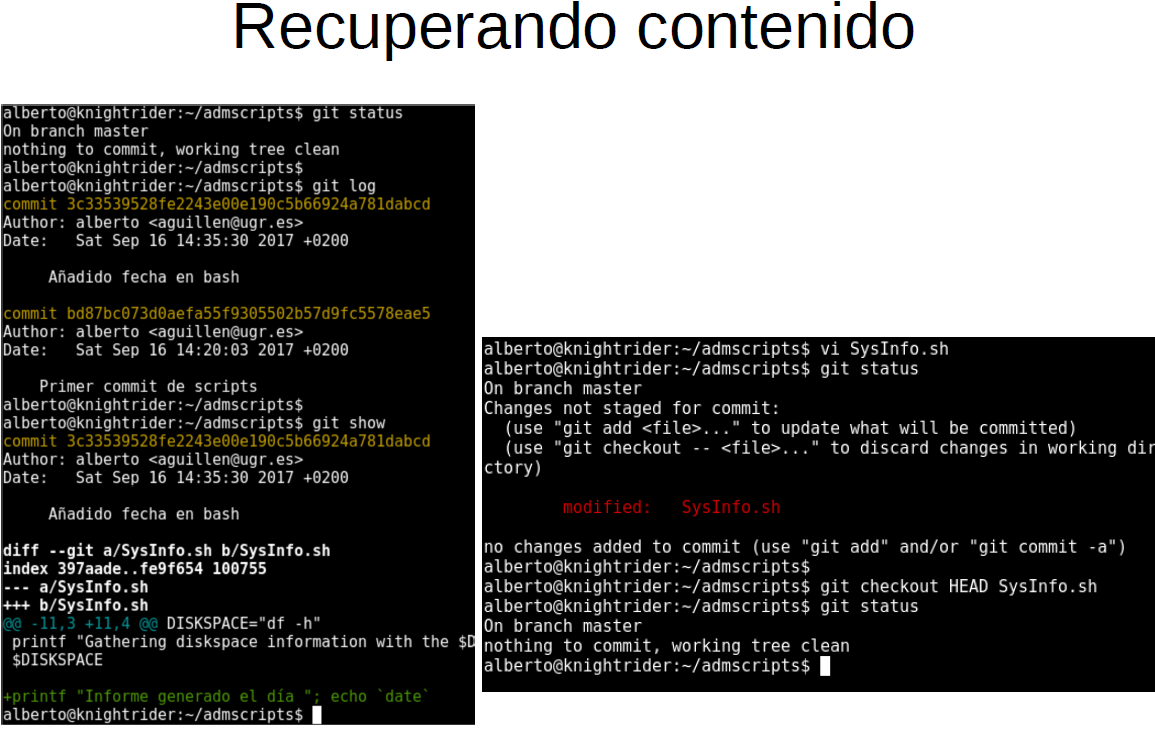
Ahora podemos con ***git checkout HEAD Planificacion/prueba.txt*** actualizarlo en el commit actual para en el caso de salir modificado como hemos indicado anteriormente, salga actualizado.

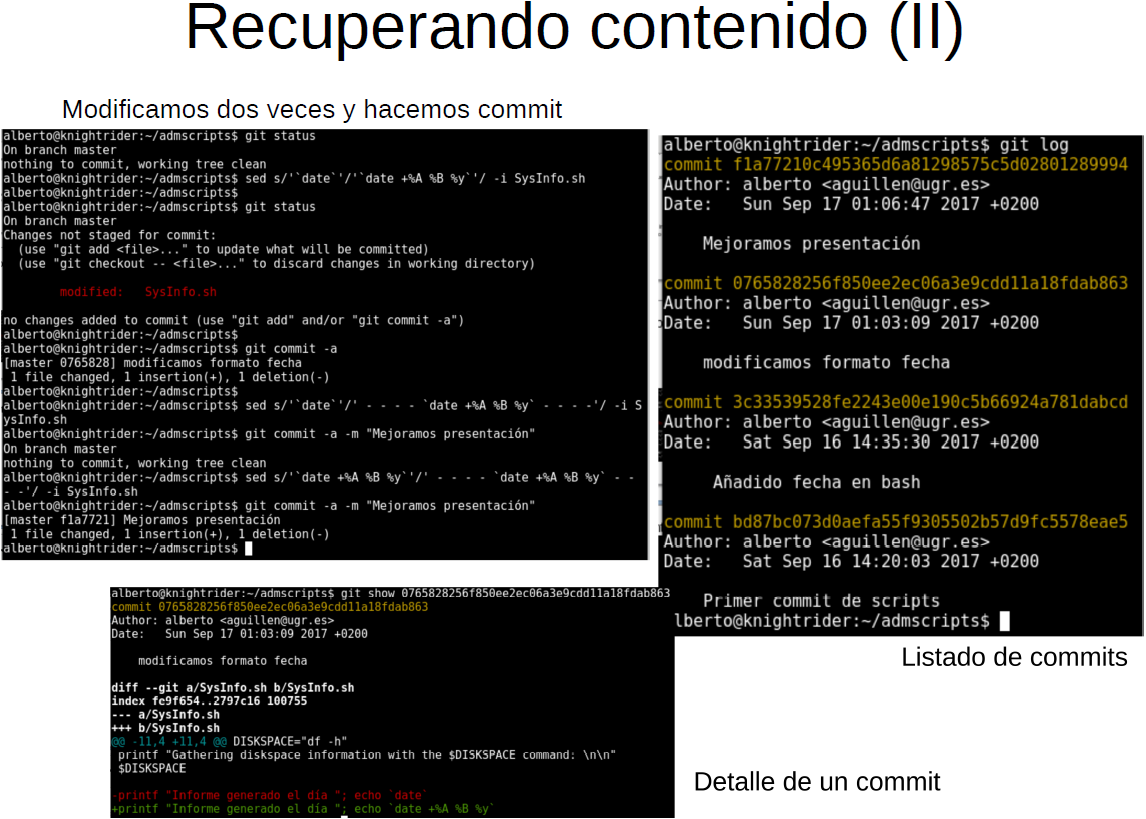


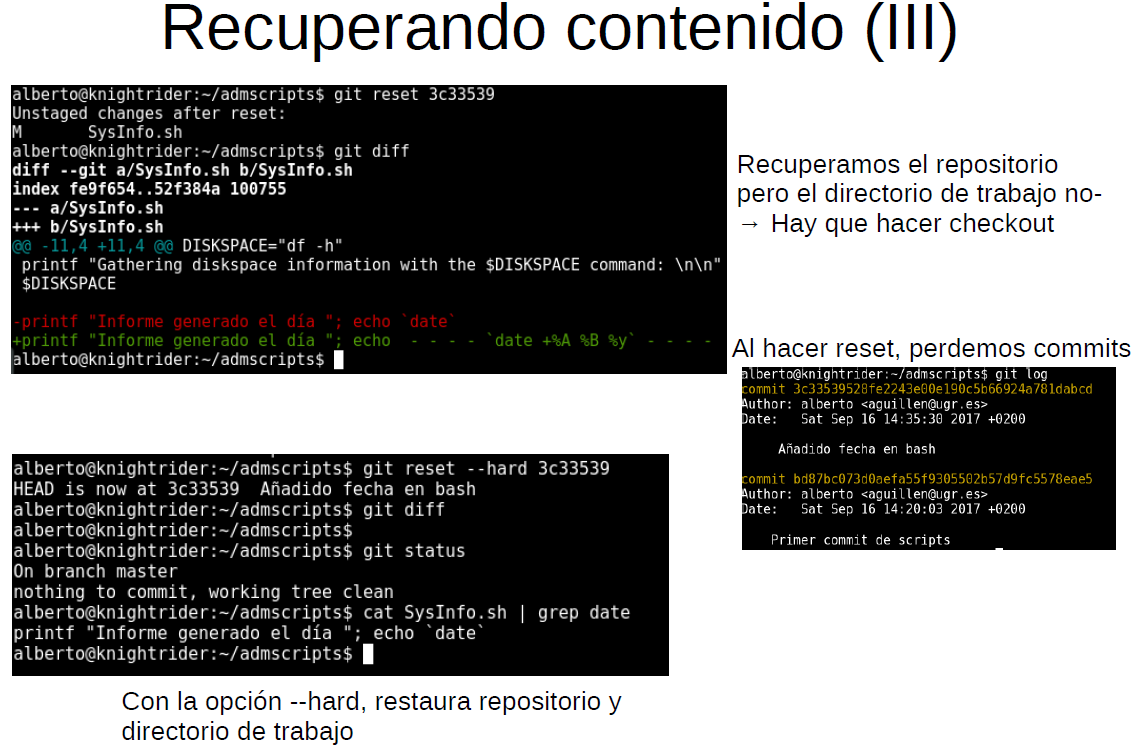
* NOTAS:

Con ***git reset***:

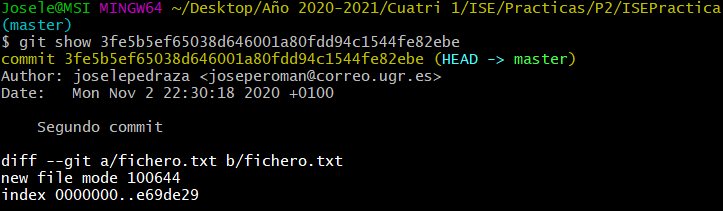






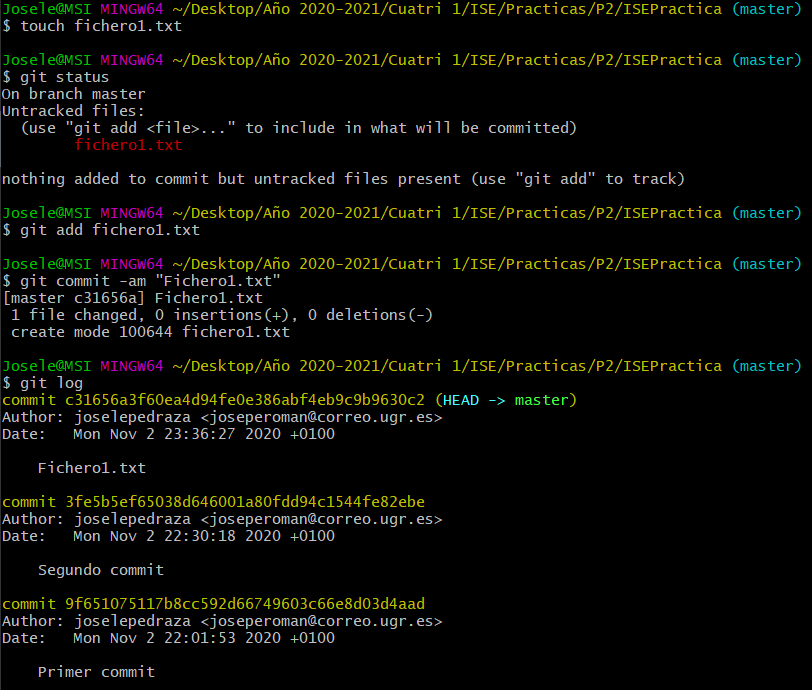


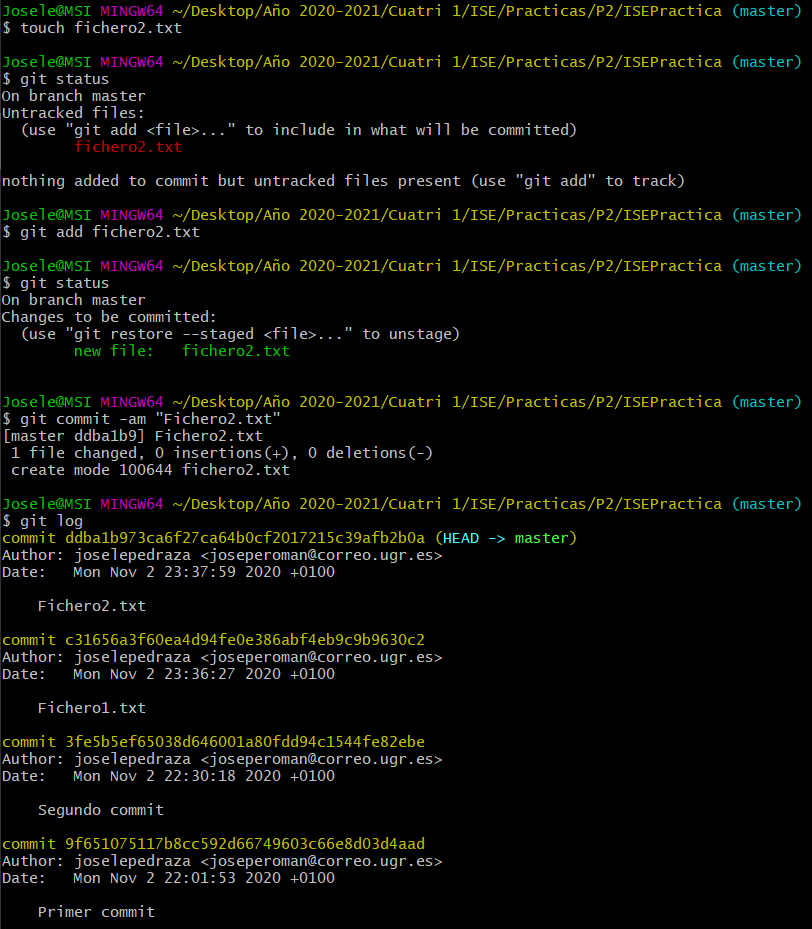
Con ***git show*** podemos ver información sobre los commits pasándole el SHA:



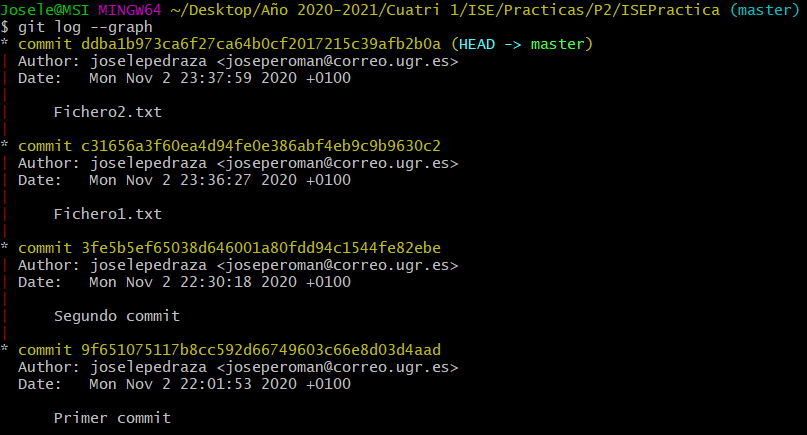
Como hemos visto, con ***git reset*** se desecha último commit y no hay forma de recuperarlo, como alternativa a perder todo el trabajo que hemos hecho, tenemos ***git revert <SHA>*** . Lo que hace es copiar el commit que le digamos y lo mete en la pila siendo ahora el HEAD, y así podemos mantener tambien las versiones anteriores.

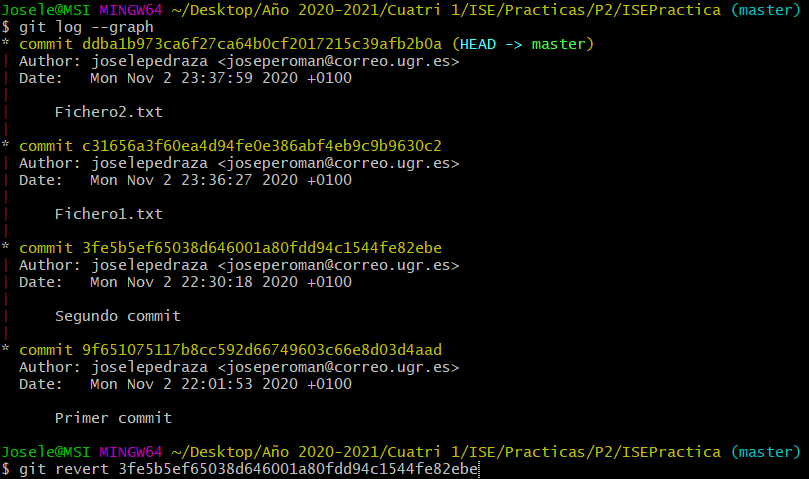
Vamos a ver su funcionamiento:

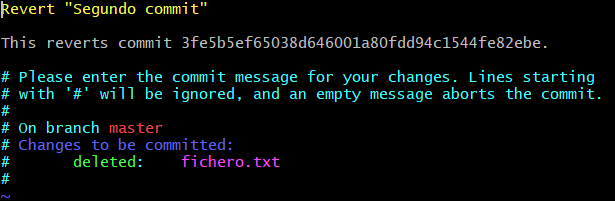


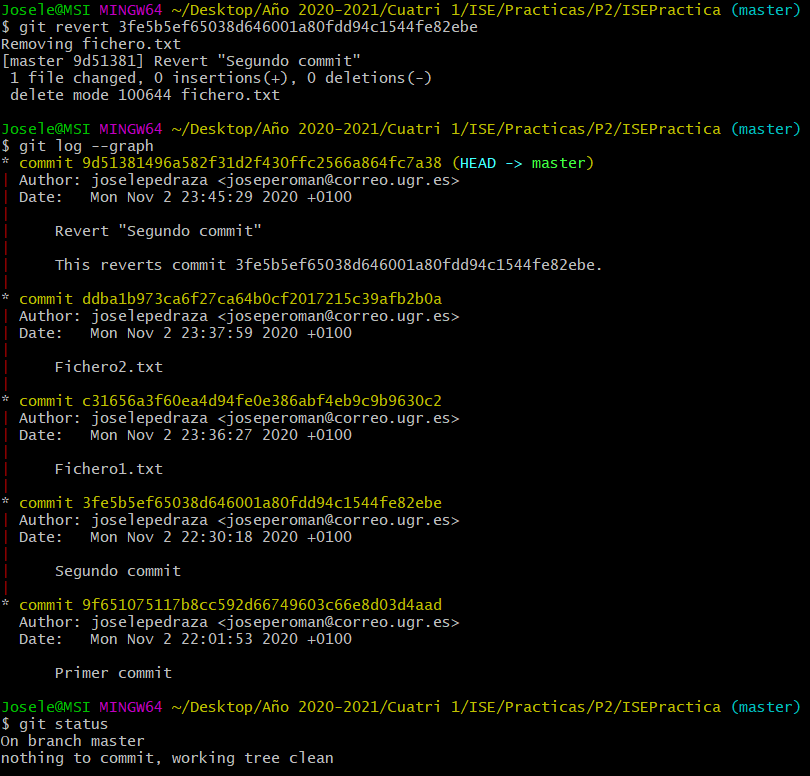


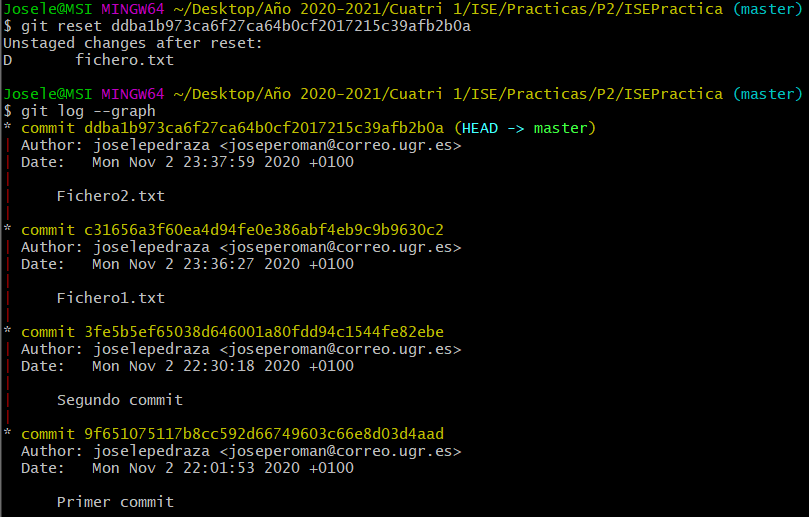
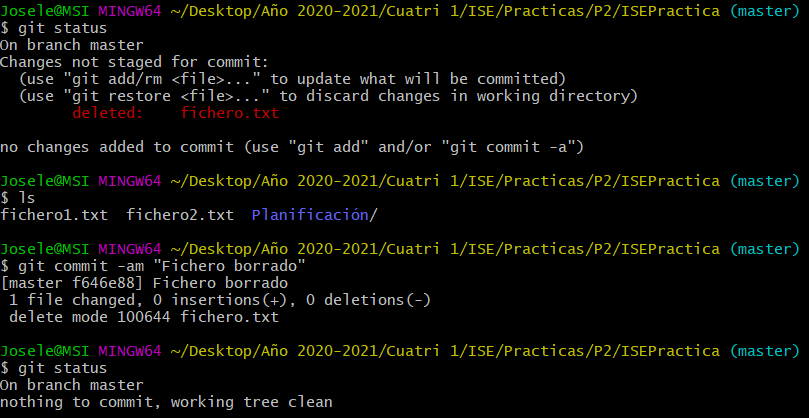
Con ***git log --graph*** nos une los commits con flechitas como en una pila :)



Una vez creados los dos ficheros y hecho un commit para cada uno, nos damos cuenta que por lo que sea no queremos esos ficheros demomento en nuestro sistema, pero quizás los usemos más adelante, por lo que no queremos perderlos. Entonces hacemos un ***git revert <sha>*** al segundo commit para reestablecer el sistema sin esos dos ficheros (pero no desecharlos permanentemente).

Nos aparece un feedback de lo que se está haciendo (cerramos vi con ***:q***)

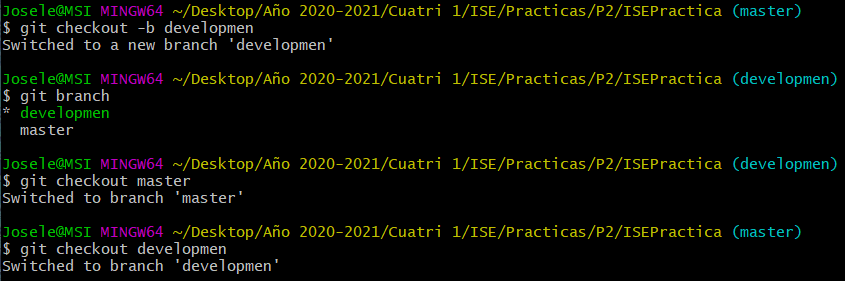
Comprobamos con *git log* y *git status*.

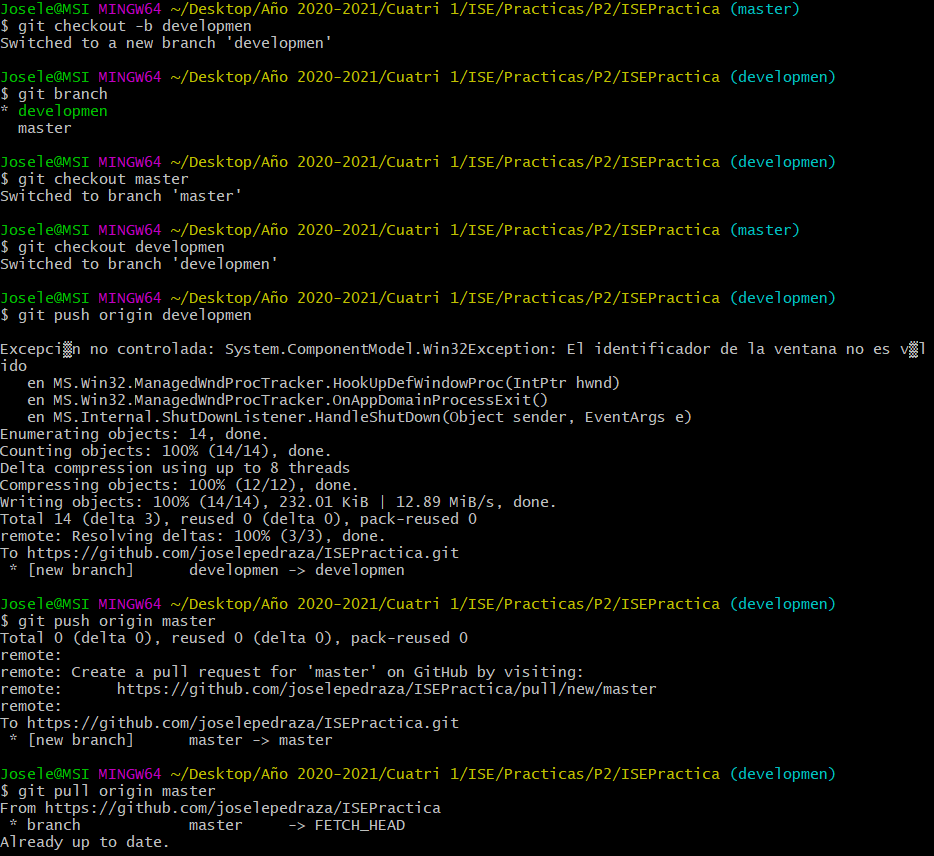
Si ahora quisiesemos volver a la versión con los dos ficheros hacemos el ***git reset <sha>*** con el identificador del commit del “Fichero2.txt”.

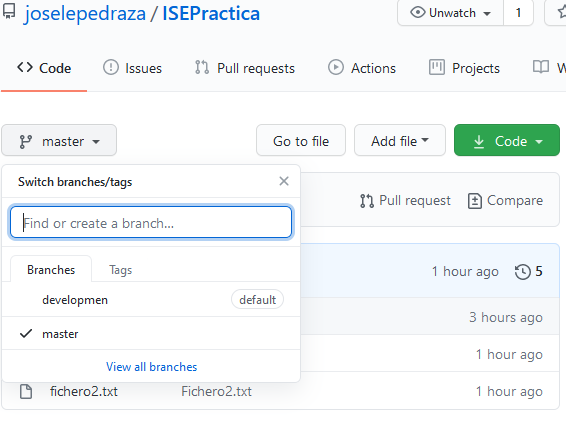
* RAMAS:

Se suele trabajar con varias ramas porque lo normal es que un proyecto esté divido en diferentes módulos y varios equipos de trabajo. Normalmente hay una persona encargada o supervisor final que va cogiendo los cambios de cada rama y los va pasando a la master o rama de producción para que no haya problemas de incompatibilidades (por ejemplo, rama para el modulo de login, rama para el modulo de buscar, esta rama para el de comprar, etc). Cuando está todo hecho, se suele hacer un ***pull request*** que es una petición para enviar tus cambios a la rama master, si el supervisor no lo valida, lo arreglas y solicitas otro.

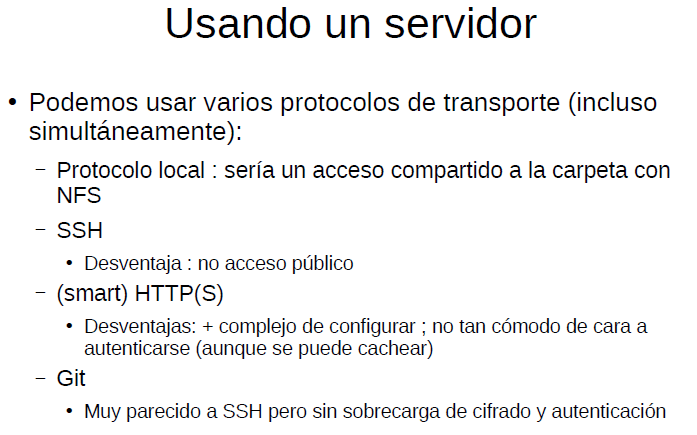
* + Podemos bifurcar el grafo usando ramas:
    - ***git branch*** : Nos muestra en qué rama estamos (master por defecto)
  + Para cambiar de rama usamos:
    - ***git checkout <nombre\_rama>***
  + Para crear una rama nueva
    - ***git branch <nombre\_rama>***
    - ***git checkout -b <nombre\_rama>*** (en este caso, crea la rama nueva y se cambia a ella)
  + ***git stash***
    - Permite salvar el directorio actual sin tener que hacer un commit (así podremos cambiar de rama)
    - Podemos dejar trabajo “a medias” sin hacer commit
    - ***git stash apply*** --> aplica el último stash creado
    - ***git stash list*** --> muestra los disponibles
    - ***git stash apply <stash>*** --> aplica el determinado
  + Normalmente en modo supervisor:
    - ***git merge/git rebase*** --> Para pasar toda la información de una rama a otra.

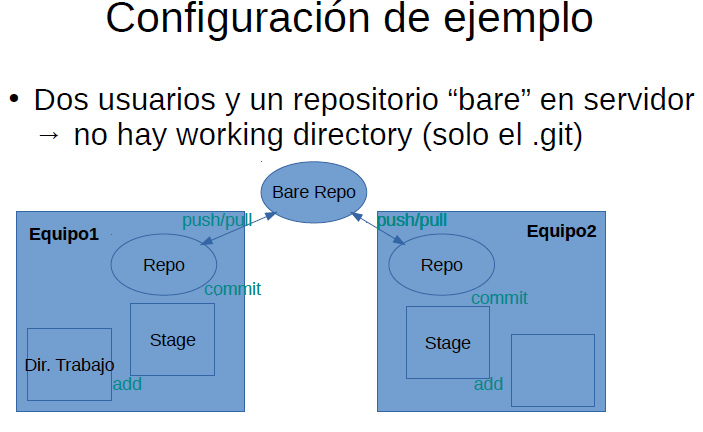
Para crear una rama nueva usamos ***git checkout -b developmen*** para el modulo de desarrollo (y se cambia a esta rama, ya no estaríamos en la master, por lo que cualquier cambio no le afectaría). Si hacemos ***git branch*** podemos ver las ramas del repositorio.

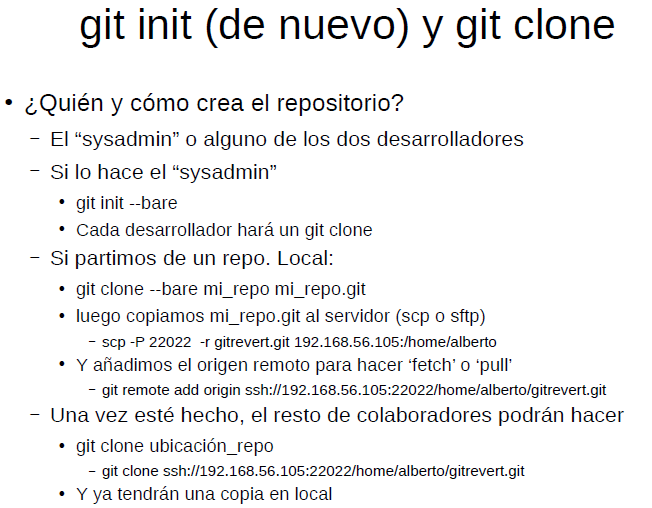
Ahora comprobamos que en nuestro perfil de GitHub no aparece todo esto que hemos hecho todavia dado que hemos hecho todos los cambios “en local” y todos estos cambios están “metidos” en commits en local, preparados en la zona de commits en el working directory, pero aún no están subidos al servidor. Para subir cambios a una rama usamos ***git push*** y para bajarlos usamos ***git pull*.[[2]](#footnote-2)**

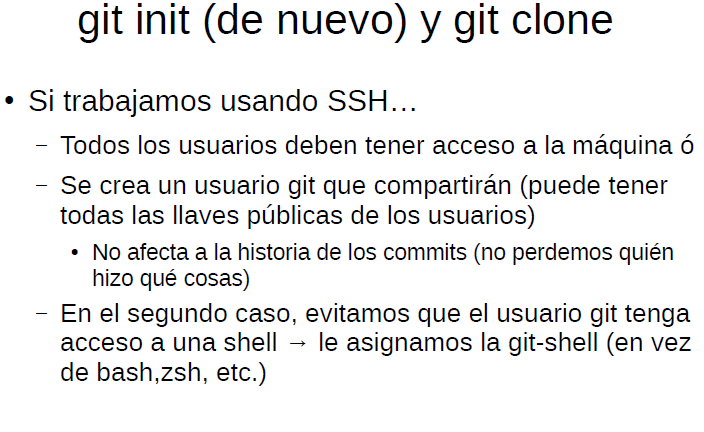


Si las ramas quedasen desincronizadas podemos hacer un ***pull request***. (no es el caso)

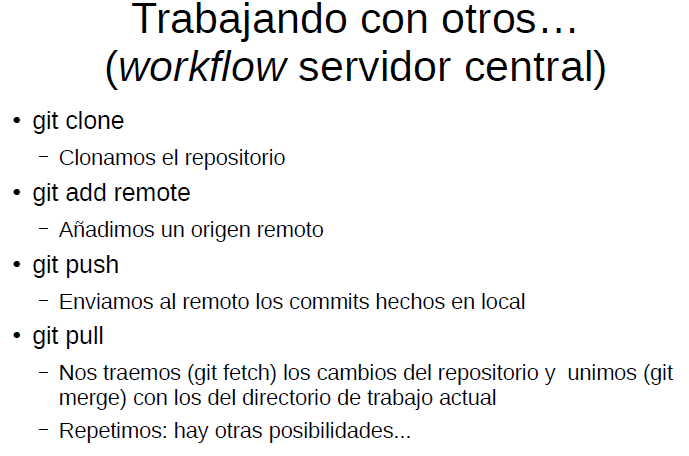




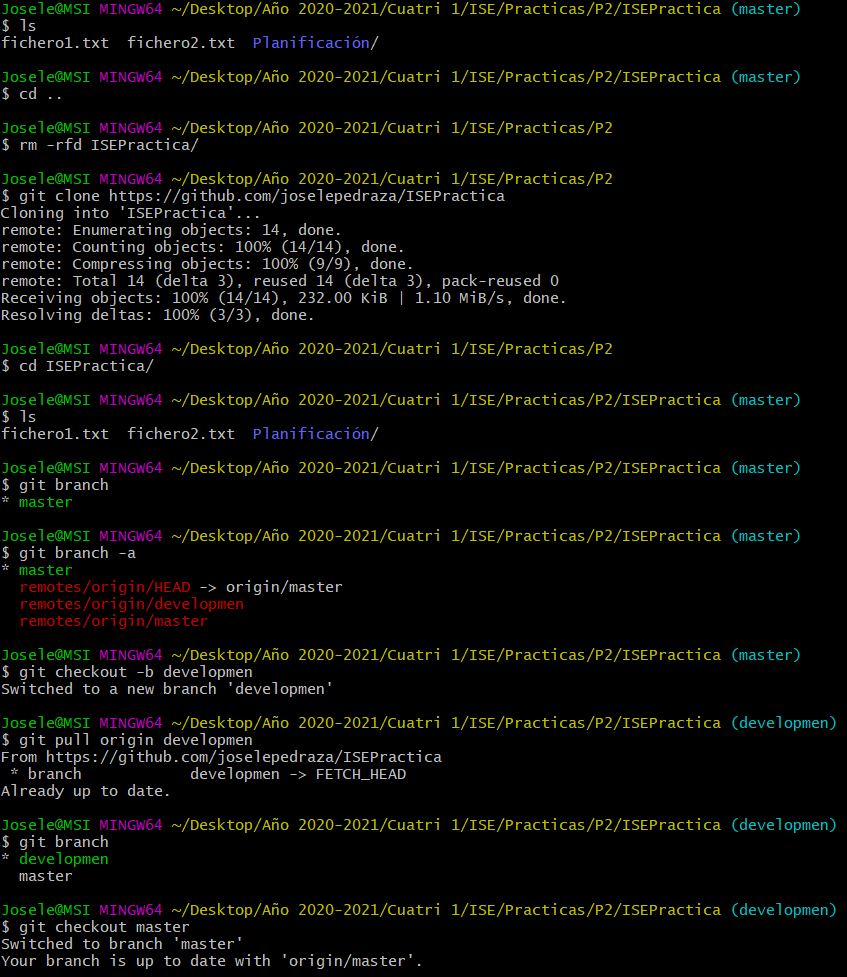




git init es lo que hemos hecho nosotros, con git clone lo coges y lo clonas y directamente se queda actualizado (en el git clone no hace falta que un usuario entre y haga el init, lo hace el git clone implicitamente).



Vamos a probar git clone. En este punto, voy a borrar todo lo que tengo en la carpeta local con ***rm -rfd ISEPractica/*** , esto en el servidor no le afecta a nada. Para restaurar todo otra vez (lo que acabo de borrar en el pc local) cojo la URL del git y le hacemo el ***git clone <URL>***.

Solo te clona la rama por defecto, si hacemos un ***git branch -a*** vemos que también tenemos la developmen pero está desincronizada, entonces le haremos un ***git pull***.

**LECCIÓN3:**

**5) Instalando un servidor web básico.**

**5.1) Webmin**



Trabajaremos con Ubuntu y CentOS, donde instalaremos diferentes servicios. CentOS suele dar más problemas por el módulo de seguridad de SELinux. También veremos failtoban, herramienta que permite banear durante un tiempo ips que hacen más de x intentos de acceso (peticiones) por x puerto. También veremos la herramienta screen que sirve para dejar en segundo plano sesiones de SSH (podemos apagar el ordenador y después retomar por el mismo punto sin problemas).

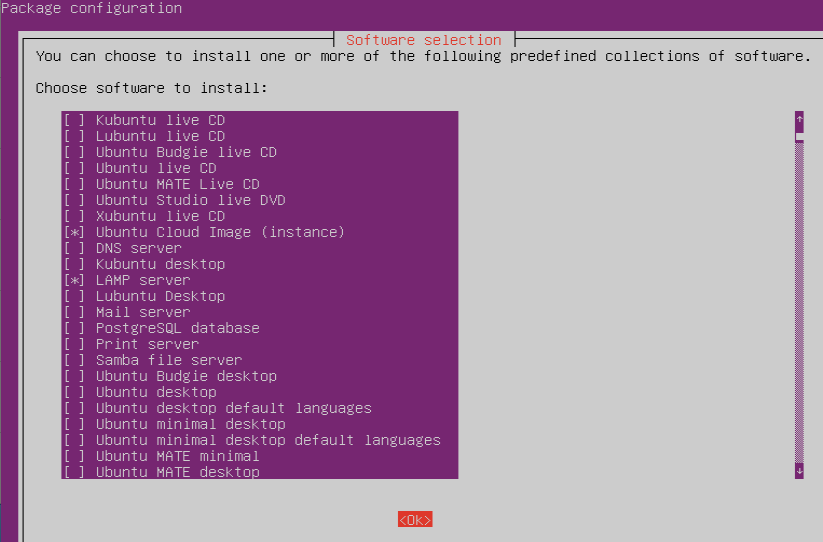
Un servidor web es una máquina a la que se realizan peticiones y te sirve información. Proporciona un servicio.

Desde hace unos años se tiene una arquitectura básica de tecnologías para desarrollar servidores web, arquitectura LAMP (Linux, Apache, MySQL/MariaDB Php/Python). Apache es un servidor web, una forma de montarte la parte de escucha (sockets), el tema de los puertos, es decir, el servidor como tal (a parte de las bases de datos o SOs), en Linux apunta a /var. Otra herramienta más actual y bastante potente y profesional, es Ngynx.

Configuración de LAMP en Ubuntu Server

Configuraremos un servidor básico en Ubuntu, configurando LAMP, para que nos instale todo lo que nos hace falta para montarlo. Iremos a un navegador en nuestro pc o host (también podríamos hacer un *curl*) y comprobamos que podemos abrir la página principal que nos servirá la máquina Ubuntu.

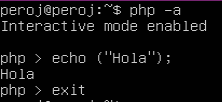
Arrancamos la máquina y empezamos. Necesitamos AMP (Apache, MySQL y Php) ya que Linux es donde estamos trabajado. Podemos instalarlo a mano usando cualquier gestor de repositorios (apt, snap...) pero en este caso y para hacerlo a la vez usaremos la herramienta *tasksel* que descargamos en la lección1 junto con *ssh*. Con ***sudo tasksel*** abrimos la interfaz gráfica para marcar herramientas que queremos instalar. Marcamos LAMP server y tendremos listo nuestro servidor web en un minuto aproximadamente.



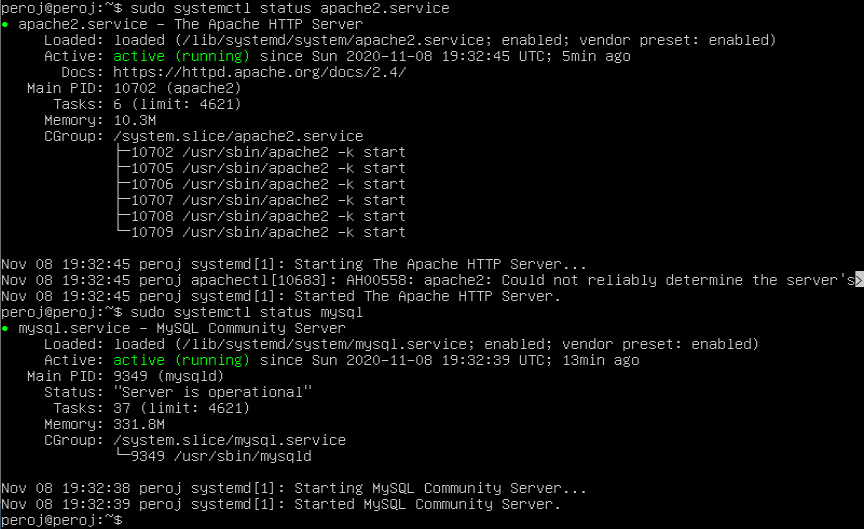
Vemos que nos da un error de *apt-get* y podemos intuir que no encuentra alguna dependencia o *taksel* tiene algún repositorio o dependencia desactualizados. Entonces actualizamos toda la lista de dependencias de Ubuntu con ***sudo apt-get update***. Cuando acabe, repetimos el proceso y comprobamos que se nos instala LAMP correctamente con *tasksel* y vemos que ya pasa del fichero 34 (donde faltaba la dependencia) e instala todo correctamente.

Comprobamos que funciona lo que acabamos de instalar:

* Php --> lo comprobamos con ***php -a*** que nos abre el modo interactivo.

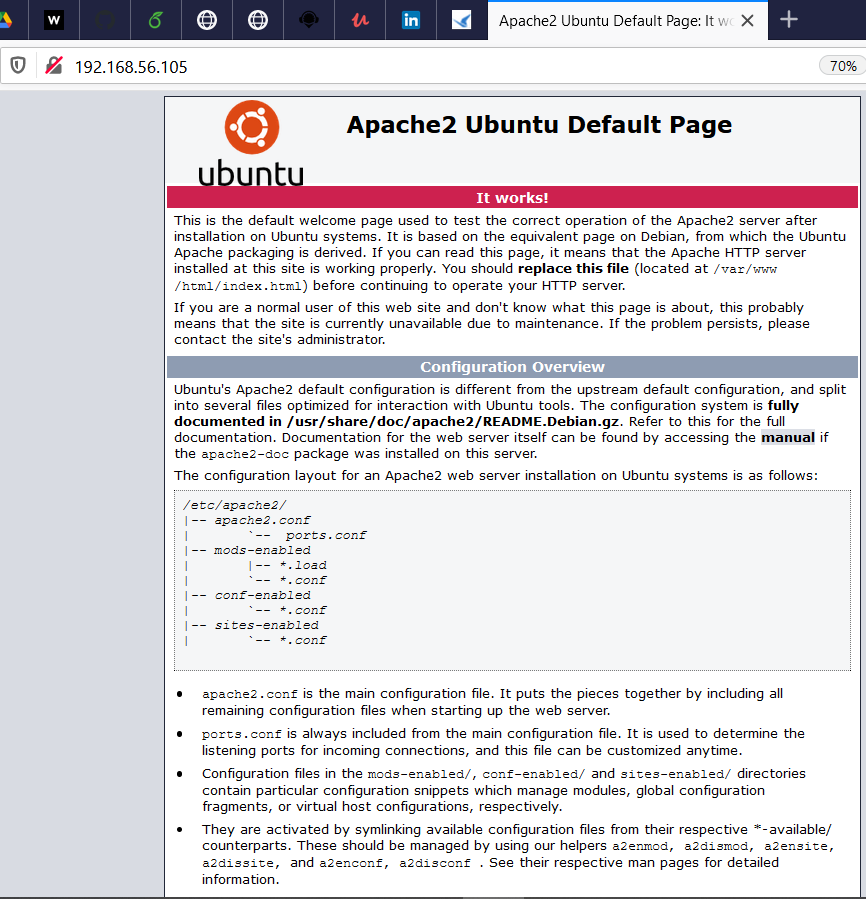


* Apache2 --> Para comprobar el estado del servicio, como siempre, usamos ***sudo systemctl status apache2*** (o ***apache2.service***). Y vemos que está *running* (servidor de http de apache está running por el puerto por defecto del protocolo HTTP[[3]](#footnote-3) :80 y HTTPS por el :443).



* MySQL --> Para comprobar el estado del servidor de base de datos MySQL Server usamos ***sudo systemctl status mysql***. El puerto por defecto es el :3306. Entonces aqui tenemos que diferenciar entre MySQL (base de datos a nivel local) y MySQL Server (servidor de base de datos, para acceder a la información de manera remota abriendo un socket por el puerto 3306, sirviendo las respuestas a las peticiones a las bases de datos.

Ahora para comprobarlo, nos vamos al navegador de nuestro ordenador local o host (o desde *cmd* de Windows haciendo un ***curl***a la IP de nuestra máquina Ubuntu) e introducimos en la URL la dirección IP: 192.168.56.105.



Vemos que nuestro servidor está sirviendo la página por defecto del servidor Apache2 instalado en nuestro servidor Ubuntu.

En principio todo bien, pero debemos tener en cuenta las limitaciones de montar un servidor de esta manera tan rápida, dado que se puede dar el caso de que por ejemplo como a priori no hemos llevado un control de las versiones que se han ido instalando de Apache2, MySQL... entonces por lo que sea estas versiones puede ser que nos de algún tipo de incompatibilidad o error con otras aplicaciones del sistema. Este es el inconveniente de *tasksel*, debemos también saber desenvolvernos buscando estas versiones manualmente con *snap* o *apt*.

Todo esto es mucho más lento en CentOS por el sistema de seguridad SELinux.

Diferencia entre página web estática y dinámica:

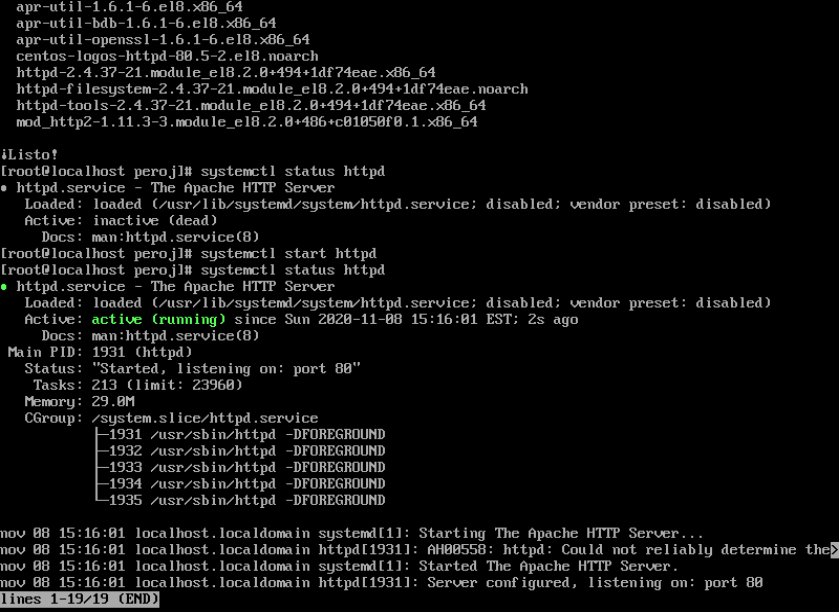
Una página web estática no tiene interacción ninguna con el usuario (no cambia) y no puede cambiar el contenido de forma dinámica, las páginas estáticas a día de hoy no se usan más que para páginas de presentación personal o formularios. Las aplicaciones web que usamos a día de hoy están formadas por páginas dinámicas, donde la página realmente adapta el contenido a la interacción con el usuario sin necesidad de recargar manualmente, siguiendo en la misma URL, en la misma página.

Configuración de LAMP en Ubuntu Server

Abrimos la máquina con CentOS donde ya configuramos *ssh* anteriormente y configuraremos el servidor web.

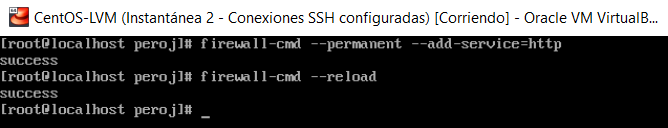
Después de configurar el servidor Apache2 comprobaremos con un simple script en *php* con un *if* si tenemos conexión con la base de datos, siendo esto una página web dinámica muy sencilla, dado que responde a un evento.

Hacemos ***sudo su*** ya que en CentOS si deberemos de tocar más cositas. Para instalar el servidor de Apache en CentOS debemos usar ***yum install httpd*** que es el nombre que tiene en esta distribución (recordamos que la letra ***d*** en CentOS indica “daemon” que es una herramienta de la parte del servidor ya que se queda en background). Comprobamos que está activo con ***systemctl status httpd,*** pero vemos que lo reconoce, pero no está activo asique para activarlo hacemos un ***systemctl start httpd*** y volvemos a comprobar.

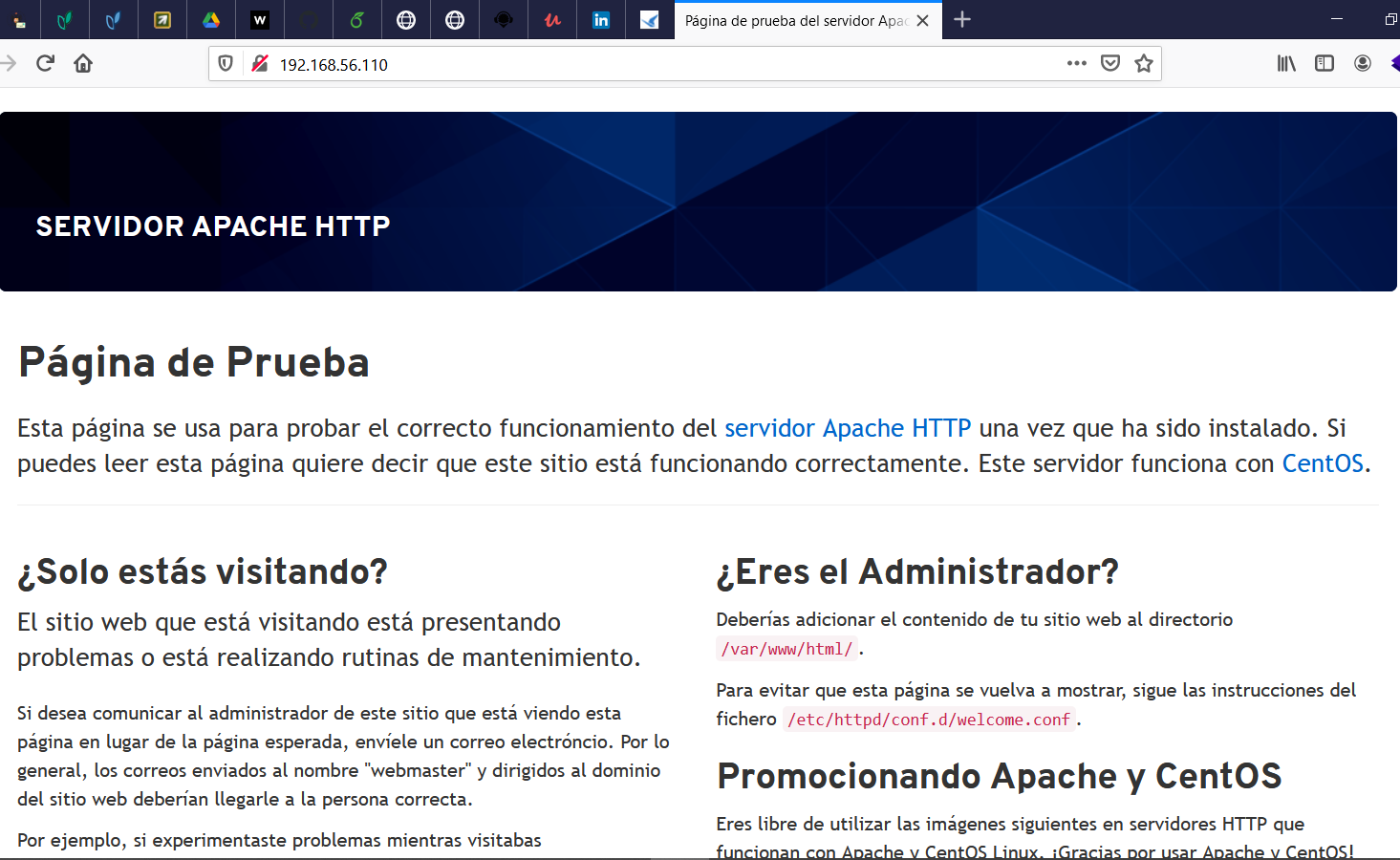


Ahora hacemos como en Ubuntu, nos vamos al navegador del host y en la URL ponemos la IP del servidor de CentOS: 192.168.56.110. Vemos que no consigue encontrar, se queda buscando. Esto es debido al Firewall de CentOS, dado que es muy seguro.

Entonces configuramos el cortafuegos de Centos con ***firewall-cmd*** , la opción ***--permanent*** para dejar el puerto abierto siempre pero después debemos hacer un ***firewall-cmd --reload*** para que lo coja sin tener que reiniciar, la opción ***--add-service=http*** para añadir el servicio que el firewall sabe que va por el puerto :80 dado que es el puerto por defecto y un servicio estándar.



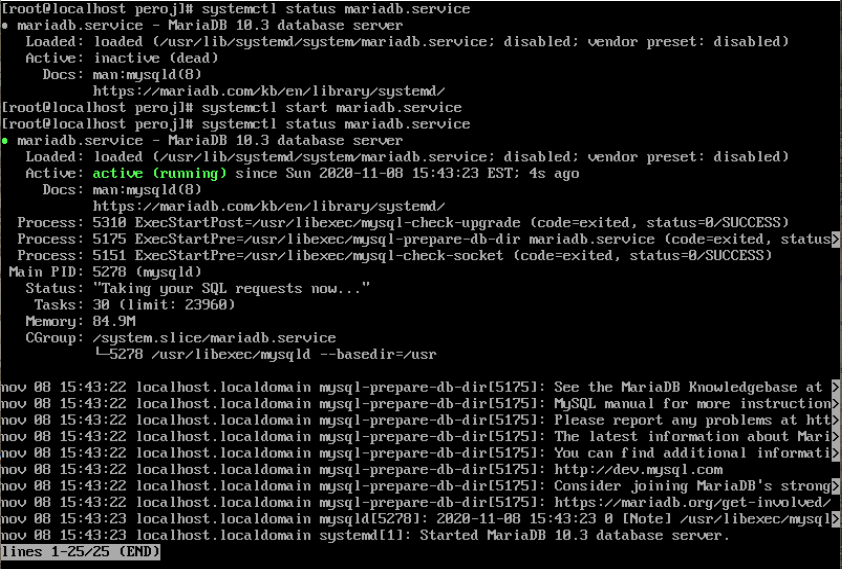
Volvemos a comprobar en el navegador de la máquina anfitrión y ya vemos la página de inicio (estática) del servidor web Apache para CentOS (vemos que no es la misma que nos mostraba en Ubuntu, dado que no es exactamente la misma versión).



Si no queremos que esta página se vuelva a mostrar (si no hay contenido en ***/var/www/html/***, que es donde se incluyen las aplicaciones web que estemos desarrollando, el resto de servicios no tienen por qué ir ahí) debemos modificar el fichero ***/etc/httpd/conf.d/welcome.conf*** .

Ahora mismo del LAMP[[4]](#footnote-4) ya tenemos LA (Linux y Apache) entonces nos faltan MP, el gestor de base de datos de MySQL que en el caso de CentOS será MaríaDB) y php o python.

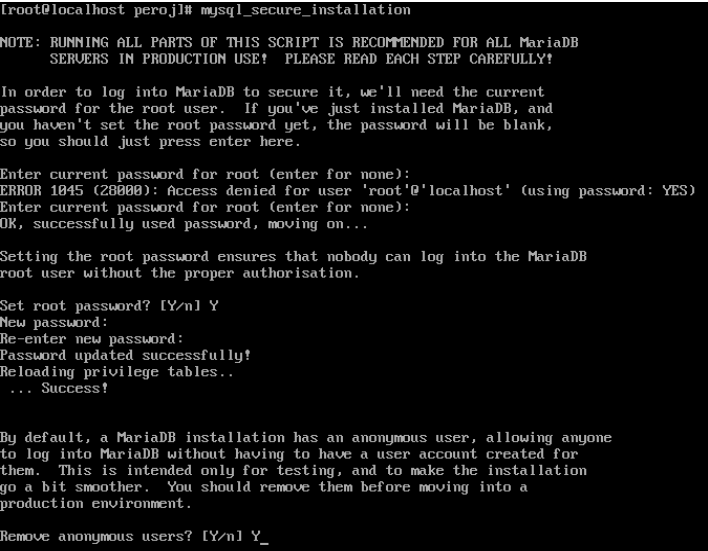
Vamos a instalar tanto MariaDB y MariaDB Server con ***yum install mariadb*** y ***yum install mariadb-server***. Comprobamos si está activo el servicio lo hacemos con ***systemctl status mariadb.service***. Vemos que no está activo por lo tanto lo activamos con ***systemctl start mariadb.service*** y volvemos a comprobar.



Ahora bien, lo normal es hacer una configuración del sistema para evitar ciertas vulnerabilidades que puede ocasionar el servidor de base de datos, dado que si dejamos la configuración por defecto podemos tener problemas ya que la conoce todo el mundo. Es recomendable también cambiar el puerto por defecto de este tipo de servicios.

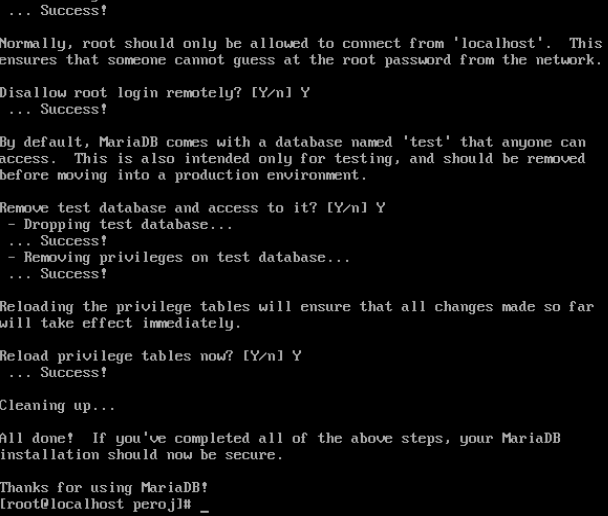
Nosotros nos vamos a ir a un script que se instala con mariadb que se llama ***mysql\_secure\_installation*** y es recomendable ejecutarlo justo después de instalar el servicio dado que te va haciendo una serie de preguntas de cara a que evites una serie de vulnerabilidades del sistema y a que se cambie la contraseña de root. Lo arrancamos, nos dice que leamos atentamente y le damos a *enter* dado que no hay configurada una contraseña de root actualmente. La configuramos como siempre para proteger la base de datos.

Ahora nos dice que hay un usuario anónimo por defecto dentro del sistema de MariaDB que permite a cualquier persona logearse en el sistema sin tener una cuenta de usuario (esto se usa solo para un propósito de test) y debemos removerlo antes de que sea un servidor de producción.

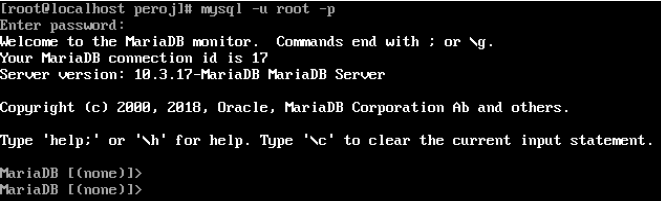


A continuación, nos dice: normalmente el usuario root solo tiene que tener acceso desde el local host para así asegurar que no se pueda acceder desde fuera. Es decir, para administrar el servidor (al igual que hacíamos con ssh) es mejor quitar el acceso a root desde fuera y si tu como administrador quieres toquetear algo de la base de datos, te conectas por ssh y ya de manera local te metes como root por el ssh de la máquina.

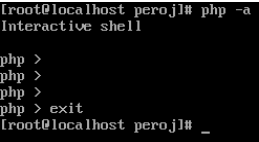
Después también vamos a eliminar la base de datos llamada *‘test’* que viene incluida por defecto en MariaDB, dado que aquí se podría acceder cualquiera e ir escalando (esto solo está hecho para testeo, nunca mantener en producción).



Ahora nos dice que recarguemos todas las tablas de los privilegios del sistema para hacer efectivos todos estos cambios ya, le decimos que si y finalizamos con la configuración de MariaDB. Comprobamos que sigue activo y lo probamos con ***mysql -u root -p*** que tenemos acceso a la base de datos (***-u*** para el usuario y ***-p*** para la contraseña).



Hasta aquí ya tendríamos LAM, entonces nos faltaría por último instalar Php, con ***yum install php***. Ahora comprobamos que entramos en la consola interactiva con ***php -a***



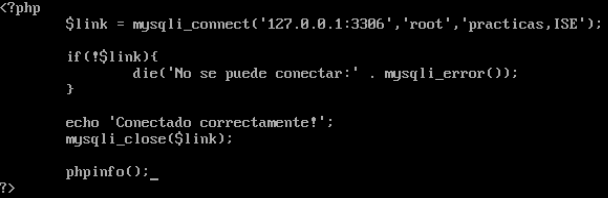
Ya tendríamos el servidor LAMP en CentOS configurado.

Ahora vamos a probar con una página web dinámica haciendo una consulta a la base de datos y la página cambia si tiene o no tiene conexión. Podemos usar un módulo de php que contiene funciones para conectarte y desconectarte y usar la base de datos, instalamos con ***yum install php-mysqlnd.x86\_64***.

Para mostrar un fichero *html* o *php* o lo que sea debe ir en ***/var/www/html/*** (el index por defecto realmente se encuentra en la carpeta donde está el fichero de configuración del servidor). Entonces aquí debemos crear un *index.php*.

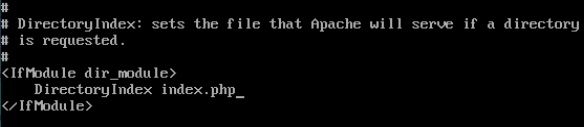


Escribimos el código para generar nuestra página dinámica básica que compruebe la conexión con la base de datos. Nos creamos una variable (link) donde almacenaremos el resultado de llamar a la función ***mysqli\_connect*** (incluida en el paquete anteriormente instalado) que se conecta pasándole la ip de la base de datos (en este caso la ip del localhost y el puerto :3306), usuario y contraseña. Si el link no es true, es decir, no se ha podido conectar correctamente a la base de datos, generamos mensaje de error y si no, si se ha podido conectar y cerramos la base de datos. Finalmente con la función ***phpinfo();*** se nos mostrará una tabla con información del sistema.



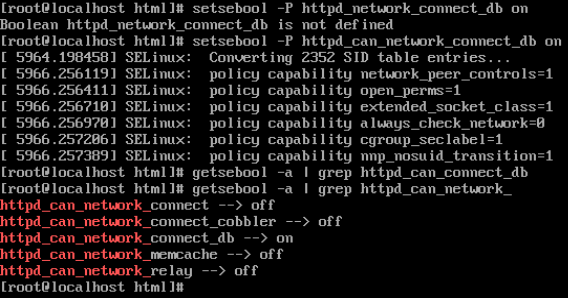
Comprobamos con ***php index.php*** que el script funciona correctamente antes de probarlo desde el host. Si devuelve un chorro de información es que es correcto, si no daría algún error en alguna línea del fichero.

Una vez comprobado el correcto funcionamiento debemos irnos al fichero de configuración de Apache ***/etc/httpd/conf/httpd.conf*** y cambiar el fichero principal del servidor al recién creado *index.php* para que así pueda interpretarlo. Podemos buscar con vi usando ***?index.html*** (con ***n*** bajamos a la siguiente encontrada y con N subimos, para salir *al modo insertar* ***i*** )**.** Y lo cambiamos al .php.

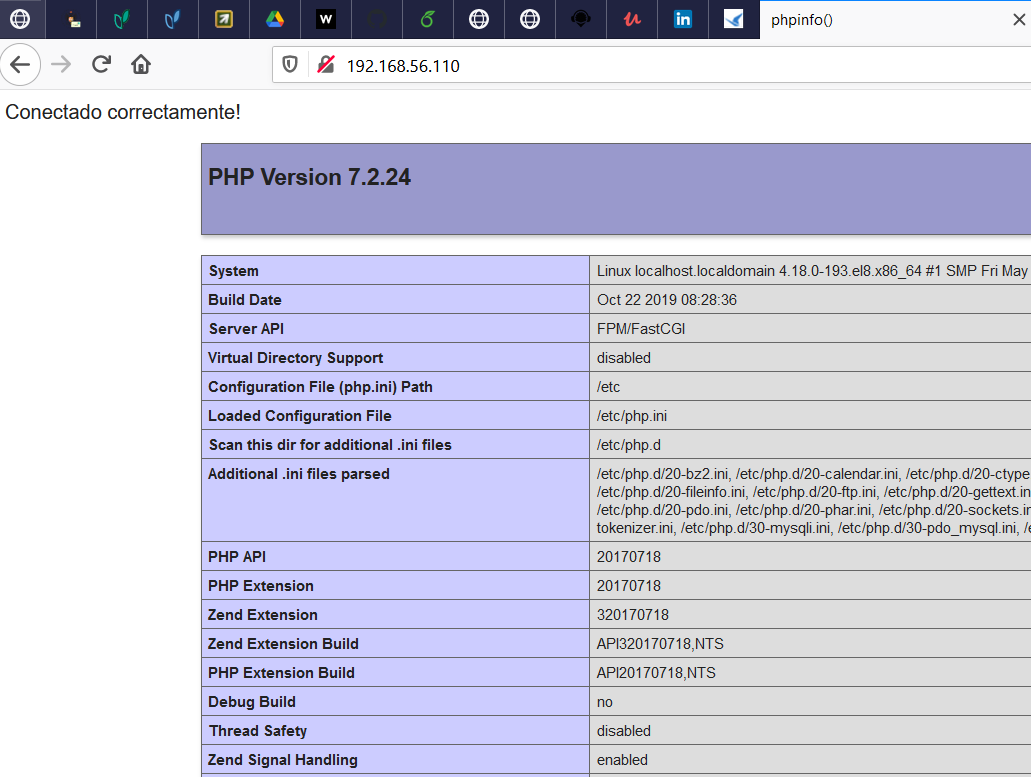


Salimos y guardamos con ***:wq*** y reiniciamos el servidor con ***systemctl restart httpd*** y comprobamos con el ***systemctl status httpd***. Si ahora nos vamos al navegador veremos que al ejecutar nuestro script nos dice que “No se puede conectar”, esto es debido al SELinux ya que no le hemos dicho a través del sistema de booleanos del sistema de seguridad, por defecto, no le deja conectarse a Apache a una base de datos. Para cambiar esto:

1. Con ***getsebool -a*** podemos ver todos estos booleanos del sistema
2. Lo buscamos con ***getsebool -a | greo httpd***
3. El flag de SELinux que nos impide conectarnos a una base de datos con apache cuando está en off es ***httpd\_can\_network\_connect\_db***.
4. Para ponerlo a *on* usamos el ***setsebool -P httpd\_can\_network\_connect\_db on***



Ahora comprobamos desde el navegador y vemos que conecta correctamente y nos aparece el mensaje y la información del servidor del phpinfo llamado desde el script.



Instalación y configuración de ***fail2ban :***

Para darle seguridad a las conexiones ssh con el servidor, lo ideal es tener un mecanismo que banee toda aquella IP que intente entrar demasiadas veces al sistema, para esto está ***fail2ban***. Herramienta muy útil a nivel de servidor personal también.

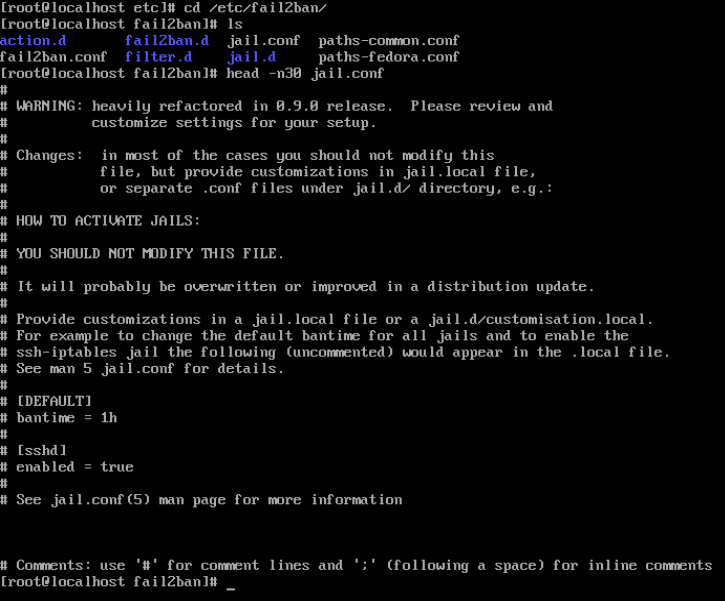
Para instalarla debemos instalar dos cosas, primero una serie de dependencias que necesita con ***yum install epel-release***. Seguidamente instalamos la herramienta con ***yum install fail2ban***. Una vez instalado, comprobamos en qué estado está con ***systemctl status fail2ban***. Como está inactivo lo levantamos con ***systemctl start fail2ban.service*** y volvemos a comprobar.



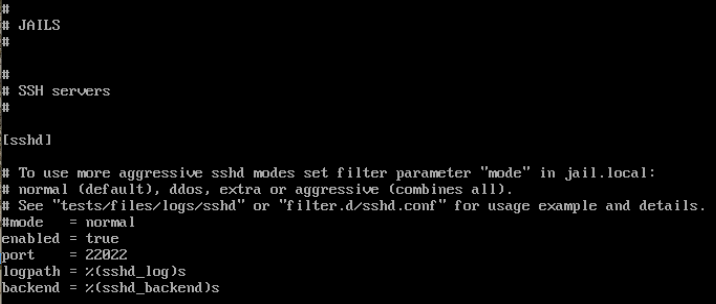
Para ver en qué estado están los baneos que ha ido haciendo la herramienta usamos ***fail2ban-client status***, y vemos que nos dice que ahora mismo hay un número de jaulas igual a 0. Las jaulas normalmente se crean una por servicio (una para ssh, otra para otro servicio...) y es donde se contendrán las direcciones IP baneadas. Por defecto, ***jail2ban*** no configura nada. Debemos entrar en la configuración e indicarle cuáles son las jaulas que queremos configurar.

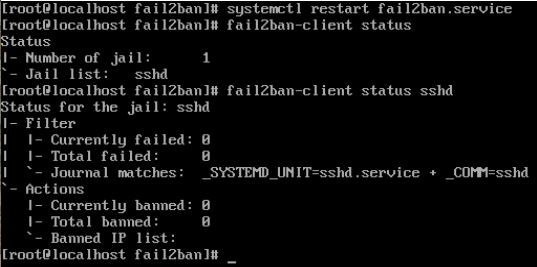
Los archivos de configuración de ***fail2ban*** están en ***/etc/fail2ban/*** donde nos encontramos varios ficheros:

* *jail.conf* --> Fichero de configuración que no se debe tocar directamente por seguridad (NO SE TOCA).



* *jail.local* --> Lo que debemos hacer es copiar el ***jail.conf*** a un ***jail.local***, y cuando recargamos ***fail2ban*** automaticamente se copia la configuración del .local al .conf. Entonces con ***cp -a jail.conf jail.local*** y abrimos con ***vi jail.local*** y buscamos las cárceles para ssh. Vemos que con la configuración actual el tiempo de baneo serán 10 minutos y el número máximo de intentos se acceso serán 5.

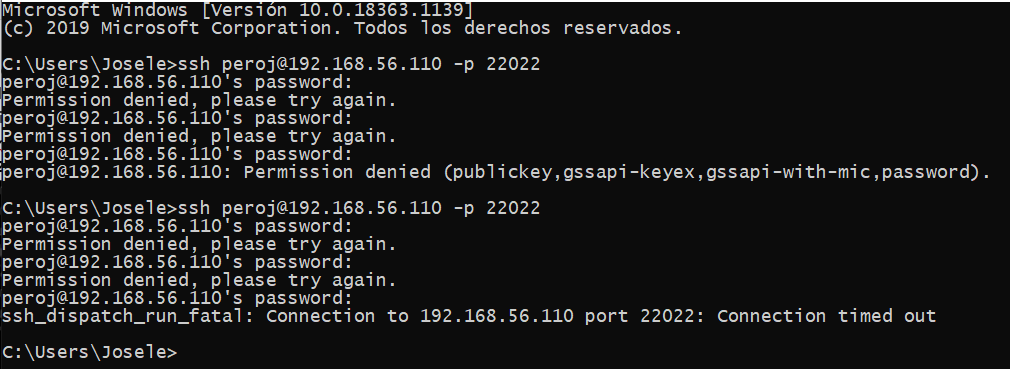


Procedemos a configurar la jaula para ssh, debemos añadir ***enable = true*** para crear la cárcel para el servicio y recordar que no tenemos el servicio ssh por el puerto por defecto (22) sino por el ***port = 22022***. Salimos y guardamos. Reiniciamos el servicio con ***systemctl restart fail2ban*** y comprobamos que ya tenemos creada la cárcel con ***fail2ba-client status*** y ***fail2ban-client status sshd***.

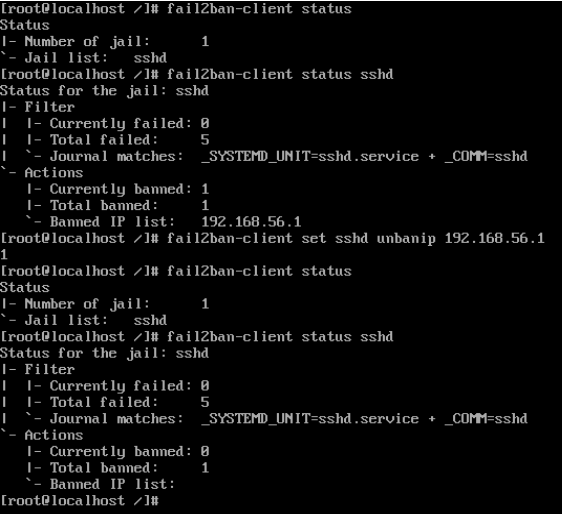
Si probamos desde UbuntuServer, antes, debemos de deshacer que no se nos solicite la contraseña al hacer ssh entre máquinas conocidas, es decir, lo que preparamos en la Lección1.

En este caso, vamos a probarlo desde el anfitrión, y antes debemos desactivar en CentOS con ***vi /etc/ssh/sshd-config*** la opción que llevamos a cabo en la Lección1 para que se vuelva a permitir usar las contraseñas en el sistema. Cambiamos la variable del fichero a ***passwordAutentication yes*** , guardamos y reiniciamos el servicio con ***systemctl restart sshd***.

Probamos a conectarnos por ***ssh*** desde el ***cmd*** de Windows e introducimos 5 veces mal la contraseña para que nos banee y nos meta en la jaula.



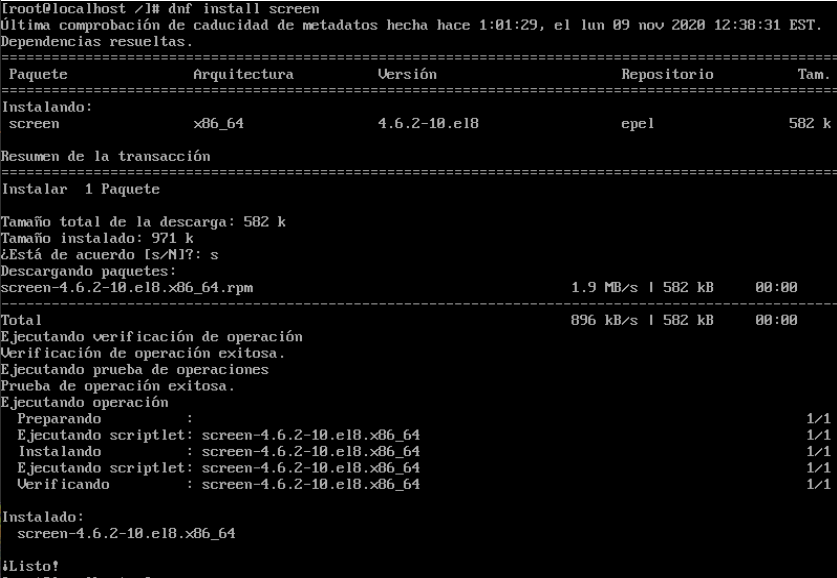
Vemos en CentOS que se nos ha baneado la IP del anfitrión y metido en la cárcel de sshd. Para desbanear una IP usamos ***fail2ban-client set sshd unbanip 192.168.56.1***. Comprobamos con el ***fail2ban-client status sshd*** que ya no estamos baneados y ya podríamos volver a conectarnos desde Windows por ssh.



Instalación y configuración de ***screen :***

Con esta herramienta podremos dejar en segundo plano sesiones de ssh. Esto es útil, por ejemplo, si estamos haciendo una copia de seguridad muy grande o un experimento y tenemos que irnos y poder cerrar la terminal o apagar el equipo (sin que se maten los procesos) y dejar el trabajo para otro momento.

Para instalar ***screen*** vamos a usar el gestor nuevo en vez de (*apt* o *snap*)con ***dnf*** ***install screen***.



Como *screen* se usa desde el cliente, es decir, debemos irnos de nuevo al anfitrión (o Ubuntu, cualquier cliente). Entonces para probarlo, hacemos ***screen*** en el *cmd* de Windows y vemos una especie de pantallazo, ejecutamos por ejemplo un ***vi /etc/ssh/sshd-config***, mientras lo tenemos abierto y modificándolo debemos irnos y dejarlo en segundo plano, entonces usamos el atajo de teclado que hace un *detached* para desconectar la pantalla y dejarlo en segundo plano: ***ctrl+a+d***. Cerramos la terminal (o apagamos el equipo cliente) y al día siguiente volvemos y queremos retomarlo por donde lo dejamos, entonces abrimos la terminal de Windows otra vez y nos volvemos a conectar con ***ssh*** [***peroj@192.168.56.110***](mailto:peroj@192.168.56.110) ***-p 22022*** y hacemos un ***screen -ls*** nos dice que hay un *socket* *detached*, es decir desconectado, en segundo plano. Para recuperarlo hacemos ***screen -r*** para reconectar a esa sesión y acto seguido vemos la pantalla del vi con el fichero de configuración. Cuando acabamos con la sesión, o hacemos directamente un ***exit*** en la sesión ó sabiendo el identificador de la sesión (4 números) podemos cerrarla con ***screen -X -S <0000\_id> quit***.

1. Cortafuegos: Barrera que protege al sistema. El sistema está conformado por muchos puertos, y si no se usa alguno de ellos, no debería mantenerse abierto ya que podría ocasionar un problema de seguridad. El cortafuegos gestiona los puertos. [↑](#footnote-ref-1)
2. Cuando en una rama trabajan varias personas normalmente antes de ponerte a trabajar haces un ***git pull*** para ver lo que han actualizado tus compañeros, y te dará la opción de con qué líneas quedarte, si con las tuyas previas (las que tienes en local) o las modificadas por tus compañeros. Si ellos han tocado cosas diferentes a ti, directamente se fusionan en el servidor sin problemas. Entonces cuando está sincronizado con los cambios de tu compañero (en el servidor) es cuando puedes subir los tuyos, porque si haces un ***git push*** y en el servidor la rama tiene cambios que tú no tienes en local te va a dar error y te va a decir que primero actualices tu directorio en local (***git pull***) y ya después sube los cambios (***git push***). [↑](#footnote-ref-2)
3. HTTP (Hypertext Transfer Protocol) año 1991 en versiones de prueba y la version estable en el 1999 y después salió la versión segura HTTPS (Hypertext Transfer Protocol Secure) - Protocolos usados en internet para transferencias de información. [↑](#footnote-ref-3)
4. A modo de curiosidad, la arquitectura LAMP fue popularizada por una famosa editorial de software hace muchos años a petición de los desarrolladores de MySQL para así englobar todos estos servicios en un estándar. [↑](#footnote-ref-4)