Practica-2-Leccion-2-Resuelta.pdf

Zukii

Ingeniería de Servidores

3º Doble Grado en Ingeniería Informática y Administración y Dirección de Empresas

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación
Universidad de Granada

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

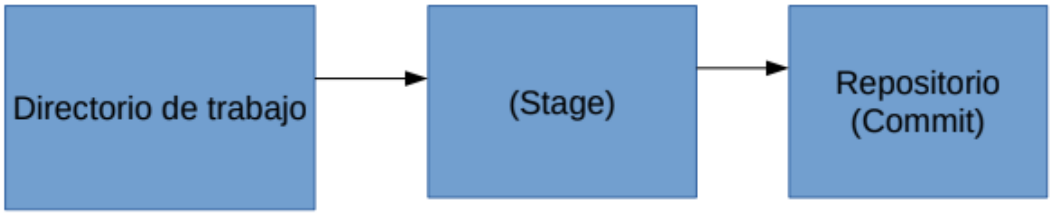
LVM Snapshots: realizar snapshots de volúmenes lógicos. Estos snapshots pueden ser utilizados para revertir el estado inicial y conocido de un LV.

//

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)

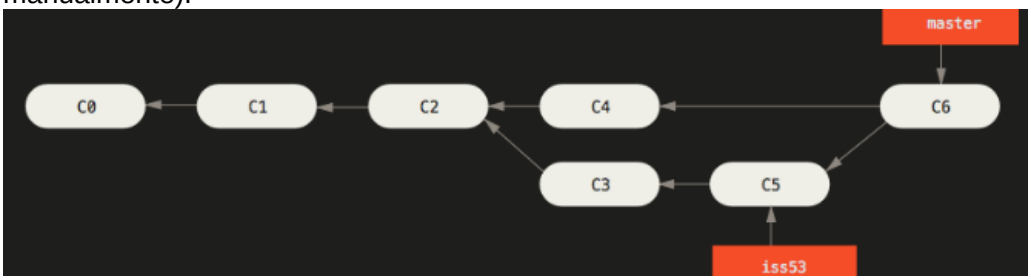
git revert <commit> (7 sha?) -> Quitará los cambios hechos al directorio respecto al commit especificado, creando un nuevo commit (si no quieres que cree un nuevo commit -n)
(git log --graph y podemos ver que hay un nuevo commit, revert del commit que especificamos)

Ramas: para desarollar parelalemente

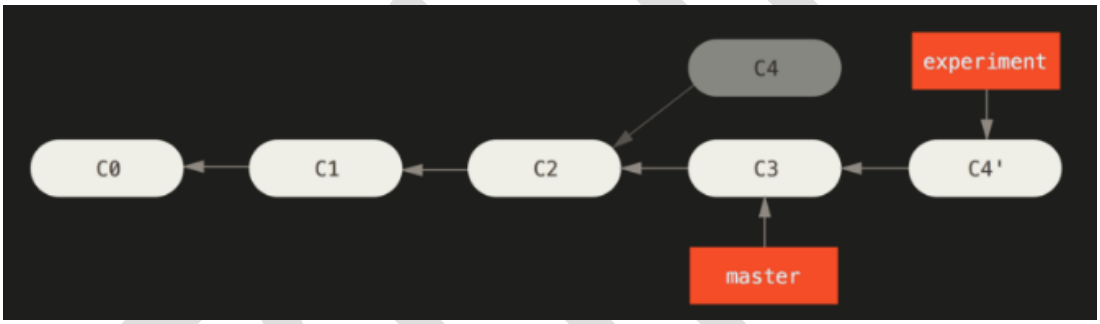
git branch -> muestra rama en la que estamos, master por defecto
git checkout <nombre_rama> -> cambiar de rama
git branch <nombre> -> crear rama
git checkout -b <nombre> -> crea rama y se cambia a ella

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”**