# SSH

## Protocol

SSH és el nom d'un protocol (definit a varis RFCs: 4251, 4252, etc) de tipus client-servidor, la principal funció del qual és permetre l'accés remot al servidor des del client mitjançant un canal segur on tota la informació transmesa està xifrada. Tècnicament, el protocol SSH està dividit internament en vàries "capes" que realitza cadascuna d'elles una funció diferent. Resumidament serien aquestes:

* Una capa de "transport" que típicament funciona sobre TCP, la qual gestiona l'intercanvi de claus inicial entre les màquines per tal d'establir un canal segur entre elles i, un cop fet això, constantment xifra i verifica la integritat de la informació transmesa per aquest canal. La seva funcionalitat és comparable a la què proporciona tot un altre protocol diferent, el TLS

* Una capa d'"autenticació d'usuari" que proporciona tot un seguit de mètodes d'autenticació per poder iniciar sessió a la màquina remota amb algun usuari vàlid. En el servidor s'han habilitar els mètodes que es vulguin i en el client, a l'hora de realitzar la connexió, es pot triar quin (o quins) d'aquests mètodes es voldra/n provar contra el servidor. Alguns d'aquests mètodes són:

  *password* : via la introducció interactiva d'una contrasenya associada a l'usuari
  *publickey* : via l'ús (no interactiu) de claus criptogràfiques que identifiquen a l'usuari. Aquestes claus són de tipus public-privat i poden estar implementades mitjançant diversos algoritmes, com ara RSA, ECDSA, etc o fins i tot -encara que no és el normal-, mitjançant certificats X.509 com en TLS
  *keyboard-interactive* : via la introducció interactiva de determinada informació mitjançant un -o més- 2FA/OTP PINs, o mitjançant resposta/es a repte/s tipus "captcha", etc...el que s'hagi configurat en particular al servidor. És un mètode més versàtil que el d'escriure una simple contrasenya (i que pot complementar-lo).
  *Mètodes GSSAPI* : proporcionats per aquesta llibreria específica, la qual implementa mecanismes d'autenticació externs com ara Kerberos o NTLM

* Una capa de "connexió" que defineix el concepte de canal. Una connexió SSH pot allotjar múltiples canals simultàniament, cadascun transferint dades en ambdós sentits. O dit d'una altra manera, aquesta capa proporciona l'habilitat de multiplexar diverses sessions (cadascuna amb la seva pròpia gestió de control de flux i finestra d'entrada) en una única connexió SSH. Aquestes sessions no tenen perquè ser "d'usuari": poden servir per transmetre dades "out-of-band" (com ara les notificacions de canvis de tamany en la finestra del terminal, els codis de sortida de processos de servidor, etc) o, sobre tot, per realitzar tasques de "forwarding" (reenviament) de dades que aconsegueixin que el servidor SSH faci de pivot entre el client i un altre servidor remot. En aquest sentit, els tipus de canals estàndard que es poden fer servir són: *shell* (per l'ús de terminals interactius i per la transferència de fitxers via SFTP o SCP), *direct-tcpip* (pel reenviament de dades de client a servidor) i *forwarded-tcpip* (pel reenviament de dades de servidor a client).

## Servidor SSHD

El servidor SSH usat més habitualment (amb diferència) en sistemes Unix és el proporcionat pel projecte OpenSSH (https://www.openssh.com). Concretament, tant a Ubuntu com a Fedora es pot instal·lar mitjançant el paquet "openssh-server" i es pot posar en marxa i gestionar com qualsevol altre servei Systemd, només que a Ubuntu el servei s'anomena "ssh" (*systemctl {start | stop | enable | disable} ssh* ) i a Fedora s'anomena "sshd" (*systemctl {start | stop | enable | disable } sshd* )

NOTA: Tal i com es pot veure si fem *systemctl cat ssh/sshd* , el servidor SSH no és més que un binari ("/usr/sbin/sshd"), el qual pot executar-se amb diversos paràmetres interessants, com per exemple (per més informació, consulteu *man sshd*) :

***-D*** : no passa el procés a segon pla (que seria el comportament per defecte). D'aquesta manera, els missatges generats pel servidor apareixeran directament a la pantalla del terminal. Aquest paràmetre s'indica a l'arxiu *.service perquè així serà Systemd el responsable de passar-lo a segon pla (amb l'avantatge de poder llavors gestionar el dimoni a la seva manera). Un altre paràmetre similar, més verbós, és ***-d*** i, més verbós encara, ***-dd***

***-f /ruta/arxiu/de/configuracio/alternatiu*** : per defecte l'arxiu de configuració del servidor SSH és, tal com de seguida veurem, "/etc/ssh/sshd_config"
***-p nº*** : número de port per on escolta el servidor (per defecte és el nº22). Aquest paràmetre té preferència sobre una eventual opció de l'arxiu de configuració anomenada *Port* (que de seguida estudiarem) però no sobre una altra similar anomenada *ListenAddress* Es pot combinar amb el paràmetre ***-4*** (per escoltar només en IPs v4) o ***-6*** (per escoltar només en IPs v6)
***-o opcio=valor*** : indica una opció de configuració de qualsevol de les que hi podrien haver en l'arxiu de configuració "/etc/ssh/sshd_config". Útil per sobreescriure el seu valor (o el valor per defecte que sigui).
***-h /ruta/arxiu/clau/privada/de/host*** : per defecte la clau privada de host del servidor SSH és, depenent de l'algoritme usat, "/etc/ssh/ssh_host_ecdsa_key" o "/etc/ssh/ssh_host_ed25519_key" o "/etc/ssh/ssh_host_rsa_key" En els casos en què el servidor SSH no s'executi com a "root", no serà capaç de llegir els arxius anteriors (en no tenir permisos suficients); és en aquests casos on aquesta directiva pot ser interessant de fer servir. Com a complement a l'ús de claus de host és l'ús de certificats ("à la TLS"), els quals s'han d'indicar llavors amb el paràmetre ***-c /ruta/certificat***
***-i*** : indica que el servidor SSHD no s'encarregarà d'obrir el socket d'escolta per sí mateix sinó que en delegarà en un altre servei (com pot ser X/Inetd o un "socket" Systemd)
**-g nº** : indica el nº de segons què el servidor SSHD esperarà a què el client pugui realitzar una autenticació correcta d'usuari (abans de desconnectar-se'n). Per defecte val 120 segons. Un valor de 0 indica no límit.

NOTA: Si volguéssim afegir algun dels anteriors paràmetres al binari "/usr/sbin/sshd" arrencat per *systemctl*, en lloc d'escriure'l/s directament a l'arxiu *.service (i fer *systemctl daemon-reload*), una alternativa seria escriure'l/s en l'arxiu "/etc/sysconfig/sshd" (a Fedora) o "/etc/default/ssh" (a Ubuntu), concretament en una línia tal com *OPTIONS="llista paràmetres"* (a Fedora) o *SSHD_OPTS="llista paràmetres"* (a Ubuntu) i reiniciar el servei. Això és possible gràcies a l'ús de la directiva *EnvironmentFile=* dins de l'arxiu *.service i al valor concret de la directiva *ExecStart=* que allà hi és present

---

L'arxiu de configuració on establirem tots els detalls del comportament del servidor OpenSSH (més enllà dels pocs paràmetres que podem especificar a la línia de comandes) és "**/etc/ssh/sshd_config**".

NOTA: En sistemes Fedora i Ubuntu al fitxer anterior hi apareix la línia *Include /etc/ssh/sshd_config.d/*.conf*, la qual fa que s'incloguin allà mateix, com a "trossos" de configuració el contingut de tots els eventuals fitxers indicats a la ruta escrita. Cal tenir en compte que si una mateixa directiva està repetida en varis llocs es tindrà en compte només la primera d'elles (això vol dir que les directives escrits dins dels fitxers "*.conf" tenen prioritat sobre les presents a l'arxiu "sshd_config".

Les directives més importants que hi podem trobar allà són:

| | |
|---|---|
| *Port nºport* | Port per on escoltarà peticions |
| *ListenAddress una.IP* | Restringeix l'escolta només a una determinada tarja. Si es posa la IP 0.0.0.0 vol dir "totes les interfícies". Opcionalment es pot indicar un nº de port, així *ListenAddress una.IP:nºport* ; en aquest cas el port indicat sobreescriurà l'indicat a la directiva *Port* |
| *PasswordAuthentication {yes\|no}* | Si val *yes*, ofereix la possibilitat d'usar aquest mètode d'autenticació al client que el demani provar |
| *PermitEmptyPasswords {yes\|no}* | Té un significat obvi |
| *PubKeyAuthentication {yes\|no}* | Si val *yes*, ofereix la possibilitat d'usar aquest mètode d'autenticació al client que el demani provar |
| *KbdInteractiveAuthentication {yes\|no}* | Si val *yes*, ofereix la possibilitat d'usar el mètode d'autenticació *keywoard-interactive* al client que el demani provar |
| *PermitRootLogin {yes\|no\|prohibit-password}* | Té un significat obvi: el valor "prohibit-password" vol dir que només s'autoritzarà s'usuari "root" si s'usa el sistema d'autenticació PubKey |

| | |
|---|---|
| *AllowUsers nomUsuari unAltre ...* | Només els usuaris que hi apareixen a la llista d'usuaris indicada tindran permés l'accés. Es poden fer servir comodins (* i ?) i el símbol "!" per negacions. També es pot indicar l'usuari amb la sintaxi nomUsuari@x.x.x.x on "x.x.x.x" representa la IP d'un host o d'una xarxa CIDR (o també un conjunt d'IPs establert via comodins) des d'on es permetrà l'accés de l'usuari indicat i només des d'allà. També està la directiva *AllowGroups* |
| *DenyUsers nomUsuari unAltre ...* | Només els usuaris que hi apareixen a la llista d'usuaris indicada tindran denegat l'accés. La sintaxi és la mateixa que la de la directiva anterior. En el cas d'haver les dues directives (*AllowUsers* i *DenyUsers*) primer s'analitza *DenyUsers* i després *AllowUsers* També està la directiva *DenyGroups* |
| *Match User nomUsuari,unAltre ...*<br>*Match Group nomGrup,unAltre ...*<br>*Match Host nomMaquina,unAltre ...*<br>*Match Address adreçaIP,unaAltra ...* | Les directives *Match* poden escriure's vàries vegades. Serveixen per indicar que les línies que apareixin escrites al fitxer de configuració a partir de llavors (fins arribar al final del fitxer o bé a una altra secció *Match* posterior) només s'aplicaran per les connexions concretes que concordin amb la condició indicada (usuari, grup, nom de màquina-client ó ip-client). Normalment, aquesta directiva s'escriu després d'haver indicat a l'arxiu les directives globals que es vol que s'apliquin a qualsevol connexió; si es repeteixen directives, la que estigui dins d'una secció *Match* sobreescriurà a la que estigui a la secció global però si una mateixa directiva apareix en diverses seccions *Match*, només la primera s'aplicarà. Es poden utilitzar comodins (*, ? i també !) i, en el cas de les adreces IP, també es poden escriure en format CIDR. Un exemple: *Match User usuari??,usuari*,!usuari3*<br><br>NOTA: *Match Host* només funciona si tenim la directiva *UseDNS* a *yes* |
| *MaxAuthTries nº* | Número d'intents que es permeten per intentar escriure bé la contrasenya (o de presentar una clau vàlida, si fos el cas) |
| *LoginGraceTime nºsec* | Temps que es deixa per introduir les credencials (si 0 no hi ha límit) |
| *MaxStartups x:y:z* | Els tres números indicats (x,y,z) indiquen, en general, el número de connexions simultànies permès que (encara) no han acabat d'autenticar-se. Concretament, el primer indica el número de connexions sense autenticar permeses a partir de les quals es començaran a descartar, el segon indica la probabilitat de descartar connexions (un cop arribat al primer número) i el tercer indica el número màxim de connexions a les quals es descartaran totes. |
| *MaxSessions nº* | Número de sessions obertes simultànies permeses per part d'un mateix origen. En el cas de valer 0 es continuarà, no obstant, permetent fer "forwarding". |
| *ClientAliveInterval nºsec*<br><br><br>*ClientAliveCountMax nº* | Temps que s'espera sense rebre dades per enviar al client corresponent un missatge de tipus "alive" per tal de veure si encara respon. Si val 0 (el seu valor per defecte) no s'enviarà cap missatge.<br><br>Número màxim de missatges "alive" que el servidor enviarà al client abans de desconnectar-lo si no en rep resposta (sempre que la directiva *ClientAliveInterval* valgui un valor diferent de 0). Per exemple, si *ClientAliveInterval* val 15 i *ClientAliveCountMax* val 3 |

| | |
|---|---|
| | (el seu valor per defecte), el client serà desconnectat al cap de 45 segons de no respondre a cap dels missatges "alive" enviats. Si val 0 es deshabilitarà aquest mecanisme de finalització de la connexió.<br><br>NOTA: Existeix una altra opció, que per defecte és *TCPKeepAlive yes*, la qual fa que s'envïin periòdicament un altre tipus de missatges "alive" -concretament, paquets ACK- que funcionen a nivell de TCP (a diferència dels altres, que són a nivell de SSH i, per tant, encriptats -fet que fa que puguin travessar millor els eventuals tallafocs-). Aquests altres paquets permeten detectar talls en la connexió i eviten així que aquestes es quedin "penjades" |
| *AcceptEnv var1 var2 ...* | Indica quines de les variables d'entorn (i els seus valors) enviades pel client (via directiva *SendEnv* o *SetEnv*, veure més avall) seran tingudes en compte en la sessió de l'usuari.<br><br>NOTA: Si la directiva *PermitUserEnvironment* val *yes* (per defecte no és així), també es tindran en compte les variables d'entorn indicades a l'arxiu "/home/user2/.ssh/environment" |
| *ForceCommand /ruta/comanda* | Executa la comanda indicada al servidor un cop s'hagi completat l'inici de sessió de l'usuari. Aquesta comanda sobreescriu la possible comanda que es pugui haver escrit al client (la qual es guardarà a la variable SSH_ORIGINAL_COMMAND per si de cas). Sol ser útil en seccions *Match*, sobre tot<br><br>NOTA: Si la directiva *PermitUserRC* val *yes* (per defecte és així) i no s'ha especificat cap valor per la directiva *ForceCommand*, s'executarà llavors automàticament la comanda indicada a l'arxiu "/home/user2/.ssh/rc" |
| *PrintMotd {yes\|no}* | Si val *yes*, es mostrarà el contingut de l'arxiu "/etc/motd" a l'usuari hagi completat l'inici de sessió. No obstant, des de la pròpia documentació oficial es recomana fer servir el mòdul PAM *pam_motd* en lloc d'aquesta directiva "built-in" per poder disposar de més flexibilitat i versatilitat en aquesta tasca |
| *Banner /ruta/arxiu* | Mostra, abans de procedir amb l'autenticació, el contingut del fitxer indicat (a mode de missatge previ, general per tothom). Per motius de seguretat, aquest contingut només pot estar format per caràcters ASCII imprimibles (no s'interpretarà cap sentència d'escapament ni cap codi de color, etc) |
| *LogLevel valor* | Indica el grau de detall mínim dels missatges de registre que es guardaran al Journal. El valor que es pot indicar (de menys detallat a més) és: QUIET, FATAL, ERROR, INFO, VERBOSE i DEBUG |
| *Subsystem sftp /usr/lib/openssh/sftp-server* | Activa al servidor SSH la possibilitat de funcionar també com a servidor SFTP |
| *UsePAM yes* | Si val *yes* (a Ubuntu i Fedora és així), s'usaran els mòduls PAM de tipus "account" i "session" en qualsevol dels tres mètodes d'autenticació estudiats i els de tipus "auth" (només) en els mètodes *PasswordAuthentication* i *KbdInteractiveAuthentication* |
| *Compression {yes\|no}* | Autoritza (o no) a què es puguin comprimir les dades a transmetre si així ho ha demanat el client |

| | |
|---|---|
| *Ciphers algo1,algo2,...* | Indica els algoritmes d'encriptació disponibles a usar (per ordre de preferència) durant la comunicació. Algun ha de coincidir amb els indicats al client per tal de què la comunicació es pugui realitzar.<br><br>NOTA: El seu valor per defecte, però, ve definit per la configuració del framework del sistema "crypto-policies" (en concret, pels valors de les directives *Ciphers*, *MACs*, *GSSAPIKexAlgorithms*, *KexAlgorithms*, *PubAcceptedAlgorithms*, *CASignatureAlgorithms* i *HostKeyAlgorithms*, indicats a l'arxiu "/etc/crypto-policies/back-ends/opensshserver.config" -mitjançant un *Include* al principi de tot de l'arxiu "sshd_config"-). Aquest framework, la configuració del qual es pot modificar mitjançant la comanda *update-crypto-policies*), és responsable de la gestió dels algoritmes d'encriptació no només de l'OpenSSH sinó d'altres llibreries com l'OpenSSL o la GnuTls, entre d'altres, així que convindrà no modificar-lo si no se sap el que es fa. Si la llista indicada en aquesta directiva *Ciphers* comença amb un "+" vol dir que els algoritmes indicats s'afegiran als per defecte (en lloc de substituir-los); si comença per un "-" vol dir que es restaran i si comença per un "^" s'ubicaran per davant dels per defecte. |
| *HostKey /ruta/arxiu/clau/privada/host* | La clau privada de host del servidor SSH és, segons l'algoritme usat, "/etc/ssh/ssh_host_ecdsa_key", "/etc/ssh/ssh_host_ed25519_key" o "/etc/ssh/ssh_host_rsa_key", però això es pot canviar amb aquesta directiva (en els casos en què el servidor SSH no s'executi com a "root", no serà capaç de llegir els arxius anteriors en no tenir permisos suficients; és en aquests casos on aquesta directiva pot ser interessant de fer servir).<br><br>NOTA: L'algoritme usat per defecte per generar aquestes claus també depèn de la configuració del framework del sistema "crypto-policies"; per saber com canviar-la consulteu *man update-crypto-policies*. Es pot, de totes formes, indicar manualment la llista d'algoritmes que el servidor reconeix com a vàlids (per usar la clau associada) amb la directiva *HostKeyAlgorithms* (la llista sencera es pot obtenir fent *ssh -Q HostKeyAlgorithms*) |

Per veure més directives o més informació sobre les anteriors, consultar *man sshd_config*

Client SSH (accés remot)

El client SSH usat més habitualment (amb diferència) en sistemes Unix és el proporcionat pel mateix projecte OpenSSH (https://www.openssh.com). Concretament, tant a Ubuntu com a Fedora ja ve instal·lat de sèrie (via els paquets "openssh-client" i "openssh-clients", respectivament).La forma general d'executar-lo és:

*ssh [user2@]nomoIPServidor [comanda]*

NOTA:Als exemples “user1” serà un usuari existent (i vàlid) a la màquina client i "user2" ho serà a la màquina servidora. Si a la comanda anterior no s'escriguís l'usuari "user2", s'intentarà entrar al servidor usant l'usuari "user1", si allà hi existís
NOTA:Si a la comanda anterior s'escriu la comanda final, no s'obrirà cap terminal perquè s'executarà aquesta comanda remotament i ja està.
NOTA: Una altra manera d'executar la comanda anterior seria fent: *ssh -l user2 nomoIPServidor [comanda]*

Alguns dels paràmetres més comuns que es poden afegir al client:

***-p nºport*** : Connecta a un port del servidor que sigui diferent del 22 (que és l'usat per defecte)
***-C*** : Comprimeix les dades abans de transmetre-les (això optimitza ample de banda però afegeix feina a la CPU)
***-c algo1,algo2,...*** : Indica els algo. de xifratge a usar (per ordre de preferència) durant la comunicació
***-i /ruta/arxiu/clau/privada/usuari*** **:** Per defecte la clau privada de l'usuari que executa el client SSH (és a dir, "user1") es troba sota la carpeta "~/.ssh" i, depenent de l'algoritme, pot anomenar-se "id_dsa" o "id_ecdsa" o "id_ecdsa_sk" o "id_ed25519" o "id_ed25519_sk" o "id_rsa". Aquest paràmetre permet indicar-ne una altra ruta i nom.
***-V***: Mostra versió del client

***-v*** : Activa el mode verbós. Molt útil per veure els passos de la comunicació (el paràmetre ***-q*** faria el contrari). Amb ***-vv*** és més verbós
***-F /ruta/arxiu/configuració*** : Si s'indica, només es tindrà en compte aquest fitxer i cap més
***-o opcio=valor*** : Indica una opció de configuració de qualsevol de les que hi podrien haver en l'arxiu de configuració "/etc/ssh/ssh_config". Útil per sobreescriure el seu valor (o el valor per defecte).

---

L'arxiu de configuració és "**/etc/ssh/ssh_config**" (general per tots els usuaris), o bé "**/home/user1/.ssh/config**" (particular per un usuari en concret, el contingut del qual sobreescriu el general; aquest arxiu ha de tenir permisos 600 obligatòriament -i propietari i grup l'usuari en qüestió, òbviament- perquè si no el client SSH es negarà a funcionar).

NOTA: Per concretar: en executar el client SSH, la preferència dels paràmetres sempre és: línia d'ordres -> fitxer config personal ("~/.ssh/config") -> fitxer config global ("/etc/ssh/ssh_config")
NOTA: En sistemes Fedora i Ubuntu al fitxer anterior hi apareix la línia *Include /etc/ssh/ssh_config.d/*.conf*, la qual fa que s'incloguin allà mateix, com a "trossos" de configuració, el contingut de tots els eventuals fitxers indicats

La primera directiva que sol haver dins d'aquest arxiu és la directiva *Host* amb un valor normalment de * Aquesta directiva indica el/s servidor/s, la connexió als quals feta des del client es veurà afectada per les directives que apareixin escrites a continuació en el fitxer de configuració (fins arribar al seu final o bé fins arribar a una altra directiva *Host*). D'aquesta manera, podem establir diferents configuracions segons el servidor amb el què connectem. Si s'indica "*" vol dir "a qualsevol servidor" però es pot fer més específic si s'indica una adreça IP (o vàries fent ús tant del propi comodí "*" com també dels comodins "?" i/o "!", o també escrivint-les una darrera l'altra separades per espais) o també un nom del servidor (o més, també escrits amb comodins i/o bé separats per espais, els quals hauran de concordar amb el que s'hagi indicat a la línia de comandes).

NOTA: Per defecte (degut al valor *no* de la directiva *CanonicalizeHostname*), si a la línia de comandes s'escriu un nom curt (és a dir, sense domini), es delega a la configuració DNS del sistema la seva conversió en FQDN per tal de procedir a la seva resolució DNS. Si, en canvi, *CanonicalizeHostname* valgués *yes*, es podria utilitzar la directiva *CanonicalDomains un.dom.ini* per tal d'establir el/s domini/s de recerca a afegir per defecte a cada nom curt indicar a la línia de comandes

L'ordre en què apareixen escrites les línies *Host* (i, de fet, qualsevol línia) dins del/s fitxer/s de configuració del client SSH és important: si hi ha vàries directives iguals només es llegeixen els valors de la primera que apareix. Per tant, en el cas de voler escriure diferents "subconfiguracions" per servidors diferents, l'estratègia a seguir seria escriure primer els blocs *Host* relatius a servidors molt concrets i anar escrivint els següents blocs cada cop més genèrics, fins arribar, si s'escau, a un darrer bloc *Host* * . Algunes de les directives més rellevants són:

| | |
|---|---|
| *Hostname nomOIPservidorAConnectar* | En lloc d'haver d'indicar l'adreça IP (o el nom DNS) com a valor de la directiva *Host*, aquest valor es pot indicar en aquesta altra directiva (escrita sota la primera). D'aquesta manera, el valor de la directiva *Host* podria passar llavors a ser un simple àlies, el qual es podria fer servir a la línia de comandes ja que el client sabria associar-lo a la IP/nom adient gràcies a tenir ubicada la directiva *Hostname* correcta sota la directiva *Host* en qüestió |
| *User usuari* | Indica l'usuari per defecte. Útil per no haver-lo d'indicar explícitament a la línia de comandes |
| *Port nºport* | Indica a quin port del servidor es connectarà el client. Útil si el servidor no escoltés al 22 (valor per defecte) |
| *PasswordAuthentication {yes\| no}* | Fa que el client utilitzi aquest mètode d'autenticació |
| *PubkeyAuthentication {yes\| no}* | Fa que el client utilitzi aquest mètode d'autenticació |

| | |
|---|---|
| *KbdInteractiveAuthentication {yes\| no}* | Fa que el client utilitzi aquest mètode d'autenticació. Antigament aquesta opció s'anomenava *ChallengeResponseAuthentication* |
| *PreferredAuthentications publickey,password,...* | Indica l'ordre preferit en anar provant els diferents mètodes d'autenticació disponibles (gràcies a les directives anteriors). El 1r valor és el mètode preferit; si no va, es prova el següent, i així. Per defecte *publickey* va primer, després *keyboard-interactive* i finalment *password* (entre d'altres) |
| *StrictHostKeyChecking {yes\|no\|ask}* | Si val *yes*, la clau pública del servidor (que associa, a partir de llavors amb una màquina remota amb una determinada adreça IP, entre altres elements) no es descarrega ni, per tant, tampoc s'afegeix automàticament a la llista de claus guardada a l'arxiu "/home/user1/.ssh/known_hosts" del client; (per afegir aquesta clau caldrà fer-ho a mà, doncs). Si val *ask*, es preguntarà interactivament a l'usuari si es vol afegir cada nova clau que es detecti que no hi és. La idea, amb aquests dos valors, és que es rebutgi qualsevol connexió a un servidor SSH la clau pública del qual no aparegui dins de l'arxiu "/home/user1/.ssh/known_hosts" del client. D'altra banda si val *no*, totes les claus de servidors a connectar, hagin canviat o no, s'afegiran automàticament a l'arxiu (fet que deshabilita a la pràctica aquest mètode de protecció però aconsegueix que la connexió a nous servidors es faci sense impediments). |
| *CheckHostIP {yes\|no}* | Si val *yes*, i al client *ssh* de terminal s'ha indicat el nom DNS de servidor en lloc d'una adreça IP, abans d'establir connexió amb aquest servidor es consultarà l'arxiu "~/.ssh/known_hosts" per comprovar si l'adreça IP resolta es correspon amb l'adreça IP que apareix en aquest arxiu (i, per tant, confirmar l'autenticitat del servidor i evitar atacs de "DNS Spoofing"). En el cas de què connectar amb un servidor amb un nom DNS dinàmic, però, aquesta directiva se sol definir amb un valor *no* per evitar l'aparició de "warnings" molestos cada cop que la seva IP canvia (la comprovació de la clau del servidor es fa igualment). |
| *ServerAliveInterval nºsec* | Si val més que 0 (valor per defecte), farà que el client enviï cada cert temps un paquet al servidor per evitar així que es tanqui la connexió degut a inactivitat. En el cas de què el servidor no respongui, s'enviaran tants paquets (cadascun en el temps que digui *ServerAliveInterval*) com indiqui la directiva *ServerAliveCountMax nºpaquets* Així, per exemple, si tenim que *ServerAliveInterval* val *10* i *ServerAliveCountMax* val *3* (el seu valor per defecte), al cap de 30 segons sense resposta del servidor, el client tallarà la connexió.<br><br>NOTA: La gràcia d'aquestes directives és que es pot obtenir el mateix efecte però sense haver de canviar la configuració del servidor mitjançant les directives *ClientAlive** |

| | |
|---|---|
| *ConnectionAttempts nº* | Indica el nombre d'intents (un cada segon) que el client provarà de fer la connexió amb el servidor abans de deixar-ho estar |
| *SendEnv var1 var2*<br><br>*SetEnv var1=valor1 var2=valor2* | Envia al shell que s'obrirà al servidor el valor de les variables d'entorn del shell client especificades. El servidor les haurà de reconèixer i acceptar mitjançant la seva directiva *AcceptEnv var1 var2*<br><br>Envia al shell que s'obrirà al servidor el valor concret indicat de les variables d'entorn especificades. El servidor les haurà de reconèixer i acceptar mitjançant la seva directiva *AcceptEnv var1 var2* |
| *BatchMode {yes\|no}* | Si val *no*, no es demana contrasenya/passphrase. Aquest mode és útil per ficar-ho en shell scripts automàtics que no obliguin a què hi hagi una persona al davant de la màquina, (té sentit si no es fa servir ni *PasswordAuthentication* ni *KbdInteractiveAuthentication*) |
| *LocalCommand /ruta/comanda* | Indica una comanda que s'executarà a la màquina client un cop s'hagi connectat al servidor. Com a paràmetres d'aquesta comanda es poden escriure els valors especials: %d -ruta de la carpeta home local-, %h -nom del host remot-, %l -nom del host local-, %r -nom d'usuari remot- , %u -nom d'usuari local- entre altres (la llista sencera es troba a l'apartat "TOKENS" de *man ssh_config*. Aquesta directiva només funciona, però, si la directiva *PermitLocalCommand* val *yes* (per defecte val *no*) |
| *Compression {yes\|no}* | Comprimeix les dades abans de transmetre-les. El nivell de compressió ve donat per la directiva *CompressionLevel nº* (on 1="fast", 9="best"). Això funcionarà sempre i quan el servidor accepti aquesta configuració |
| *Ciphers algo1,algo2, ...* | Indica els algoritmes d'encriptació a usar (per ordre de preferència) durant la comunicació<br><br>NOTA: Aquí també s'utilitza, com passava al servidor (però ara com a darrer recurs), la configuració del framework "crypto-policies" indicada dins de l'arxiu "/etc/crypto-policies/back-ends/opensshserver.config" -mitjançant un *Include* al final de tot de l'arxiu "/etc/ssh/ssh_config"-). |
| *IdentityFile /ruta/arxiu/clau/privada/usuari* | Per defecte la clau privada de l'usuari que executa el client SSH (és a dir, "user1") es troba sota la carpeta "~/.ssh" i, depenent de l'algoritme, pot anomenar-se "id_dsa" , "id_ecdsa", "id_ecdsa_sk", "id_ed25519", "id_rsa" o "id_ed25519_sk". Aquest paràmetre permet indicar-ne una altra ruta i nom. |

Per veure més directives interessants (com *ControlMaster/ControlPath*,...) o més informació sobre les anteriors, consultar *man ssh_config*

Client SCP (copia remota)

El client *scp* ja ve integrat dins del paquet "openssh-client/s". Bàsicament permet aquestes accions:

*"Pujar" arxius locals (es poden indicar comodins si es vol indicar més d'un) a una carpeta remota:
*scp /ruta/arxius user2@ipServ:/ruta/carpeta*

NOTA: Si no s'indica carpeta remota, per defecte és la HOME de user2
NOTA: També es pot indicar el destí remot amb la sintaxi *scp://user2@ipServ:/ruta/carpeta*

*"Baixar" arxius remots (es poden indicar comodins si es vol indicar més d'un) a una carpeta local:
*scp user2@ipServ:/ruta/arxius /ruta/carpeta*

*"Pujar" carpetes locals (amb tot el seu contingut) dins d'una carpeta remota:
*scp **-r** /ruta/carpeta1 user2@ipServ:/ruta/carpeta2*

*"Baixar" carpetes remotes (amb tot el seu contingut) dins d'una carpeta local:
*scp **-r** user2@ipServ:/ruta/carpeta2 /ruta/carpeta1*

Alguns dels paràmetres més interessants que té (els quals són molts semblants als de *ssh*) són:

***-P nºport*** : Connecta a un port del servidor que sigui diferent del 22 (que és l'usat per defecte)
***-C*** : Comprimeix les dades abans de transmetre-les (això optimitza ample de banda però afegeix feina a la CPU)
***-c algo1,algo2,...*** : Indica els algoritmes d'encriptació a usar (per ordre de preferència) durant la comunicació
***-i /ruta/arxiu/clau/privada/usuari*** **:** Per defecte la clau privada de l'usuari que executa el client SSH (és a dir, "user1") és, depenent de l'algoritme, "~/.ssh/id_dsa" o "~/.ssh/id_ecdsa" o "~/.ssh/id_ecdsa_sk" o "~/.ssh/id_ed25519" o "~/.ssh/id_ed25519_sk" o "~/.ssh/id_rsa". Aquest paràmetre permet indicar una altra
***-v*** : Activa el mode verbós. Molt útil per veure els passos de la comunicació (el paràmetre ***-q*** faria el contrari). Amb ***-vv*** és més verbós
***-F /ruta/arxiu/configuració*** : Si s'indica, només es tindrà en compte aquest fitxer i cap més
***-p*** : A l'igual que el paràmetre homònim de la comanda *cp*, manté el propietari, permisos i dates de modificació i accés originals als arxiu-còpia
***-l nº*** : mara un límit de KB/s màxims permesos d'ample de banda
***-o opcio=valor*** : Indica una opció de configuració de qualsevol de les que hi podrien haver en l'arxiu de configuració "/etc/ssh/ssh_config". Útil per sobreescriure el seu valor (o el valor per defecte).

NOTA: Una comanda alternativa a *scp* seria *rsync*, el qual pot actuar també com a client per pujar/baixar arxius/carpetes tenint al davant un servidor SSH estàndard (sempre que en aquest servidor estigui instal·lat el binari *rsync*, això sí).

Client SFTP (còpia remota)

El client *sftp* ja ve integrat dins del paquet "openssh-client/s". No obstant, per a què funcioni el servidor ha d'estar configurat adientment (veure directiva *Subsystem*). Per executar-lo cal escriure *sftp user2@ipServ:/ruta/carpeta*

NOTA:Si no s'escriu la carpeta on es vol entrar, per defecte és la HOME de user2.
NOTA: També es pot indicar el destí remot amb la sintaxi *sftp://user2@ipServ:/ruta/carpeta*

Les comandes internes del shell "sftp" són les mateixes que les d'un client "ftp" normal. Per exemple:

| | | | | |
|---|---|---|---|---|
| *cd* (en local: *lcd*) | *pwd* (en local: *lpwd*) | *ls* (en local: *lls*) | *mkdir* (en local *lmkdir*) | *rmdir* |
| *rm* | *get* | *put* | *progress* | *exit/quit* |
| *!comandaShell* | *help* | | | |

Alguns dels paràmetres més interessants que té (els quals són molts semblants als de *scp*) són:

***-P nºport*** : Connecta a un port del servidor que sigui diferent del 22 (que és l'usat per defecte)
***-C*** : Comprimeix les dades abans de transmetre-les (això optimitza ample de banda però afegeix feina a la CPU)
***-c algo1,algo2,...*** : Indica els algoritmes d'encriptació a usar (per ordre de preferència) en la comunic
***-i /ruta/arxiu/clau/privada/usuari*** **:** Per defecte la clau privada de l'usuari que executa el client SSH (és a dir, "user1") és, depenent de l'algoritme, "~/.ssh/id_dsa" o "~/.ssh/id_ecdsa" o "~/.ssh/id_ecdsa_sk" o "~/.ssh/id_ed25519" o "~/.ssh/id_ed25519_sk" o "~/.ssh/id_rsa". Aquest paràmetre permet indicar una altra
***-v*** : Activa el mode verbós. Molt útil per veure els passos de la comunicació (el paràmetre ***-q*** faria el contrari). Amb ***-vv*** és més verbós
***-F /ruta/arxiu/configuració*** : Si s'indica, només es tindrà en compte aquest fitxer i cap més
***-p*** : Manté el propietari, permisos i dates de modificació i accés originals als arxiu-còpia
***-l nº*** : mara un límit de Kb/s màxims permesos d'ample de banda
***-o opcio=valor*** : Indica una opció de configuració de qualsevol de les que hi podrien haver en l'arxiu de configuració "/etc/ssh/ssh_config". Útil per sobreescriure el seu valor (o el valor per defecte).
***-b /ruta/fitxer*** : Indica un fitxer on hi hauran escrites les comandes "sftp" (una per línia) per tal d'executar-les de forma automàtica (és a dir, sense interacció). Això és molt útil per script, tot i que per a què sigui realment útil, el mètode d'autenticació emprat hauria de ser també no interactiu (amb claus, per exemple) i així realitzar tot el procés complet de forma autònoma. Recomanable combinar-ho amb el paràmetre *-o Batchmode=no*

A més del client de consola també es pot fer servir el Nautilus (el gestor de fitxers de Gnome) com a client Sftp si s'escriu a la seva barra de navegació el següent: *sftp://user2@ipServ* (o bé *ssh://user2@ipServ ).* També es pot fer servir l'opció "Altres ubicacions"->"Conectar al servidor..."

Per fer que determinats usuaris (els quals, per comoditat, els agruparem en un grup, anomenat "sftponly", creat amb la comanda *sudo groupadd sftponly*) creats de la següent manera per a què no pugui obrir cap shell (aquí es mostra per exemple un usuari anomenat "pepito": *sudo useradd -s /bin/false -G sftponly -m pepito && sudo passwd pepito*) només puguin utilitzar Sftp dins de la seva gàbia (és a dir, per impedir que puguin accedir a fitxers que no són dins d´una determinada carpeta, la qual farem que sigui la seva pròpia carpeta personal), cal editar la configuració del servidor SSH, posant en comptes de la línia *Subsystem sftp /usr/lib/openssh/sftp-server*, la línia *Subsystem sftp internal-sftp* i afegint al final (o en algun arxiu *.conf dins de la carpeta "/etc/ssh/sshd_config.d") el següent...:

*Match Group sftponly*
*ChrootDirectory %h* #%h representa la carpeta personal de l'usuari en qüestió. Podria ser una ruta absoluta
*ForceCommand internal-sftp*

NOTA: És important que el propietari i grup de la carpeta indicada a la directiva *ChrootDirectory* sigui l'usuari "root" i que tingui els permisos 755. Per tant, en el cas del nostre usuari "pepito" caldrà executar les comandes *sudo chown root:root /home/pepito* i *sudo chmod 755 /home/pepito* A partir d'aquí, llavors caldrà crear a mà subcarpetes concretes amb els permisos i propietaris adients per a què l'usuari "pepito" i/o el grup "sftponly" puguin escriure-hi (per exemple, fent això: *sudo mkdir /home/pepito/{up,down} && sudo chmod -R 755 /home/pepito/{up,down} && sudo chown -R pepito:sftponly /home/pepito/{up,down}* )
NOTA: En sistemes Fedora amb SELinux en mode "enforcing" cal executar a més *setsebool -P ssh_chroot_rw-homedirs on* Si es prefereix posar SELinux en mode "permissive", però, llavors cal només executar *setenforce permissive*

## Client SSHFS (muntatge d'unitats remotes)

Al client només cal executar (prèvia instal.lació del paquet "sshfs" a Ubuntu, "fuse-sshfs" a Fedora):

*sshfs user2@ipServ:/ruta/carpeta/remota /ruta/carpeta/local -o uid=nºuidlocal,gid=nºgidlocal*

...on les opcions *uid* i *gid* indiquen que el propietari dels fitxers accessibles a "/ruta/carpeta/local" passin a ser l'especificat pels valors *uidlocal/gidlocal* (el qual representen un usuari local en comptes de "user2"). Això és per evitar inconsistències en els permisos (si no es fes així, estaríem veient en un punt de muntatge

local uns fitxers propietat d'un altre usuari no existent al sistema). Altres opcions interessants poden ser:

***-p nºport*** : Connecta a un port del servidor que sigui diferent del 22 (que és l'usat per defecte)
***-o ro*** : Munta la carpeta en mode només lectura
***-o allow_other*** : Permet l'accés a altres usuaris diferents del propietari
***-o idmap {user|file}*** : Defineix altres tipus de mapeig entre usuaris remots i locals
***-o reconnect*** : Intenta reconnectar automàticament amb el servidor en cas de tallada puntual

NOTA: Per saber l'identificador d'un usuari, recordeu que es pot executar la comanda *id nomusuari*

Per a què el punt de muntatge sigui permanent caldrà afegir la següent línia al fitxer "/etc/fstab":

*sshfs#user2@ipServ:/ruta/remota /ruta/local fuse auto,user,uid=º,gid=nº,allow_other 0 0*

NOTA: No obstant, la línia anterior només funciona si tenim activada l'autenticació per claus (en ser aquest un procés automàtic); en el cas d'usar autenticació per contrasenyes, en teoria en llegir-se el fitxer "/etc/fstab" durant l'arranc del sistema s'hauria de preguntar interactivament la contrasenya per accedir al servidor SSH però no es fa i aquest punt de muntatge no es munta (silenciosament). De fet, és per això que a les opcions no s'ha posat "defaults": aquesta opció equival a: "rw, auto, **nouser**, ..." amb la qual cosa no deixaríem que un usuari local normal pogués muntar la carpeta, cosa que degut al motiu explicat, haurà de fer quan hi vulgui accedir executant *mount /ruta/carpeta/local*

Per desmuntar el punt en un moment donat, cal executar al client (com amb qualsevol altre sistema de fitxers muntat mitjançant la tecnologia FUSE, de fet) la comanda: *fusermount -u /ruta/carpeta/local*

NOTA: La tecnologia FUSE (en la qual està basada "sshfs") permet implementar sistemes de fitxers que funcionen a nivell d'usuari, no pas a nivell de kernel. Això permet disposar de sistemes de fitxers "alternatius" més enllà dels típics suportats pels sistemes Linux

## Claus de màquina

A la instal.lació dels paquets, ja siguin de tipus servidor o client, es generen el parell de claus pública/privada dins de la carpeta "**/etc/ssh**" que identifiquen la màquina, i que a cada comunicació serviran per a generar al seu torn una clau de sessió única entre les dues màquines dels extrems, de manera que no hi pugui haver cap màquina impostora. Aquest parell de claus pub/priv són els fitxers anomenats "**ssh_host_*_key.pub**" i "**ssh_host_*_key**", respectivament (on "*" representa el nom de l'algoritme criptogràfic emprat per crear-les: "rsa", "ecdsa", etc.

La idea és que la clau pública del servidor es gravarà (després de preguntar primer si el client té la directiva *StrictHostKeyChecking* amb el valor per defecte *ask*) en els clients a l´arxiu "**/home/user1/.ssh/known_hosts**" la primera vegada que aquests es connectin, per assegurar-se en posteriors connexions que el servidor no sigui un impostor (ja que l'únic servidor correcte serà qui tingui la clau privada corresponent a aquesta clau pública, el qual, per definició, només en pot ser un al món). És a dir, el procés és *clau pública servidor ---> arxiu known_hosts client* , encara que això també podria passar de forma recíproca, si es volgués "autenticar" un determinat client. Si algú fos molt paranoic i no es fiés d´aquesta primera vegada, sempre pot copiar la clau pública del servidor en un "pendrive" i copiar-la “manualment” dins de l'arxiu "kown_hosts" del client., i així ja es tindran i no es preguntarà res la primera vegada.

NOTA: Juntament amb la clau pública del servidor en qüestió, dins de l'arxiu "known_hosts" es guarda el nom DNS i/o adreça IP d'aquest servidor per així poder associar cada clau amb el servidor corresponent i, per tant, poder triar-la adientment segons el servidor on s'hi vulgui connectar en cada moment

Per saber, a la pregunta que apareix abans d'acceptar la clau, si aquesta clau pública es correspon o no a la del servidor en qüestió, es mostra a la pantalla l'anomenat "fingerprint" de la clau, que no és res més que un resum d'un pocs caràcters calculat amb un algoritme de "hash" estàndard per no haver de mostrar tota la clau sencera, que és molt llarga. La idea seria llavors comparar aquest "fingerprint" mostrat amb el vertader del servidor, el qual es pot calcular si s'executa (al servidor) la comanda *ssh-keygen -l -f /etc/ssh/ssh_host_*_key.pub* (on es pot afegir el paràmetre *-v* per mostrar una representació visual del fingerprint (més fàcil de reconèixer a simple vista; per a què aquesta representació visual aparegués també

juntament amb la pregunta de la primera connexió a la pantalla del client, caldria que la directiva de client *VisualHostKey* valgués *yes*).

D'altra banda, es poden obtenir claus públiques de servidors d'una forma força automatitzada abans ni tan sols de contactar-hi amb ells per iniciar una sessió SSH amb la comanda especialitzada *ssh-keyscan {ipServ|nomServ}* (interessants els seus paràmetres *-f*, *-t* i *-H*, veieu la seva pàgina del manual per detalls).

Finalment, per esborrar de l'arxiu "~/.ssh/known_hosts" una determinada clau (corresponent a la IP/host indicat) es pot executar la comanda *ssh-keygen -R {ipServ|nomServ}* (aquesta és molt útil quan el servidor ha canviat d'IP, per exemple, i llavors el client rebutja connectar-s'hi perquè creu que és un servidor impostor; a més, funciona també encara que els noms estiguin "hashejats" dins de l'arxiu).

> NOTA: No cal usar *grep* pot comprovar si la clau d'un determinat servidor es troba dins de l'arxiu "known_hosts" o no; es pot fer directament amb la comanda *ssh-keygen -F {ipServ|nomServ}* (la qual, a més, funciona amb noms "hashejats")

Claus d'usuari

Existeix una altra forma de loguejar-se en el servidor SSH que implica no haver d´escriure una contrasenya cada cop. Es tracta d´utilitzar autenticació basada en un parell de claus pública/privada d´usuari. La idea és que cada usuari generi el seu parell propi, i coloqui la seva clau pública al servidor (la privada romandrà secreta a la màquina client, opcionalment protegida per una "passphrase").

Tot i que un avantatge obvi respecte l'autenticació per contrasenya és que l'autenticació per claus permet accedir directament al servidor de forma no interactiva, la "gràcia" d'aquest mecanisme és que, paradoxalment, és molt més segur que utilitzar contrasenyes perquè en basar-se en una clau privada que només té l'usuari "user1" guardada dins de la seva màquina client, ningú des de cap altre màquina client podrà entrar-hi al servidor. Això és possible perquè quan el servidor rebi la petició per entrar d'un determinat usuari provinent d'un determinat client, comprovarà que la seva clau pública associada (que el servidor ja hauria de tenir emmagatzemada prèviament) es correspongui efectivament amb el client en qüestió: si no és així, no hi haurà manera d'entrar.

El procediment per implementar aquest tipus d'autenticació basat en claus és el següent (suposarem que essent "user1" al client", volem connectar-nos com a "user2" a la màquina servidora):

**1.-**Crear, a la màquina client, el parell de claus per "user1", així...:

*ssh-keygen [-b 1024] [-t {rsa|ecdsa|ed25519|...}] [-f nomarxiuclaus]*

... on ***-b*** és el nº de bits de la clau (pot ser 512, 1024 o 2048...però consulteu la pàgina del manual pels valors adients segons l'algoritme emprat per generar la clau en qüestió) , ***-t*** és precisament el nom de l'algoritme criptogràfic que es farà servir per crear-la i ***-f*** serveix per indicar un altre nom als fitxers generats diferents dels per defecte, els quals són "**/home/user1/.ssh/id_*nomAlgoritme***" (clau privada) i "**/home/user1/.ssh/id_*nomAlgoritme*.pub**" (clau pública)

La comanda anterior pregunta interactivament una "passphrase", que representa una "contrasenya" incrustada dins de la clau privada per protegir el seu ús si algú la roba (ja que obtenint la clau privada ja tindrem l'autenticació trencada). No obstant, si es posa, ens la preguntarà cada cop que ens volguem connectar, fet que farà que perdem l'avantatge dels inicis de sessió no interactius.

> NOTA: Es pot canviar una passphrase un cop estigui definida, amb aquesta comanda: *ssh-keygen -p -q -t rsa -f /ruta/id_rsa.pub -P passantic -N passnou* El paràmetre ***-q*** evita els missatges per la sortida estàndar. El paràmetre ***-p*** es per canviar la passphrase sense crear un nou fitxer de clau privada.

**2.-**Copiar a la màquina servidora, dins de l´arxiu "**/home/user2/.ssh/authorized_keys**", el contingut de l´arxiu "id_*xxx*.pub" (la clau pública generada al pas anterior). Per fer això, es pot executar, a la màquina client, alguna de les següents comandes, a escollir:

*cat ~/.ssh/id_xxx.pub | ssh user2@ipServ "cat - >> ~/.ssh/authorized_keys"*
o *scp ~/.ssh/id_xxx.pub user2@ipServ:.ssh/authorized_keys* (si només farem servir aquest client)
o *ssh-copy-id -i ~/.ssh/id_xxx.pub user2@ipServ*

NOTA: Els permisos de la carpeta "/home/user2/.ssh" del servidor han de ser 700 i els de l´arxiu "authorized_keys" han de ser 600 obligatòriament

NOTA: Caldria confirmar, a més, que la configuració del servidor tingui activada aquest tipus d'autenticació. En concret, al seu arxiu de configuració han d´aparèixer com a mínim la directiva *PubkeyAuthentication yes* (per defecte, ja és així). Opcionalment també es podria deshabilitar l'accés per contrasenya amb *PasswordAuthentication no* D'altra banda, també és interessant comprovar la directiva *AuthorizedKeysFile %h/.ssh/authorized_keys* , la qual serveix per indicar l'arxiu on es guarden les claus públiques dels clients (per defecte ja té el valor aquí mostrat així que aquí no caldria fer res tampoc)

NOTA: L´arxiu "authorized_keys" del servidor també es pot fer servir per altres coses. Per exemple,si volem que cada cop que es connecti un client s´executin automàticament una sèrie de comandes, el que s´ha de fer és escriure al principi de tot de la línia (deixant un espai en blanc entre mig del que ve després) el següent: *command="rutaprograma"* (impedint així que l´usuari executi el que ell havia escrit: en tot cas, aquest es posaria com a valor de la variable SSH_ORIGINAL_COMMAND). Si volem que l´usuari en qüestió només es pugui connectar des d´una màquina concreta, s´ha d´escriure *from="ip1,ip2"* (s´accepta *,? y ! i si s'usa *UseDNS yes* es podran indicar noms de màquines en lloc d'IPs). Altres opcions són: *no-port-forwarding*, *no-agent-forwarding*, *nopty* (deshabilita la reserva d'un terminal per l'execució de comandes), *environment="VARIABLE=valor"* (estableix variables d'entorn a la sessió d'"user2", sempre i quan la directiva *PermitUserEnvironment* valgui *yes)*, etc. Totes aquestes opcions han d'escriure's separades entre sí per comes.

## Anell de claus

Per poder fer servir "passphrases" però, a més, poder fer que el procés d'autenticació continuï sent totalment automàtic, podem utilitzar a la màquina client el programa *ssh-agent*, que és el responsable de mantenir en RAM una "base de dades" de "passphrases" (l'anomenat "anell de claus") per fer-les disponibles (concretament a través d'un "socket" intern del sistema, la ruta del qual es trobarà disponible com a valor de la variable SSH_AUTH_SOCK) als clients SSH que les puguin necessitar en un moment donat per trobar la "passphrase" adient a la clau a utilitzar en aquell moment.

A les distribucions actuals, el programa *ssh-agent* se sol posar en marxa automàticament en iniciar sessió local en un entorn gràfic via el "display manager" corresponent (així que en principi no cal fer "res", per tenir a l'abast l'anell de claus, més enllà d'iniciar sessió dins d'un escriptori) però si iniciem sessió local en un terminal (via *login*, per exemple), aquest programa no se sol posar en marxa, així que hem de fer les passes indicades a la nota següent si volem gaudir de la seva utilitat:

NOTA: A continuació es detallen els passos per posar en marxa *ssh-agent* a l'inici d'una sessió local de forma "artesana":

**1.**-Crea l'arxiu "**~/.config/systemd/user/ssh-agent.service**" amb el següent contingut:

```
[Unit]
Description=SSH key agent's systemd user service
[Service]
Type=simple
Environment=SSH_AUTH_SOCK=%t/ssh-agent.socket
ExecStart=/usr/bin/ssh-agent -D -a $SSH_AUTH_SOCK
[Install]
WantedBy=default.target
```

NOTA: Bàsicament, el que fan les línies anteriors és crear un servei d'usuari que posa en marxa ssh-agent i l'associa a un determinat "socket", que és per on la resta de programes accediran a l'anell
NOTA: El símbol *%t* equival al valor de la variable d'entorn $XDG_RUNTIME_DIR, el qual es correspon a la carpeta pròpia de l'usuari on s'ubiquen els elements temporals (com ara el socket utilitzat per *ssh-agent*). Es pot veure la llista de símbols %xxx admesos a l'apartat "Specifiers" de *man systemd.unit*

**2.**-Afegeix al final de l'arxiu "**~/.bashrc**" la següent línia:

```
export SSH_AUTH_SOCK="$XDG_RUNTIME_DIR/ssh-agent.socket"
```

NOTA: Bàsicament, el que fa la línia anterior és establir a la variable SSH_AUTH_SOCK la ruta adient del "socket" obert per ssh-agent al pas anterior. Això és necessari per a què tots els clients executats dins del Bash sàpiguen quina ruta és i, per tant, hi puguin contactar en cas de necessitat

**3.**-Inicia/Habilita el servei creat al 1r pas: *systemctl --user enable ssh-agent && systemctl --user start ssh-agent*

NOTA: La NOTA anterior explica com posar en marxa *ssh-agent* a nivell d'inici de sessió i, per tant, fer disponible la base de dades de claus mantinguda per ell a tots els programes que s'executin dins de la sessió de l'usuari en qüestió. No obstant, una alternativa més "fina" és associar *ssh-agent* només a un programa específic (i tot els programes fills d'aquest però només aquests i prou). Per exemple, si volguéssim vincular *ssh-agent* només a un determinat terminal, podríem executar *ssh-agent gnome-terminal &* D'aquesta manera, en tancar el terminal ens estaríem assegurant de matar el procés *ssh-agent* (cosa que, si ho fem de la forma explicada a la NOTA anterior, només passarà quan tanquem l'inici de sessió local) i, per tant, d'"esborrar" de la memòria la "passphrase" (i així haver-la de tornar a introduir al següent cop). Això pot ser una configuració més segura. En tot cas, sempre podrem matar l'agent amb la comanda *ssh-agent -k*

Un cop l'agent estigui funcionant (podeu comprovar-ho sempre fent *ps -ef | grep "ssh-agent"*) cal, però, afegir el contingut a l'anell (és a dir, les "passphrases" a compartir pels eventuals clients SSH -comanda *ssh, scp, sftp, sshfs*, etc- que les necessitin). Això es pot fer bàsicament de dues maneres:

* Manualment: executant (abans de qualsevol altre client SSH per a què així les "passphrases" afegides ja estiguin disponibles des del principi) la comanda *ssh-add* , tal qual. Si aquesta comanda, en executar-se, troba la clau privada SSH de l'usuari local ("/home/user1/.ssh/id_rsa"), preguntarà la "passphrase" per incorporar-la a l'anell i ja estarà tot fet (mentre duri la sessió local d'aquest usuari). Aquesta comanda *ssh-add* té, però diversos paràmetres interessants, com són:

  *nomclau* : Si la clau privada concreta que es desitja afegir no té la ruta o nom per defecte
  *-t nºsec* : Indica el temps que romandrà la clau en qüestió en l'anell (per defecte, per sempre)
  *-l* : Llista els fingerprints de les claus que estan dins l´anell
  *-d nomclau* : Esborra la clau indicada de l´anell
  *-D* : Esborra totes les claus de l´anell

* Automàticament: afegint a la configuració dels clients SSH ("**~/.ssh/config**") la línia *AddKeysToAgent yes* Aquesta línia fa que siguin les pròpies comandes *ssh/scp/sftp/...* les que elles mateixes es comuniquin amb *ssh-agent* per tal d'afegir a l'anell, el primer cop que la facin servir, la "passphrase" vinculada a la clau privada utilitzada (sense necessitat, doncs, d'executar *ssh-add* per res). Això sí, d'aquesta manera, la primera vegada que s'executa un client SSH de la suite sí que es demanarà la "passphrase" però els següents cops ja no (fins que es tanqui la sessió local de l'usuari en qüestió).

  NOTA: Altres valors possibles de la directiva *AddKeysToAgent* poden ser *ask_ssh* (requereix confirmació del programa SSH_ASKPASS) o *confirm* (equivalent a usar el paràmetre *-c* de *ssh-add*), entre altres

  NOTA: La configuració indicada al paràgraf anterior no és la única manera d'afegir "passphrases" a l'anell automàticament... De fet, en aquest sentit, el escriptoris moderns disposen dels seus propis mecanismes alternatius per interaccions amb *ssh-agent* que vinculen automàticament l'addició de les "passphrases" de les claus privades d'un usuari amb l'inici de sessió local d'aquest usuari. Però això ja és un altre tema.

En tot cas, un cop ja haguem afegit la "passphrase" en qüestió a l'anell de la manera que sigui, tal com hem dit, l'agent al seu torn la passarà automàticament a qualsevol client SSH executat durant de la sessió, cada cop que aquest la necessiti. D'aquesta manera, en executar qualsevol client SSH, aquest no preguntarà mai la "passphrase".

**EXERCICIS:**

**1.- a)** Instal·la el paquet "openssh-server" en una màquina virtual qualsevol (Ubuntu o Fedora) però que tingui la seva tarja en mode adaptador pont (per ser accessible des de l'exterior). Crea-hi, a banda de l'usuari "usuari", un altre usuari anomenat "pepito" (amb la comanda *sudo useradd -m -s /bin/bash pepito*) i dona-li contrasenya (amb la comanda *sudo passwd pepito*). Fes també *echo "Hola" | sudo tee /home/pepito/.bashrc*

**b)** Configura aquest servidor SSH (emprant un fitxer anomenat "01-meu.conf" ubicat dins de la carpeta "/etc/ssh/sshd_config.d") per a què:

* Escolti al port 2222
* Només deixi autenticar-se (amb contrasenya) a l'usuari "pepito" i cap més
* Mostri un missatge genèric per tothom abans de loguejar-se contingut a l'arxiu "/opt/benvinguda"
*(la resta d'opcions mantindran el seu valor per defecte indicat a l'arxiu de configuració de servidor general, "/etc/ssh/sshd_config", que romandrà sense tocar)

NOTA: El nom del fitxer *.conf és important perquè indica l'ordre en què es llegirà respecte els altres fitxers que hi puguin haver dins de la carpeta "/etc/ssh/sshd_config.d" (degut a la línia *Include /etc/ssh/sshd_config.d/*.conf* existent al començament de l'arxiu "/etc/ssh/sshd_config"); els fitxer d'aquesta carpeta es llegeixen en ordre lexicoalfabètic segons el seu nom, així que un fitxer que tingui un nom començant amb un número menor que un altre es llegirà primer (i, per tant, les directives que inclogui es tindran en compte en lloc d'altres homònimes que hi podrien haver posteriorment). Per tant, a la pràctica, el més comú és afegir les directives personalitzades en un fitxer amb un nom que comenci amb un nombre petit
NOTA: En sistemes amb SELinux activat (com és el cas de Fedora per defecte), per a què el canvi de port funcioni cal, a més, executar la comanda *sudo semanage port -a -t ssh_port_t -p tcp* 2222 (o bé, si es vol fer de forma temporal, *sudo setenforce permissive*). D'altra banda, seria bo aturar el tallafocs per fer les proves més fàcils (*sudo systemctl stop firewalld*)

**c)** Després de reiniciar el servidor SSH, prova d'iniciar-hi sessió des de la màquina real (amb el client SSH estàndard que hi ha disponible) fent servir l'usuari "usuari" del servidor SSH (recorda d'indicar el número de port mitjançant el paràmetre *-p*). ¿Quin missatge veus i què passa quan intentes entrar? Ara prova-ho amb l'usuari "pepito" ¿Què passa ara?

**d)** Modifica la configuració del client SSH de la màquina real (concretament la que és exclusiva pel teu usuari local, és a dir, "~/.ssh/config" -recorda que aquest fitxer ha de tenir permisos 600) per a què:

* En el cas (només!) de connectar-se al servidor anterior...
    ... es connecti automàticament al port 2222
    ... utilitzi l'usuari "pepito" sempre per defecte
    ... es pugui escriure un àlies anomenat "miserver" associat a la IP del servidor
* En el cas de connectar-se a qualsevol altre servidor..
    ... s'executi sempre en l'ordinador local la comanda "ls %d"
*(la resta d'opcions mantindran el seu valor per defecte indicat a l'arxiu de configuració de client general per tots els usuaris locals, "/etc/ssh/ssh_config", que romandrà sense tocar)

**e)** Executa el client *ssh* indicant només l'àlies "miserver" en comptes de la IP del servidor i prou (és a dir, no indiquis ni l'usuari ni el port a connectar). ¿Què passa? Desconnecta't i ara executa el client *ssh* contra qualsevol altre servidor SSH (pots fer servir el del company). ¿Què veus en iniciar-hi sessió? Elimina finalment l'arxiu "~/.ssh/config".

**2.-a)** Elimina el contingut de l'arxiu "01-meu.conf" creat a l'exercici anterior i substitueix-lo ara pel següent:

```
LoginGraceTime 10
Match User pepito
    ForceCommand ls -a
Match Address 192.168.12.0/24
   ForceCommand ls -la
```

¿Per a què serveixen les línies anteriors? Prova-ho iniciant sessió contra el servidor (recorda de reiniciar-lo per a què el canvi tingui efecte!) tant amb l'usuari "usuari" com amb l'usuari "pepito". Finalment, esborra-les.

**b)** ¿Per a què serveix la directiva *StrictModes yes* d'un servidor SSH? (llegeix *man sshd_config* per saber-ho)

**c)** ¿Per què no funciona la comanda *ssh -o "PasswordAuthentication=no" usuari@ip.Serv.SSH* ? (pots afegir-ne el paràmetre *-v* -mode "verbós"- per veure totes les passes que realitza el client en fer la connexió i autenticació, si t'és d'ajuda)

**cII)** ¿Per a què serveix la directiva *PreferredAuthentications* del client SSH? (consulta la segona taula de teoria groga indicada als apunts anteriors, o bé *man ssh_config*, per esbrinar-ho)

**d)** Fes el necessari, tant a la configuració del servidor com a la configuració del client, per enviar la variable d'entorn HOLA amb el valor "Bon dia" de la sessió de l'usuari local de la màquina client a la sessió de "pepito" en el servidor SSH. Comprova-ho executant *echo $HOLA* tant dins del terminal local del client com dins del terminal d'una sessió SSH de l'usuari "pepito".

**dII)** ¿Quina diferència hi ha entre les directives de client *SetEnv* i *SendEnv ?*

**3.-a)** Connecta de nou al servidor SSH implementat als exercicis anteriors des del client SSH de la màquina real amb l'usuari "pepito". Un cop establerta la connexió, busca la manera de trobar, al servidor, aquesta connexió executant la comanda *ps -ef | grep -E "pepito@(pts|tty)"* i mata el procés corresponent (amb *kill -s 9 nºPID*). ¿Què li passa al client?

**b)** Sabent primer què fan les comandes *tar -cf - carpeta/* | cat - > desti.tar"* , dedueix per a què podria servir llavors executar la següent comanda: *tar -cf - ~/Imatges/* | ssh usuari@ip.Serv.SSH "cat - > fotos.tar"* Fes-ho i comprova, amb *tar -tf fotos.tar*, quin és el contingut d'aquest fitxer "tar" que ha aparegut al servidor.

**bII)** Sabent primer què fan les comandes *cat /etc/passwd | lpr* , dedueix per a què podria servir executar les següents comandes: *cat /etc/passwd | ssh nomUsuari@nomServ lpr*

**4.-a)** Configura, tal com explica la teoria, l'autenticació basada en claus per l'usuari "pepito" del servidor SSH implementat als exercicis anteriors utilitzant la màquina real com a client SSH i el teu usuari local "asix2". Aquesta autenticació ha d'incloure una "passphrase" (la que tu vulguis).

**aII)** Comprova que puguis entrar amb aquest usuari sense que et demani la contrasenya de l'usuari (no obstant, la "passphrase" te l'haurà de demanar). Un cop iniciada la sessió SSH amb "pepito", tanca-la.

**b)** Torna a iniciar sessió SSH amb "pepito" de nou: hauràs de veure que aquest segon cop ja no se't demana cap "passphrase" i pots entrar directe al servidor. ¿Per què? Pista: llegeix la darrera NOTA de la teoria

**bII)** Surt de la sessió SSH i ara executa a la màquina client la comanda *ssh-add -l* ¿Què veus? ¿Què passarà si executes *ssh-add -D* i tornes a voler iniciar una sessió SSH amb l'usuari "pepito"? ¿Per què?

**c)** Observa la sortida de la comanda *pstree* i comprova quin procés és el pare del procés *ssh-agent*. ¿Què implica això?

**d)** Comprova que amb l'usuari "usuari" encara pots entrar al servidor SSH amb contrasenya. Tot seguit, però, configura-li també l'autenticació basada en claus (ara sense "passphrase"!) i comprova que, efectivament, a partir d'ara pots entrar amb aquest usuari al servidor SSH també sense que et demani la contrasenya.

**5.-a)** Utilitza la comanda *scp* per tal de copiar un arxiu qualsevol de la màquina client (la real) a la carpeta "/home/pepito" de la màquina servidora. Comprova-ho. Esborra tot seguit aquest arxiu en local i, finalment, copia de nou amb *scp* l´arxiu acabat de pujar al servidor SSH a la ubicació del client on era originalment.

**b)** Fes el mateix amb una carpeta: "puja-la" amb tot el seu contingut dins de la carpeta "/home/pepito" remota, esborra-la del sistema local i finalment torna-la a "descarregar" allà on era.

**6.-**Configura el servidor SSH per a què només els usuaris "manolito" i "luisito", ambdós pertanyents al grup "sftpmola" i sense capacitat per obrir cap shell, puguin accedir via SFTP (amb contrasenya) a la seva carpeta personal de forma engabiada amb permisos de lectura-escriptura. Comprova-ho.

**7.-a)** Utilitza la comanda *sshfs* de la màquina real (ja hi deu estar instal·lada) per muntar dins d'una carpeta del client (anomenada "/home/asix2/hola") el contingut de la carpeta "/home/pepito" del servidor SSH (i on aquest contingut haurà d'aparèixer com a propietat de l'usuari "asix2").

**b)** Crea algun fitxer dins de la carpeta "hola" (amb *touch*, per exemple) i afegeix-li algun contingut a dins (amb *nano*, per exemple). Tot seguit observa amb *ls -l* quin és el propietari d'aquest fitxer mirant-ho en el client (dins de la carpeta "/home/asix2/hola") i també mirant-ho en el servidor (dins de la carpeta "/home/pepito"). Raona per què veus diferents propietaris en cadascuna d'aquestes vistes.

**c)** Desmunta ara la carpeta "hola" i comprova al servidor que l'arxiu creat i editat a l'apartat anterior roman

**d)** Repeteix l'apartat a) però ara sense afegir el paràmetre *-o* a la comanda *sshfs*. Tot seguit observa de nou amb *ls -l* quin és el propietari d'aquest fitxer mirant-ho en el client (dins de la carpeta "/home/asix2/hola") i també mirant-ho en el servidor (dins de la carpeta "/home/pepito"). Raona per què veus diferents propietaris en cadascuna d'aquestes vistes i respecte l'apartat b). Desmunta finalment de nou la carpeta "hola".

**8.- a)** Modifica manualment la clau pública emmagatzemada dins de l'arxiu "/home/asix2/.ssh/known_hosts" corresponent al servidor SSH que estem fent servir, i tot seguit intenta connectar-hi ¿Què passa? Després de veure-ho, contesta "No" a la pregunta que se t'haurà mostrat a la pantalla i tot seguit torna a deixar el valor de la clau editada tal com estava inicialment.

> NOTA: Es pot indicar la ruta d'un altre fitxer per a què faci de fitxer "known_hosts" (o cap, si s'indica "/dev/null", per exemple!) amb la directiva ***UserKnownHostsFile /ruta/fitxer*** Una altra directiva interessant és ***HostKeyAlias alias***, la qual serveix per indicar quin serà l'àlias que es guardarà a l'arxiu "known_hosts" (en lloc del nom DNS real i/o de la IP -segons el que valgui la directiva *CheckHostIP*-)

**b)** ¿Què passaria si, en lloc de tenir al client la directiva *StricHostKeyChecking* establerta a *ask,* estigués establerta a *yes*? (consulta la segona taula de teoria groga indicada als apunts anteriors, o bé *man ssh_config*, per esbrinar-ho)

**c)** Elimina ara la clau pública del servidor SSH que estem fent servir en aquests exercicis mitjançant la comanda *ssh-keygen*. ¿Què passarà ara quan vulguis tornar a iniciar sessió en aquest servidor? Comprova-ho

**d)** ¿Per a què serveix el paràmetre *-H* de la comanda *ssh-keygen* (consulta al seu manual) i quin sentit tindria fer-lo servir? Comprova-ho

> NOTA: Per no haver d'executar la comanda anterior cada vegada que s'afegeixi una clau pública de servidor nova, es pot indicar la directiva ***HashKnownHosts yes*** al fitxer de configuració del client ("/etc/ssh/ssh_config" o "~/.ssh/config")

**e)** ¿Per a què serveix la comanda *ssh-keyscan*? (consulta la teoria dels apunts anteriors, o bé la seva pròpia pàgina del manual per esbrinar-ho) ¿I el seu paràmetre *-H*?

**9.-a)** Arrenca, a més de la màquina virtual servidora SSH, una segona màquina virtual que tingui entorn gràfic (Fedora o Ubuntu, tant és) i instal·la-hi el paquet "waypipe".

**b)** Instal·la el mateix paquet "waypipe" a la màquina servidora SSH (cal que estigui present en ambdós extrems) i també el paquet "gnome-2048" (el qual es correspon a un videojoc gràfic; tant se val si aquesta màquina no té entorn gràfic; s'instal·laran totes les dependències per a què aquesta aplicació funcioni).

**c)** Executa, a la "segona" màquina (la que has encès a l'apartat a) d'aquest exercici) la comanda *waypipe ssh pepito@ip.serv.SSH gnome-2048*. ¿Què passa? Pista: executa la comanda *ps -C gnome-2048* tant al client SSH com al servidor SSH i dedueix què està passant a partir del que veus

NOTA: La comanda anterior en realitat és una manera ràpida (un "wrapper") d'executar les següents comandes:

*waypipe -s /tmp/sckt-local client &*
*ssh -R /tmp/sckt-remote:/tmp/sckt-local -t user@theserver waypipe -s /tmp/sckt-remote server -- weston-terminal*
*kill %1*

Waypipe al servidor funciona com un compositor Wayland (és a dir, com una "pantalla virtual" on les aplicacions gràfiques són renderitzades). La idea és establir un canal entre els dos processos waypipe (el client i el servidor) de forma que l'aplicació gràfica executada al servidor "cregui" que es renderitzarà sobre el waypipe servidor, el qual, però, en realitat, el que farà serà "traspassar" tota la informació gràfica rebuda de l'aplicació per poder ser visualitzada a través del canal establert fins el waypipe client, el qual la transmetrà, al seu torn, al compositor Wayland real del sistema client, que és sobre s'estan visualitzant realment totes les aplicacions (inclòs l'escriptori) d'aquest sistema.