

## Servidor SSH – Prática dirigidas

Fonte: <https://gnulinuxbr.wordpress.com/tag/servidor-ssh/>

Comandos intermediários do GNU/Linux

## Servidor SSH – Prática dirigidas



### Configurando o Servidor(básico)

1 – Instale o servidor de SSH

```
Terminal
Arquivo  Editar  Ver  Terminal  Abas  Ajuda
server-arq:/home/ecouto# aptitude install ssh
Lendo listas de pacotes... Pronto
Construindo árvore de dependências
Lendo informação de estado... Pronto
Lendo informações estendidas de estado
Inicializando estados de pacotes... Pronto
Lendo descrições de tarefas... Pronto
Nenhum pacote será instalado, atualizado ou removido.
0 pacotes atualizados, 0 novos instalados, 0 a serem removidos e 0 não atualizados.
É preciso obter 0B de arquivos. Depois do desempacotamento, 0B serão usados.
Escrevendo informações estendidas de estado... Pronto
E: Diretório '/var/log/apt/' está faltando
Lendo listas de pacotes... Pronto
Construindo árvore de dependências
Lendo informação de estado... Pronto
Lendo informações estendidas de estado
Inicializando estados de pacotes... Pronto
Lendo descrições de tarefas... Pronto

server-arq:/home/ecouto#
```

2 – edite o arquivo sshd\_conf

**Port 22**

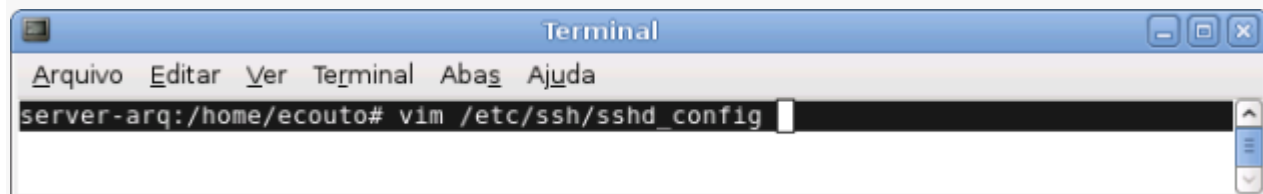
**Protocol 2**

**LoginGraceTime 60**

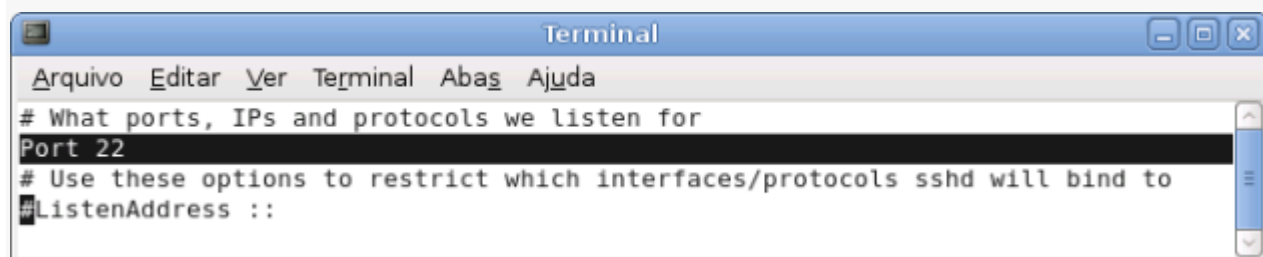
**PermitRootLogin no**

**PubkeyAuthentication yes**

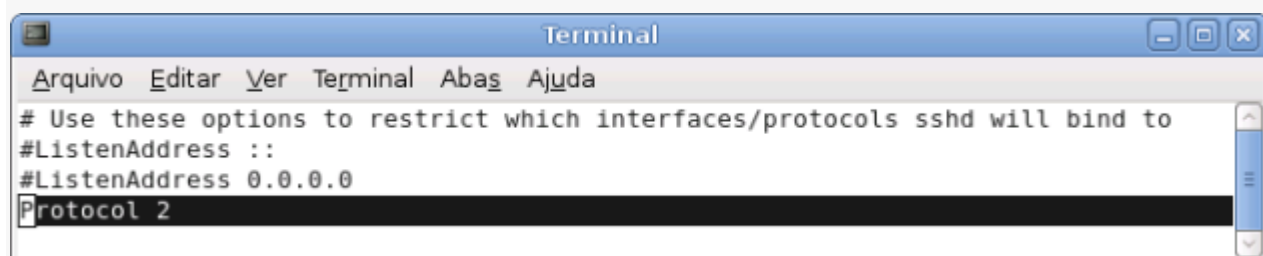
**PermitEmptyPasswords no**



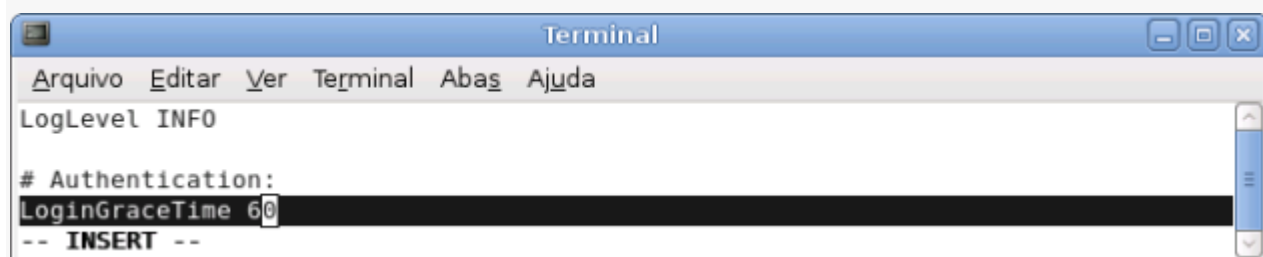
```
Terminal
Arquivo  Editar  Ver  Terminal  Abas  Ajuda
server-arq:/home/ecouto# vim /etc/ssh/sshd_config
```



```
Terminal
Arquivo  Editar  Ver  Terminal  Abas  Ajuda
# What ports, IPs and protocols we listen for
Port 22
# Use these options to restrict which interfaces/protocols sshd will bind to
#ListenAddress ::
```

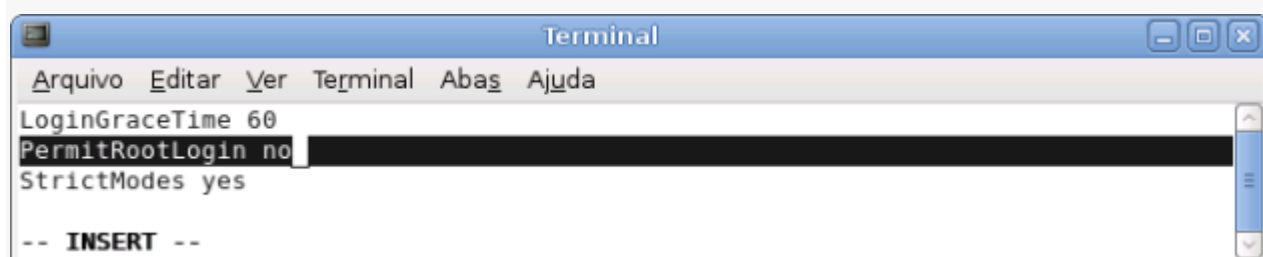


```
Terminal
Arquivo  Editar  Ver  Terminal  Abas  Ajuda
# Use these options to restrict which interfaces/protocols sshd will bind to
#ListenAddress ::
#ListenAddress 0.0.0.0
Protocol 2
```



```
Terminal
Arquivo  Editar  Ver  Terminal  Abas  Ajuda
LogLevel INFO

# Authentication:
LoginGraceTime 60
-- INSERT --
```



```
Terminal
Arquivo  Editar  Ver  Terminal  Abas  Ajuda
LoginGraceTime 60
PermitRootLogin no
StrictModes yes

-- INSERT --
```

```
Terminal
Arquivo  Editar  Ver  Terminal  Abas  Ajuda

RSAAuthentication yes
PubkeyAuthentication yes
#AuthorizedKeysFile      %h/.ssh/authorized_keys
-- INSERT --
```

```
Terminal
Arquivo  Editar  Ver  Terminal  Abas  Ajuda

# To enable empty passwords, change to yes (NOT RECOMMENDED)
PermitEmptyPasswords no

# Change to yes to enable challenge-response passwords (beware issues with
-- INSERT --
```

3 – Após realizar as devidas alterações, irei subir o daemon do servidor SSH.

```
Terminal
Arquivo  Editar  Ver  Terminal  Abas  Ajuda

server-arq:/# /etc/init.d/ssh stop
Stopping OpenBSD Secure Shell server: sshd.
server-arq:/# /etc/init.d/ssh start
Starting OpenBSD Secure Shell server: sshd.
server-arq:/# █
```

4 – Determinando qual é a porta utilizada pelo SSH

```
Terminal
Arquivo  Editar  Ver  Terminal  Abas  Ajuda

server-arq:/# grep ssh /etc/services
ssh          22/tcp      # SSH Remote Login Protocol
ssh          22/udp
server-arq:/# █
```

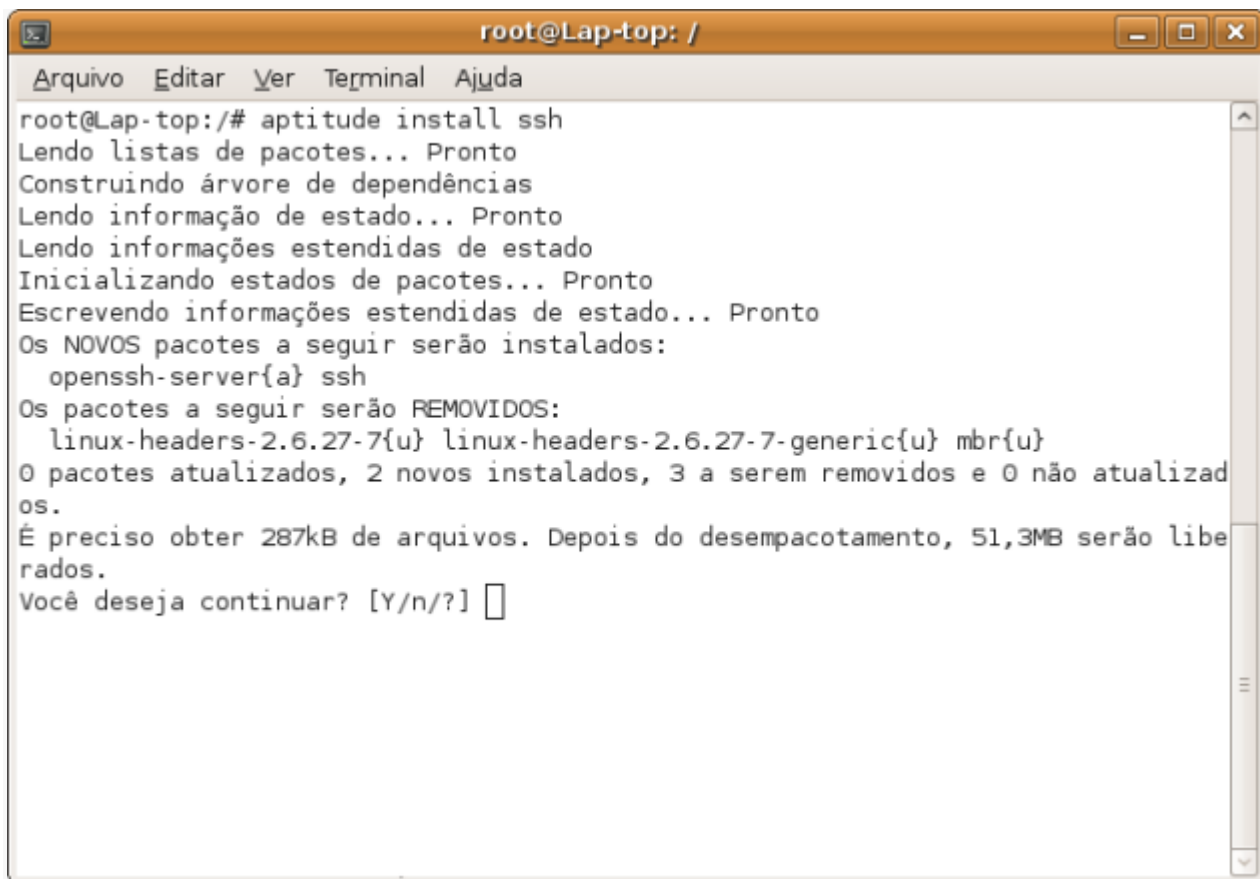
5 – Verifiquei se a porta 22 está aberta e escutando

```
Terminal
Arquivo  Editar  Ver  Terminal  Abas  Ajuda

server-arq:/# netstat -ltan | grep 22
tcp6        0      0 :::22          :::*           OUÇA
server-arq:/# fuser -v 22/tcp
                USER      PID ACCESS COMMAND
22/tcp:         root      2723 F....  lshd
server-arq:/# █
```

## Configurando o Cliente (básico)

1 – Instale o servidor de SSH



```
root@Lap-top: /
Arquivo  Editar  Ver  Terminal  Ajuda
root@Lap-top:/# aptitude install ssh
Lendo listas de pacotes... Pronto
Construindo árvore de dependências
Lendo informação de estado... Pronto
Lendo informações estendidas de estado
Inicializando estados de pacotes... Pronto
Escrevendo informações estendidas de estado... Pronto
Os NOVOS pacotes a seguir serão instalados:
  openssh-server{a} ssh
Os pacotes a seguir serão REMOVIDOS:
  linux-headers-2.6.27-7{u} linux-headers-2.6.27-7-generic{u} mbr{u}
0 pacotes atualizados, 2 novos instalados, 3 a serem removidos e 0 não atualizados.
É preciso obter 287kB de arquivos. Depois do desempacotamento, 51,3MB serão liberados.
Você deseja continuar? [Y/n/?] ☐
```

2 – edite o arquivo ssh\_conf.

**host 10.0.0.2 – Este host é referente ao meu servidor ssh**

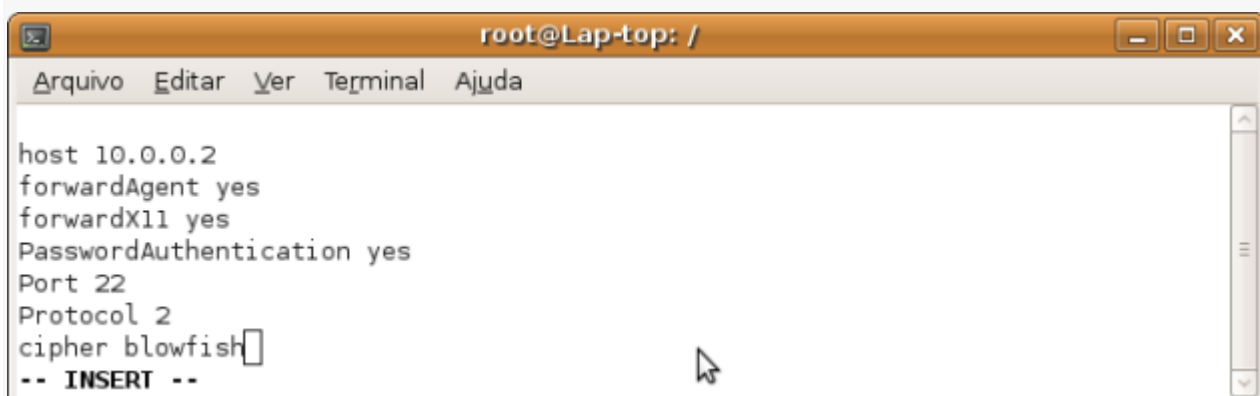
**ForwardAgent yes**

**ForwardX11 yes**

**Port 22**

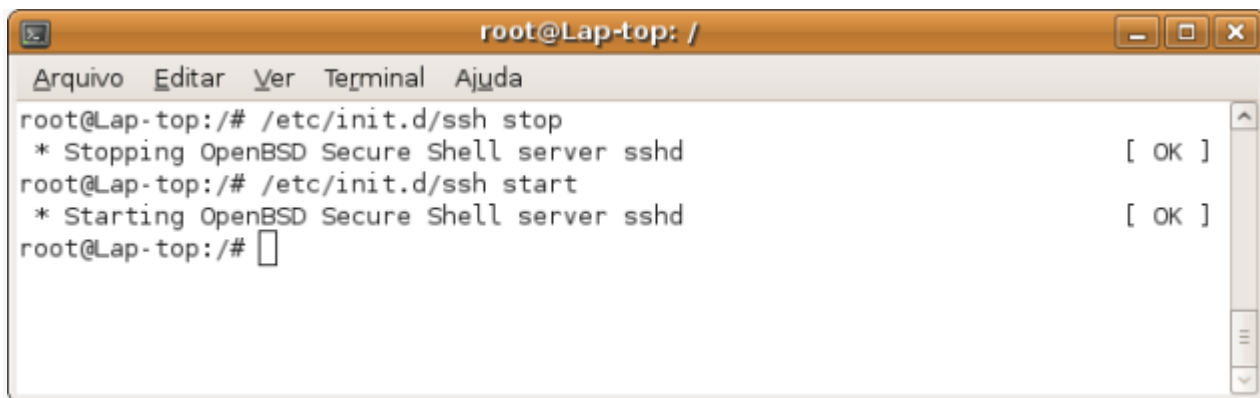
**Protocol 2**

**cipher blowfish**



```
root@Lap-top: /
Arquivo  Editar  Ver  Terminal  Ajuda
host 10.0.0.2
forwardAgent yes
forwardX11 yes
PasswordAuthentication yes
Port 22
Protocol 2
cipher blowfish ☐
-- INSERT --
```

3 – Após a conclusão das alterações, irei subir novamente o daemon do SSH



```
root@Lap-top: /
Arquivo  Editar  Ver  Terminal  Ajuda
root@Lap-top:/# /etc/init.d/ssh stop
* Stopping OpenBSD Secure Shell server sshd [ OK ]
root@Lap-top:/# /etc/init.d/ssh start
* Starting OpenBSD Secure Shell server sshd [ OK ]
root@Lap-top:/#
```

4 – Acessando o servidor server-arq, com o usuário ecouto



```
ecouto@server-arq: ~
Arquivo  Editar  Ver  Terminal  Ajuda
root@Lap-top:/# ssh ecouto@10.0.0.4
ecouto@10.0.0.4's password:
ecouto@server-arq:~$ hostname
server-arq
ecouto@server-arq:~$
```

[Comandos Intermediários](#), [Servidor SSH - Práticas dirigidas](#) [Servidor SSH](#)

## Servidor SSH – Transferindo usando scp

### comandos intermediário do GNU/Linux

#### Comando scp

Cópia de arquivos ou diretórios via rede

Enviar arquivos para outra máquina da rede local ou internet, use no formato;

**scp arquivo-a-copiar userdestino@ipdestino:~/**

Após o IP acrescentei dois pontos, barra normal e caminho completo, para copiar para o home do usuário destino, eu posso substituir o caminho do home do usuário por "til", segue alguns exemplos de linha de comando.

Copiar arqteste.txt para o home do usuário destino:

**scp arqteste.txt ecouto@10.0.0.2:~/**

Copiar teste.txt para o sub-diretório do home do usuário, /home/ecouto/copias/faq

**scp teste.txt ecouto@10.0.0.2:~/copias/faq**

Observe que foi substituído o /home/ecouto por "til", o comando acima pode ser substituído por este sem o til;

**scp teste.txt ecouto@10.0.0.2:/home/ecouto/copias/faq**

Copiar todos os arquivos e sub-diretórios a partir do prompt de comando local para o home do usuário ecouto no destino.

```
scp -r * ecouto@10.0.0.2:~/
```

Copiar para um sub-diretorio especifico no micro de destino, não use (~) e informe o caminho:

```
scp -r * ecouto@10.0.0.2:/home/copias/copiadoserv
```

Salvar no destino com outro nome, também acrescentar a data e hora de criação:

```
scp -r /home/samba ecouto@10.0.0.2:/home/ecouto/servsamba-`date +%d.%b.%Y-%H-%M`
```

Resultado; caminho e nome do arquivo no destino tem este formato: **/home/ecouto/servsamba-20.Out.2009-18-20**

Atualizar paginas do Apache, requer alteração de permissão para usuário gravar.

```
scp -r * ecouto@10.0.0.2:/srv/www/default/html/ecouto
```

Download, buscar arquivos e diretórios em outra maquina, copiar da maquina remota para maquina local, pode fazer a conexão e a partir da linha de comando remota executar outra linha de comando para enviar arquivos para minha maquina, isto implica em fazer a conexão para depois enviar os arquivos, caso eu queira somente transferir os arquivos para minha maquina, então não precisarei fazer a conexão, em uma tacada só, a solução mais simples é esta;

```
scp -r user-remoto@maquina_remota:~/sub-diretório/diretorio-download .
```

```
scp user-remoto@maquina_remota:~/sub-diretório/arquivo-download .
```

O ponto no final da linha indica para salvar no diretório atual.

```
scp -r ecouto@10.0.0.5:~/acesso-livre .
```

Onde; ecouto é usuário da maquina remota, 10.0.0.5 é o IP da maquina remota, acesso-livre é o diretório na maquina remota que desejo copiar para a maquina local, observe ponto no final, indica para salvar a copia no diretorio local, mesmo diretório que executou a linha de comando, no exemplo acima -r indica para copiar o diretório acesso-livre, para copiar arquivos usei este formato.

```
scp ecouto@10.0.0.5:~/teste.txt .
```

Para salvar em outro local, substitui o ponto pelo caminho na maquina local, exemplo para salvar na maquina local em /home/ecouto/tmp

```
scp ecouto@10.0.0.5:~/teste.txt /home/ecouto/tmp
```

```
scp -r ecouto@10.0.0.5:~/acesso-livre /home/ecouto/tmp
```

```
scp -r ecouto@10.0.0.5:/home/ecouto/acesso-livre /home/ecouto/tmp
```

Este recurso é muito prático na copia de configuração de outra maquina da rede ou até mesmo via internet, exemplo de copia do script de firewall de outra maquina.

```
scp ecouto@10.0.0.9:/etc/rc.d/firewall.sh /etc/rc.d
```

Copiar o diretório squid com os arquivos de acl,

```
scp -r ecouto@10.0.0.11:/etc/squid /etc/
```

Dependendo do arquivo ou diretório, como nos exemplos acima, foi preciso ser root na maquina local para gravar e permissão de leitura na maquina remota.

Tem um espaço separador entre o nome do arquivo ou diretório remoto e o caminho local.

Dois comandos na mesma linha, usei ; para indicar o proximo comando, mesmo assim tenho que digitar a senha duas vezes.

Exemplo para download e apagar. Prático para transferir arquivos de backup, apagar depois da cópia.

```
scp ecouto@10.0.0.5:~/teste.txt . ; ssh ecouto@10.0.0.5 '(rm ~/teste.txt)'
```

O resultado é mesmo que digitar em duas linhas.

```
scp ecouto@10.0.0.5:~/teste.txt .  
ssh ecouto@10.0.0.5 '(rm ~/teste.txt)'
```

Quando for o mesmo usuário nas duas maquinas, pode omitir o usuário, neste formato.

```
scp 10.0.0.5:~/teste.txt . ; ssh 10.0.0.5 '(rm ~/teste.txt)'
```

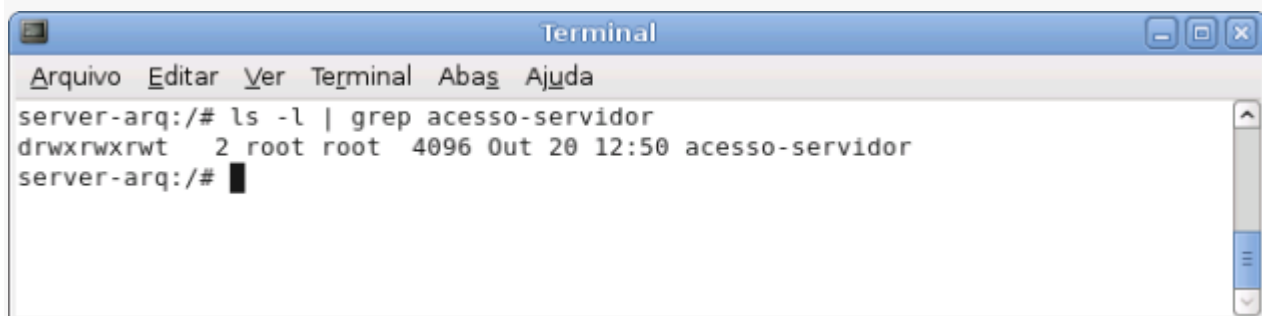
Quando a maquina remota está atrás de firewall, poderei fazer as conexões de maquina em maquina, copiar de uma para outra até disponibilizar em um local com acesso direto ou simplesmente acessar o firewall remoto e incluir uma regra que redirecione para a maquina que desejo alcançar.

Exemplo de comando em uma estação para acessar outra estação atrás de outra rede que tem firewall redirecionando a porta 2222 para a estação remota, baixar para o diretório local o diretório remoto /etc/postfix.

```
scp -P 2222 -r ecouto@10.0.0.8:/etc/postfix .
```

## Práticas dirigidas

1 – /home/root/acesso-servidor



2 – /home/ecouto/acesso-cliente

```
root@Lap-top: /home/ecouto
Arquivo  Editar  Ver  Terminal  Ajuda
root@Lap-top:/home/ecouto# ls -l | grep acesso-cliente
drwxrwxrwt 2 root  root   4096 2009-10-20 12:01 acesso-cliente
root@Lap-top:/home/ecouto#
```

3 – Copiando do cliente o backup-passwd que está no diretório home do usuário ecouto.

```
Terminal
Arquivo  Editar  Ver  Terminal  Abas  Ajuda
server-arq:/# scp ecouto@10.0.0.5:/home/ecouto/acesso-cliente/backup-passwd-laptop .
ecouto@10.0.0.5's password:
backup-passwd-laptop                                100% 1586    1.6KB/s   00:00
server-arq:/#
```

4 – copiando recursivamente e armazenando os arquivos no diretório atual

```
Terminal
Arquivo  Editar  Ver  Terminal  Abas  Ajuda
server-arq:/# scp -r ecouto@10.0.0.5:/home/ecouto/acesso-cliente .
ecouto@10.0.0.5's password:
teste.txt                                           100%    0    0.0KB/s   00:00
backup-passwd-laptop                               100% 1586    1.6KB/s   00:00
server-arq:/#
```

5 – Copiando recursivamente e especificando o caminho completo para onde serão armazenado os arquivos

```
Terminal
Arquivo  Editar  Ver  Terminal  Abas  Ajuda
server-arq:/# scp -r ecouto@10.0.0.5:/home/ecouto/acesso-cliente acesso-servidor/
ecouto@10.0.0.5's password:
teste.txt                                           100%    0    0.0KB/s   00:00
backup-passwd-laptop                               100% 1586    1.6KB/s   00:00
server-arq:/#
```

6 – Copiando recursivamente e especificando o caminho completo e também a data da criação dos arquivos copiados

```
Terminal
Arquivo  Editar  Ver  Terminal  Abas  Ajuda
server-arq:/# scp -r ecouto@10.0.0.5:/home/ecouto/acesso-cliente acesso-servidor/`date +%d.%b.%Y-%H-%M`
ecouto@10.0.0.5's password:
teste.txt                                           100%    0    0.0KB/s   00:00
backup-passwd-laptop                               100% 1586    1.6KB/s   00:00
server-arq:/#
```



## Servidor SSH – Transferindo arquivos SFTP

### Comandos intermediários do GNU/Linux

#### Transferindo arquivos via SFTP

O SSH é um verdadeiro canivete suíço. Além de permitir rodar aplicativos e fazer toda a administração de um servidor remotamente, ele também pode ser usado para transferir arquivos. A forma mais básica de fazer isso é usar o sftp, um pequeno utilitário que faz parte do pacote padrão.

Ele oferece uma interface similar à dos antigos programas de FTP de modo texto, mas todos os arquivos transferidos através dele trafegam através de um túnel encriptado, criado através do SSH. Na prática, temos uma espécie de VPN temporária, criada no momento em que é efetuada a conexão. A melhor parte é que o próprio SSH cuida de tudo, não é necessário instalar nenhum programa adicional.

Para se conectar a um servidor usando o sftp, o comando é:

```
$ sftp usuario@servidor
```

Se o servidor ssh na outra ponta estiver configurado para escutar em uma porta diferente da 22, é preciso indicar a porta no comando, incluindo o parâmetro “-o port=”, como em:

```
$ sftp -o port=2222 ecouto@gnulinuxbr
```

A partir daí, terei terei o prompt do sftp. Com o comando “put” usei para dar upload de um arquivo e “get” para baixar um arquivo do servidor para a pasta local. Para navegar entre as pastas do servidor, use os comandos “cd pasta/” (para acessar a pasta), “cd ..” (para subir um diretório), “ls” (para listar os arquivos) e “pwd” (para ver em qual diretório está). Veja um exemplo:

```
ecouto@sunny-days:~$ sftp -o port=2222 ecouto@gnulinuxbr.wordpress.com.br
```

```
Connecting to gnulinuxbr.wordpress.com.br...
```

```
Password: *****
```

```
sftp> ls
```

```
gnulinuxbr.wordpress.com.tar.gz www
```

```
sftp> get gnulinuxbr.wordpress.com.tar.gz
```

```
Fetching /home/ecouto/gnulinuxbr.wordpress.tar.gz tognulinuxbr.wordpress.com.tar.gz  
/home/ecouto/gnulinuxbr.wordpress.com.tar.gz 100% 523MB 945.1KB/s 09:13
```

```
sftp> put backupdb.gz
```

```
Uploading backupdb.gz to /home/ecouto/ backupdb.gz  
backupdb.gz 100% 98MB 967KB/s 01:43
```

```
sftp> pwd
```

```
Remote working directory: /home/ecouto
```

Existem ainda os comandos “lcd” (local cd), “lls” (local ls), “lmkdir” (local mkdir) e “lpwd” (local pwd), que permitem mudar o diretório local. Digamos, por exemplo, que eu queira está atualmente no diretório “/mnt/arquivos”. Ao abrir a conexão via

sftp, tudo que eu baixar será colocado automaticamente neste diretório. Mas, digamos que eu queira baixar um determinado arquivo para o diretório `"/home/joao"`. Eu usaria, então, o comando `"lcd /home/joao"` para mudar o diretório local e depois o `"get arquivo"` para baixá-lo já na pasta correta. Na hora de dar upload de um arquivo é a mesma coisa. Você pode usar o `"lls"` para listar os arquivos no diretório local e depois o `"put arquivo"` para dar upload.

Naturalmente, existem meios mais práticos de transferir os arquivos, usando programas gráficos que suportam o sftp. O dois mais usados são o próprio Konqueror (no KDE) e o Nautilus (no Gnome), que além de serem gerenciadores de arquivos, suportam o uso de diversos protocolos de transferência, incluindo o sftp.

A grande limitação do sftp (e das interfaces gráficas para ele) é que ele exige intervenção manual e, por isso, não é adequado para uso em scripts para realizar transferências automatizadas (como no caso de um script de backup). Para isso, temos o **"scp"**, um cliente ainda mais primitivo, que permite especificar em uma única linha o login e endereço do servidor, junto com o arquivo que será transferido.

A sintaxe do scp é: `"scp arquivo_local login@servidor:pasta_remota"`, como em:

```
$ scp /home/ecouto/arquivo.tar ecouto@gnulinuxbr.wordpress.com:/var/www/download
```

Eu posso adicionar também as opções **"-p"** (que preserva as permissões de acesso além das datas de criação e modificação do arquivo original), **"-r"** (que permite copiar pastas, recursivamente), **"-v"** (verbose, onde são mostradas todas as mensagens) e **"-C"** (que ativa a compressão dos dados, ajudando muito na hora de transferir grandes arquivos via Internet). Nesse caso, o comando ficaria:

```
$ scp -prvC /home/ecouto/arquivo.tar ecouto@gnulinuxbr.wordpress.br:/var/www/download
```

Ao incluir a opção **"-r"**, eu posso especificar diretamente uma pasta no primeiro parâmetro, o que faz com que todo o seu conteúdo seja transferido. Esta opção é interessante para backups.

O **SSH** pode ser ainda usado como **"meio de transporte"** por outros programas, como no caso do rsync, um utilitário que permite sincronizar uma pasta local com uma pasta do servidor. Ele é capaz de fazer uma cópia diferencial, transferindo apenas os trechos dos arquivos que foram modificados o que o torna um utilitário quase que ideal para backup de pastas com um grande volume de arquivos. Ele é capaz inclusive de consertar arquivos danificados e dar upload de atualizações, enviando apenas as partes dos arquivos que forem diferentes, o que torna a transferência muito mais rápida.

O uso básico do **rsync**, para sincronizar duas pastas locais, seria **"rsync -a origem/ destino/"**. A pasta destino poderia ser um segundo HD, um cartão de memória ou um compartilhamento de rede, por exemplo. Usado dessa forma, o rsync serve apenas para fazer backups locais, mas, ao combiná-lo com o SSH, abrimos uma nova gama de possibilidades.

Para usar o rsync via SSH, o comando acaba sendo bem mais complexo, mas o resultado é bem interessante. Ele vai apenas atualizar as partes dos arquivos remotos que foram modificados, sem dar upload dos arquivos inteiros novamente, como muitos programas de backup fariam.

Para sincronizar a pasta local `"/home/gnulinuxbr"` com a pasta remota `"/backup"`, no servidor `"gnulinuxbr.wordpress.com"` (onde seria feito um backup dos arquivos locais), usando o login `"ecouto"`, por exemplo, o comando seria:

```
$ rsync -av --rsh="ssh -l ecouto" /home/ecouto/ ecouto@gnulinuxbr.wordpress.com:/backup
```

Para recuperar posteriormente o backup no caso de um desastre, baixando os arquivos salvos no servidor, bastaria inverter a ordem dos diretórios no comando, como em:

```
$ rsync -av --rsh="ssh -l ecouto" ecouto@gnulinubr.wordpress.com:/backup /home/ecouto/
```

No primeiro comando os arquivos da pasta `"/home/ecouto"` vão para a pasta `/backup` do servidor e no segundo eles são recuperados, sobrescrevendo os arquivos locais. A parte mais significativa deste comando é o parâmetro `--rsh="ssh -l ecouto"`, que diz para o **rsync** usar um programa externo (**SSH**) para fazer o trabalho.

Uma observação é que usando apenas os parâmetros `"-av"`, o **rsync** apenas atualiza e grava novos arquivos na pasta do servidor, sem remover arquivos que tenham sido deletados na pasta local. Por um lado isso é bom, pois permite recuperar arquivos deletados acidentalmente, mas por outro pode causar confusão. Se eu preferir que os arquivos que não existem mais sejam deletados ao atualizar o backup, adicionarei o parâmetro `"--delete"`, como em:

```
$ rsync -av --delete --rsh="ssh -l ecouto" /home/ecouto/ ecouto@gnulinuxbr.wordpress.com:/backup
```

O **rsync** não vem instalado por padrão na maioria das distribuições, mas pode ser instalado rapidamente usando o gerenciador de pacotes, como em:

```
# apt-get install rsync
```

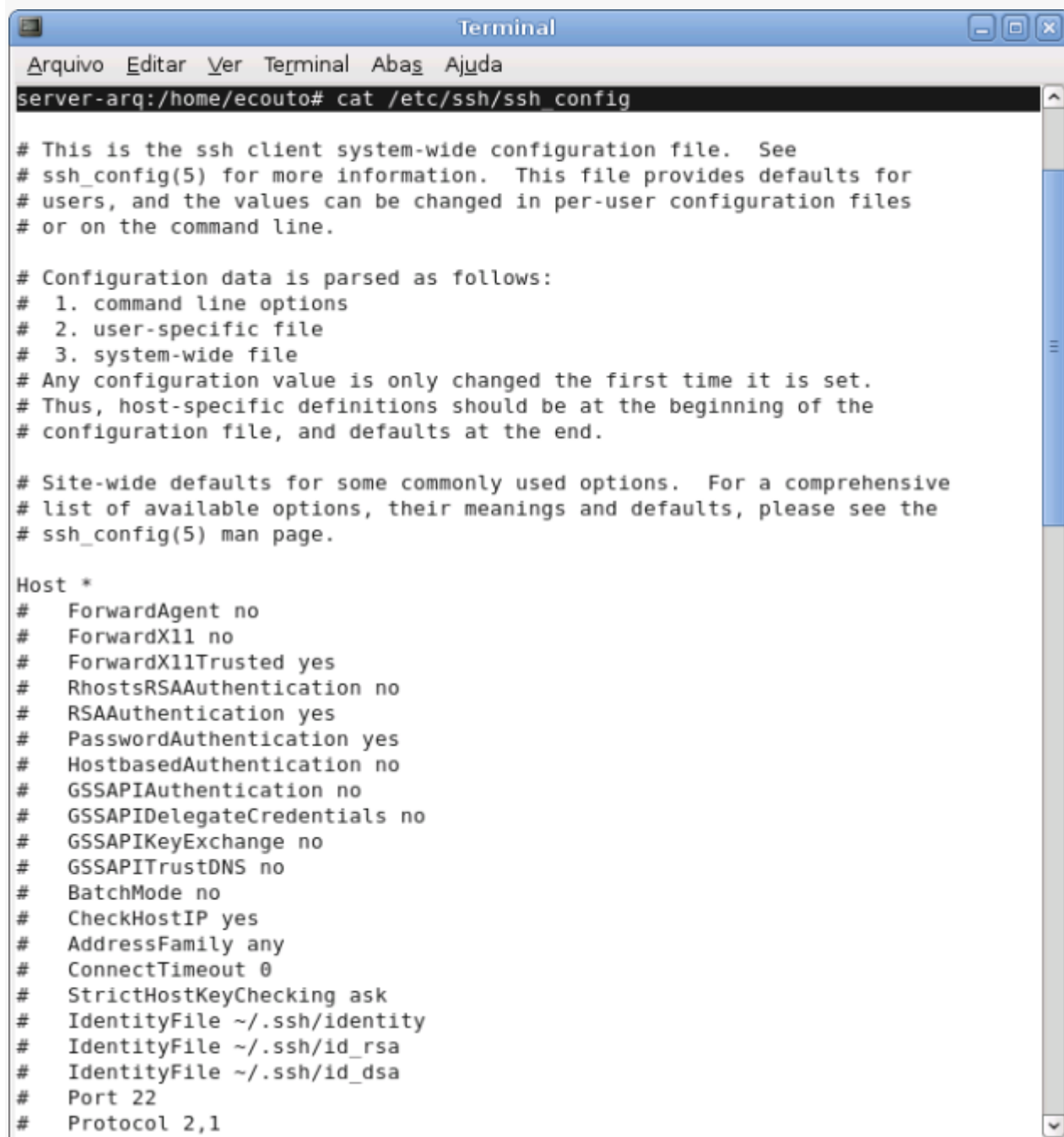
[Comandos Intermediários](#), [Servidor SSH - Transferindo arquivos usando SFTP](#) [Servidor SSH](#)

## Servidor SSH – Resumo dos seus arquivos de configuração

### Comandos intermediários do GNU/Linux

#### Servidor SSH – Resumo dos seus arquivos de configuração

A configuração do servidor, independentemente da distribuição usada, vai no arquivo `"/etc/ssh/sshd_config"`, enquanto a configuração do cliente vai no `"/etc/ssh/ssh_config"`. Note que muda apenas um `"d"` entre os dois.

A terminal window titled "Terminal" with a menu bar containing "Arquivo", "Editar", "Ver", "Terminal", "Abas", and "Ajuda". The command prompt shows the user is in the directory "/home/ecouto" and has executed the command "cat /etc/ssh/ssh\_config". The output displays the configuration file's content, which includes comments about the file's purpose, parsing rules, and site-wide defaults, followed by a list of configuration options for the "Host \*" section.

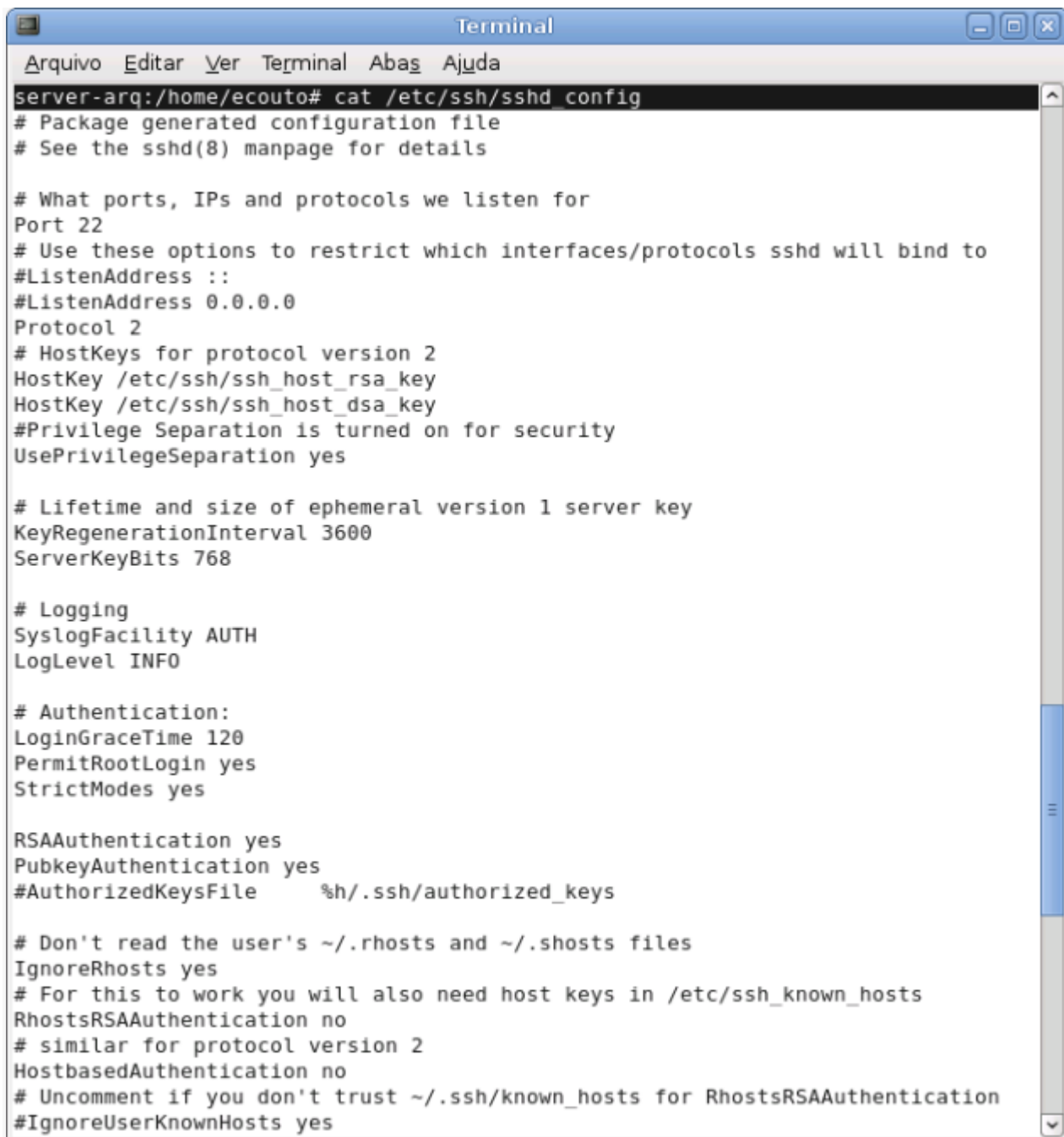
```
server-arq:/home/ecouto# cat /etc/ssh/ssh_config

# This is the ssh client system-wide configuration file.  See
# ssh_config(5) for more information.  This file provides defaults for
# users, and the values can be changed in per-user configuration files
# or on the command line.

# Configuration data is parsed as follows:
# 1. command line options
# 2. user-specific file
# 3. system-wide file
# Any configuration value is only changed the first time it is set.
# Thus, host-specific definitions should be at the beginning of the
# configuration file, and defaults at the end.

# Site-wide defaults for some commonly used options.  For a comprehensive
# list of available options, their meanings and defaults, please see the
# ssh_config(5) man page.

Host *
#   ForwardAgent no
#   ForwardX11 no
#   ForwardX11Trusted yes
#   RhostsRSAAuthentication no
#   RSAAuthentication yes
#   PasswordAuthentication yes
#   HostbasedAuthentication no
#   GSSAPIAuthentication no
#   GSSAPIDelegateCredentials no
#   GSSAPIKeyExchange no
#   GSSAPITrustDNS no
#   BatchMode no
#   CheckHostIP yes
#   AddressFamily any
#   ConnectTimeout 0
#   StrictHostKeyChecking ask
#   IdentityFile ~/.ssh/identity
#   IdentityFile ~/.ssh/id_rsa
#   IdentityFile ~/.ssh/id_dsa
#   Port 22
#   Protocol 2,1
```

A screenshot of a terminal window titled "Terminal". The window has a menu bar with "Arquivo", "Editar", "Ver", "Terminal", "Abas", and "Ajuda". The terminal content shows the command "cat /etc/ssh/sshd\_config" being executed, displaying the configuration file's contents. The configuration includes comments and settings for ports, protocols, host keys, privilege separation, key regeneration, logging, authentication, and host-based authentication.

```
server-arq:/home/ecouto# cat /etc/ssh/sshd_config
# Package generated configuration file
# See the sshd(8) manpage for details

# What ports, IPs and protocols we listen for
Port 22
# Use these options to restrict which interfaces/protocols sshd will bind to
#ListenAddress ::
#ListenAddress 0.0.0.0
Protocol 2
# HostKeys for protocol version 2
HostKey /etc/ssh/ssh_host_rsa_key
HostKey /etc/ssh/ssh_host_dsa_key
#Privilege Separation is turned on for security
UsePrivilegeSeparation yes

# Lifetime and size of ephemeral version 1 server key
KeyRegenerationInterval 3600
ServerKeyBits 768

# Logging
SyslogFacility AUTH
LogLevel INFO

# Authentication:
LoginGraceTime 120
PermitRootLogin yes
StrictModes yes

RSAAuthentication yes
PubkeyAuthentication yes
#AuthorizedKeysFile      %h/.ssh/authorized_keys

# Don't read the user's ~/.rhosts and ~/.shosts files
IgnoreRhosts yes
# For this to work you will also need host keys in /etc/ssh_known_hosts
RhostsRSAAuthentication no
# similar for protocol version 2
HostbasedAuthentication no
# Uncomment if you don't trust ~/.ssh/known_hosts for RhostsRSAAuthentication
#IgnoreUserKnownHosts yes
```

Outra observação é que além do OpenSSH, que abordo aqui, existem outras versões do SSH, como o Tectia (uma versão comercial, disponível no <http://ssh.com>) e o SunSSH que, embora conservem diferenças no funcionamento e na configuração, são compatíveis entre si. O SSH é, na verdade, um protocolo aberto e não o nome de uma solução específica.

O uso básico do SSH é bastante simples, já que basta usar o comando **"ssh login@servidor"** para acessar o servidor remoto e, a partir daí, rodar os comandos desejados. Entretanto, essa é apenas a ponta do iceberg. O SSH é tão rico em funções que até mesmo os administradores mais experientes raramente usam mais do que um punhado das funções disponíveis.

Ao ser ativado, o padrão do servidor SSH é permitir acesso usando qualquer uma das contas de usuário cadastradas no sistema, pedindo apenas a senha de acesso. Para acessar o servidor **"10.0.0.2"**, usando o login **"ecouto"**, por exemplo, o comando seria:

```
$ ssh ecouto@10.0.0.2
```

Em vez de usar a arroba, você pode também especificar o login usando o parâmetro **"-l"** (de login), como em:

```
$ ssh -l ecouto 10.0.0.2
```

Você pode também acessar o servidor usando o domínio, como em:

```
$ ssh ecouto@gnulinux.wordpress.com
```

Caso eu omita o nome do usuário, o SSH presume que você quer acessar usando o mesmo login que está utilizando na máquina local. Se eu estiver logado como "joao", ele tentará fazer login usando uma conta "joao" no servidor remoto. Naturalmente, só funciona caso você use o mesmo login em ambas as máquinas.

Ao acessar micros dentro da rede local, poderei também chamá-los pelo nome, como em "**ssh ecouto@servidor**". Neste caso, irei precisar primeiro editar o arquivo "/etc/hosts" (no cliente), incluindo os números de IP das máquinas e os nomes correspondentes. O formato deste arquivo é bem simples, basta fornecer o IP e o nome da máquina correspondente, um por linha, como em:

**127.0.0.1 localhost**

**10.0.0.3 arq-server**

**10.0.0.2 servidor**

A configuração do arquivo "**/etc/hosts**" vale apenas para a própria máquina, ele nada mais é do que um arquivo com aliases. Se eu quiser que os apelidos sejam válidos também para as demais máquinas da rede, a melhor opção é configurar um servidor DNS de rede local.

Voltando à configuração do SSH, irei a algumas opções importantes dentro da configuração do cliente SSH, que vão no arquivo "**/etc/ssh/ssh\_config**":

Aplicativos gráficos: Além de oferecer acesso via linha de comando, o SSH permite rodar aplicativos gráficos remotamente (**X11 forwarding**). Muitas distribuições trazem o recurso desabilitado por padrão. Nestes casos, edite o arquivo "**/etc/ssh/ssh\_config**" (no cliente) e substitua a linha "ForwardX11 no" por:

**ForwardX11 yes**

Outra opção é adicionar o parâmetro "-X" ao se conectar, como em "**ssh -X mario@10.0.0.2**". A partir daí, poderei chamar os aplicativos gráficos normalmente, como se estivesse usando um terminal local.

O maior problema com o uso de aplicativos gráficos via SSH é que ele só funciona satisfatoriamente dentro da rede local. Via Internet os aplicativos gráficos ficam realmente muito lentos (mesmo em uma conexão de 4 ou 8 megabits), pois o protocolo do X é otimizado para uso local, com uso intensivo de pacotes de retorno e sem nenhum tipo de cache. Isso faz com que muitos administradores desabilitem o X11 forwarding no próprio servidor.

Para rodar aplicativos gráficos de forma segura via Internet, a melhor solução é usar o NX Server. Ele é um sistema de acesso remoto baseado no SSH, que utiliza um protocolo bastante otimizado. Nele você tem um desktop completo (similar ao VNC), mas com um desempenho muito superior, mesmo em conexões via modem.

Compressão: No caso de servidores acessíveis via Internet, você pode reduzir um pouco o consumo de banda ativando a compressão de dados via **gzip**, o que é feito adicionado a linha:

**Compression = yes**

Também posso ativar a compressão adicionando a opção "**-C**" na hora de se conectar. Quase todas as opções do cliente SSH podem ser especificadas tanto no arquivo, quanto via linha de comando.

Keep Alive: Outro problema comum relacionado ao SSH é a conexão ser fechada pelo servidor depois de alguns minutos de inatividade. Em muitas eu posso querer manter a conexão aberta por longos períodos, sem precisar ficar dando um "ls" ou um "pwd" a cada dois minutos para manter a conexão aberta.

ambém posso evitar o problema fazendo com que o próprio cliente mande pacotes periodicamente a fim de manter a conexão aberta. Para ativar o recurso, adicionei a opção "ServerAliveInterval" no arquivo, especificando o intervalo entre o envio dos pacotes (em segundos):

### **ServerAliveInterval 120**

Este é um exemplo de arquivo `/etc/ssh/ssh_config`, configurado com as opções que mostrei até aqui (excluindo os comentários):

**ForwardX11 yes**

**Compression = yes**

**Port 22**

**ServerAliveInterval 120**

Como em outros arquivos de configuração, eu não preciso preocupar em incluir cada uma das opções disponíveis, pois o SSH simplesmente usa os valores default para as opções não incluídas no arquivo. Com isso, irei precisar me preocupar apenas com as opções que já conheço, omitindo as demais.

Verificação do servidor: Como parte das verificações de segurança, o SSH utiliza também um sistema baseado em chaves assimétricas para verificar a identidade do servidor. O servidor possui uma chave pública, que é enviada ao cliente na primeira conexão. As identificações de todos os servidores conhecidos ficam armazenadas no arquivo `ssh/known_hosts`, dentro do seu diretório home. Sempre que eu me conecto daí em diante, o cliente SSH envia um "desafio" ao servidor, uma frase encriptada usando a chave pública (do servidor), que só pode ser descoberta usando a chave privada.

Isso previne um tipo de ataque muito comum chamado **"man in the middle"** (que poderia ser traduzido para "intermediário", ou "impostor"), em que alguém simplesmente substitui o servidor por outra máquina, usando o mesmo IP, ou sabota o servidor DNS da rede (ou do provedor de acesso) de forma que ele entregue um IP forjado quando eu tento acessar meu servidor baseado no domínio.

O servidor falso pode ser configurado para gravar as senha e responder com uma mensagem do tipo "O servidor está em manutenção, tente novamente daqui a algumas horas". Dessa forma, ele vai ter não apenas acesso à minha senha, mas tempo de usá-la para acessar o servidor verdadeiro sem que eu desconfie. Entretanto, isso é previsto dentro do design do SSH; ele percebe que a identificação do servidor mudou e me avisa do problema:

Para continuar é preciso que eu remova a linha com a identificação do servidor salva no arquivo `ssh/known_hosts`, dentro do diretório home do usuário que está utilizando. Isso pode ser feito de duas formas.

A primeira é remover a chave usando o comando `ssh-keygen -R`, seguido pelo endereço IP ou o domínio do servidor (o mesmo que eu especifico ao me conectar a ele), como em:

**# ssh-keygen -R 10.0.0.5**

**/home/gdhn/.ssh/known\_hosts updated.**

Original contents retained as **/home/gdhn/.ssh/known\_hosts.old**

Como pode ver, o comando substituiu a edição manual do arquivo, removendo a linha correspondente ao servidor e salvando um backup do arquivo original por precaução.

A segunda opção é editar diretamente o arquivo **“.ssh/known\_hosts”**, comentando ou removendo a linha referente ao servidor (deixando as demais). Dentro do arquivo, eu irei ver que uma longa linha para cada servidor acessado, começando com o endereço IP ou o nome de domínio usado no acesso.

Em qualquer um dos dois casos, da próxima vez que eu tentar se conectar, o SSH exibe uma mensagem mais simpática, perguntando se eu quero adicionar a nova chave:

The authenticity of host '192.168.1.200 (192.168.1.200)' can't be established.

RSA key fingerprint is f1:0f:ae:c6:01:d3:23:37:34:e9:29:20:f2:74:a4:2a.

Are you sure you want to continue connecting (yes/no)?

Não existe forma de fazer com que o cliente SSH adicione as novas chaves automaticamente, isso seria uma brecha de segurança. É sempre preciso primeiro remover a chave antiga manualmente.

As chaves de identificação são geradas durante a instalação do SSH e salvas nos arquivos **“/etc/ssh/ssh\_host\_rsa\_key”** e **“/etc/ssh/ssh\_host\_dsa\_key”** (no servidor). Para não disparar o alarme nos clientes quando precisar reinstalar o sistema no servidor, ou quando substituí-lo por outra máquina, salve os dois arquivos em um pendrive e restaure-os depois da instalação. Eu também poderei fazer isso mesmo ao migrar para outra distribuição, pois as localizações dos dois arquivos não mudam.

Uma última opção seria desabilitar a checagem das chaves, adicionando a opção **“StrictHostKeyChecking no”** na configuração dos clientes. Contudo, isso não é recomendável, pois desabilita completamente a checagem, abrindo brechas para ataques.

Eu também posso configurar várias opções relacionadas ao servidor SSH, incluindo a porta TCP a ser usada editando o arquivo **“/etc/ssh/sshd\_config”**. Assim como no caso da configuração do cliente, a maior parte das opções dentro do arquivo podem ser omitidas (pois o servidor simplesmente utiliza valores default para as opções que não constarem no arquivo), mas, de qualquer forma, é saudável especificar todas as opções que conhece: além de evitar enganos, é uma forma de praticar e memorizar as opções.

Porta: Uma das primeiras linhas é a:

## Port 22

Esta é a porta que será usada pelo servidor SSH. O padrão é usar a porta 22. Ao mudar a porta do servidor aqui, eu terei que usar a opção **“-p”** ao conectar a partir dos clientes para indicar a porta usada, como em:

**# ssh -p 2222 ecouto@gnulinuxbr.wordpress.com.br**

Outra opção é editar o arquivo **“/etc/ssh/sshd\_config”** (nos clientes) e alterar a porta padrão usada por eles. Mudar a porta padrão do SSH é uma boa ideia quando entra a questão com a segurança. Muitos dos ataques “casuais” (quando o atacante simplesmente varre faixas inteiras de endereços, sem um alvo definido), começam com um portscan genérico, onde é feita uma varredura em faixas inteiras de endereços IP, procurando por servidores com portas conhecidas abertas, como a 21, a 22 e a 80. Isso torna o teste mais rápido, permitindo localizar rapidamente um grande volume de hosts com portas abertas. A partir daí, os



ataques vão sendo refinados e direcionados apenas para os servidores vulneráveis encontrados na primeira varredura. Colocar seu servidor para escutar uma porta mais escondida, algo improvável como a porta 32456 ou a 54232 reduz sua exposição.

Controle de acesso: Logo abaixo vem a opção "**ListenAddress**", que permite limitar o SSH a uma única interface de rede (mesmo sem usar firewall), útil em casos de micros com duas ou mais placas; o típico caso em que eu quero que o SSH fique acessível apenas na rede local, mas não na Internet, por exemplo. Digamos que o servidor use o endereço "**10.0.0.2**" na rede local e eu queira que o servidor SSH não fique disponível na Internet. Então eu adicionaria a linha:

**ListenAddress 10.0.0.2**

Note que especificamos nesta opção o próprio IP do servidor na interface escolhida, não a faixa de IP's da rede local ou os endereços que terão acesso a ele.

Protocolo: Atualmente utilizamos o SSH 2, mas ainda existem alguns poucos clientes que utilizam a primeira versão do protocolo. Por padrão, o servidor SSH aceita conexões de clientes que utilizam qualquer um dos dois protocolos, o que é indicado na linha:

**Protocol 2,1**

O protocolo SSH 1 tem alguns problemas fundamentais de segurança, por isso é recomendável desativar a compatibilidade com ele, aceitando apenas clientes que usam o SSH 2. Neste caso, a linha fica apenas:

**Protocol 2**

Restringir a versão do protocolo ajuda a melhorar a segurança, pois evita que usuários utilizando clientes SSH antigos abram brechas para ataques. Existem, por exemplo, muitos clientes SSH para uso em celulares que suportam apenas o SSH 1 e utilizam algoritmos fracos de encriptação. Desativando a compatibilidade com o SSH 1 eu cortarei o mal pela raiz.

Usuários e senhas: Outra opção interessante, geralmente colocada logo abaixo, é a:

**PermitRootLogin yes**

Esta opção determina se o servidor aceitará que usuários se loguem como root. Do ponto de vista da segurança, é melhor deixar esta opção como "no", pois assim o usuário precisará primeiro se logar usando um login normal e rodar comandos como root usando o "sudo" ou "su -":

**PermitRootLogin no**

Dessa forma, é preciso saber duas senhas, ao invés de saber apenas a senha do root, eliminando a possibilidade de algum atacante obstinado conseguir adivinhar a senha de root e, assim, obter acesso ao servidor.

Por padrão, o SSH permite que qualquer usuário cadastrado no sistema se logue remotamente, mas eu também posso refinar isso através da opção "AllowUsers", que especifica uma lista de usuários que podem usar o SSH. Quem não estiver na lista, continua usando o sistema localmente, mas não consegue se logar via SSH. Isso evita que contas com senhas fracas, usadas por usuários que não têm necessidade de acessar o servidor remotamente coloquem a segurança do sistema em risco. Para permitir que apenas os usuários "joao" e "maria" possam usar o SSH, por exemplo, adicionei a linha:

**AllowUsers joao maria**

Também posso ainda inverter a lógica, usando a opção “DenyUsers”. Neste caso, todos os usuários cadastrados no sistema podem fazer login, com exceção dos especificados na linha, como em:

**DenyUsers ricardo manuel**

Outra opção relacionada à segurança é a:

**PermitEmptyPasswords no**

Esta opção faz com que qualquer conta sem senha fique automaticamente desativada no SSH, evitando que alguém consiga se conectar ao servidor “por acaso” ao descobrir a conta desprotegida. Lembre-se de que a senha é justamente o ponto fraco do SSH. De nada adianta usar 2048 bits de encriptação se o usuário escreve a senha em um post-it colado no monitor, ou deixa a senha em branco.

Banner: Alguns servidores exibem mensagens de advertência antes do prompt de login, avisando que todas as tentativas de acesso estão sendo monitoradas ou coisas do gênero. A mensagem é especificada através da opção “Banner”, onde eu indico um arquivo de texto com o conteúdo a ser mostrado, como em:

**Banner = /etc/ssh/banner.txt**

X11 Forwarding: Além de ser usada na configuração dos clientes, a opção “X11Forwarding” pode ser também incluída na configuração do servidor:

**X11Forwarding yes**

Ela determina se o servidor permitirá que os clientes executem aplicativos gráficos remotamente. Se o servidor for acessado via Internet ou se possui um link lento, eu posso deixar esta opção como “no” para economizar banda. Desta forma, os clientes poderão executar apenas comandos e aplicativos de modo texto. Note que ao usar “X11Forwarding no” o encaminhamento é recusado pelo próprio servidor, de forma que o usuário não consegue rodar aplicativos gráficos mesmo que a opção esteja ativa na configuração do cliente.

– SFTP: O SSH inclui um módulo de transferência de arquivos (o SFTP). Ele é ativado através da linha:

**Subsystem sftp /usr/lib/sftp-server**

É realmente necessário que esta linha esteja presente para que o SFTP funcione. Comente esta linha apenas se você realmente deseja desativá-lo.

## Servidor SSH – Entendendo sua estrutura

### Comandos intermediários do GNU/Linux

#### Servidor SSH

#### Pequena introdução

Em informática o **Secure Shell** ou **SSH** é, simultaneamente, um programa de computador e um protocolo de rede que permite a conexão com outro computador na rede, de forma a executar comandos de uma unidade remota. Possui as mesmas funcionalidades do **TELNET**, com a vantagem da conexão entre o cliente e o servidor ser criptografada.

O **SSH** é um conjunto de padrões e protocolos associado que permite estabelecer um canal seguro entre dois computadores. Ele utiliza o sistema de chave criptográfica pública para autenticar usuários. A idéia do SSH é prover confidencialidade e integridade dos dados trocados entre dois computadores usando criptografia e mensagens de autenticação codificadas (MAC's).

Este protocolo é tipicamente utilizado para conectar-se à máquina remotas e executar comandos, entretanto, há inúmeras outras funcionalidades como realizar tunelamentos, redirecionamento de portas, conexões X11, além de transferência de arquivos.

Em geral, o **SSH** utiliza a porta **22/TCP** e é a alternativa segura ao **TELNET** e **FTP** uma vez que eles não utilizam criptografia.

## Características

Abaixo as principais características do serviço `ssh`.

- Conexão de dados criptografada entre cliente/servidor.
- Cópia de arquivos usando conexão criptografada.
- Suporte a ftp criptografado (sftp).
- Suporte a compactação de dados entre cliente/servidor.
- Controle de acesso das interfaces servidas pelo servidor `ssh`.
- Suporte a controle de acesso tcp wrappers.
- Autenticação usando um par de chaves pública/privada RSA ou DSA.
- Algoritmo de criptografia livre de patentes.
- Suporte a PAM.
- Suporte a caracteres ANSI (cores e códigos de escape especiais no console).

## Ficha técnica

Pacote: `ssh`

Utilitários:

- `ssh` – Cliente ssh (console remoto).
- `slogin` – Link simbólico para o programa `ssh`.
- `sshd` – Servidor de shell seguro ssh.
- `scp` – Programa para transferência de arquivos entre cliente/servidor
- `ssh-keygen` – Gera chaves de autenticação para o ssh
- `sftp` – Cliente ftp com suporte a comunicação segura.
- `sftp-server` – Servidor ftp com suporte a comunicação segura.
- `ssh-add` – Adiciona chaves de autenticação DSA ou RSA ao programa de autenticação.
- `ssh-agent` – Agente de autenticação, sua função é armazenar a chave privada para autenticação via chave pública (DSA ou RSA).
- `ssh-keyscan` – Scaneia por chaves públicas de autenticação de hosts especificados. O principal objetivo é ajudar na construção do arquivo local `known_hosts`.
- `ssh-copy-id` – Usado para instalação do arquivo `identity.pub` em uma máquina remota.

## Iniciando o servidor/reiniciando/recarregando a configuração

O arquivo que controla o funcionamento do daemon do `ssh` é controlado pelo arquivo `/etc/init.d/ssh`.

A execução do `ssh` através de `inetd` é automática quando é feita uma requisição para a porta **22**.

### Arquivos de configuração:

- `/etc/ssh/sshd_config` – Arquivo de configuração do servidor ssh.
- `/etc/ssh/ssh_config` – Arquivo de configuração do cliente ssh.
- `~/.ssh/config` – Arquivo de configuração pessoal do cliente ssh.

## Opções de linha de comando

### Opções de linha de comando do servidor `sshd`:

- **-b bits** – Especifica o número de bits da chave do servidor (768 por padrão).
- **-d** – Modo de depuração – O servidor envia detalhes sobre seu funcionamento aos logs do sistema e não é executado em segundo plano. Ele também responderá conexões pelo mesmo processo. Podem ser usadas no máximo 3 opções `-d` para aumentar os detalhes de depuração.
- **-f** arquivo\_configuração Indica um arquivo de configuração alternativo (por padrão é usado `/etc/ssh/sshd_config`). O `ssh` pode ser configurado através de opções de linha de comando mas requer um arquivo de configuração para ser executado. Opções de linha de comando substituem as especificadas no arquivo de configuração.
- **-g** segundos – Especifica o tempo máximo para a digitação de senha de acesso. Após o tempo especificado o servidor encerra a conexão. O valor padrão é 600 segundos e 0 desativa este recurso.
- **-h** arquivo\_chave – Diz qual arquivo contém a chave privada local. O padrão é `/etc/ssh/ssh_host_key` e somente o usuário root deve ter permissões de leitura neste arquivo. Será necessário especificar esta opção caso o `sshd` não esteja sendo executado como usuário root. É possível ter múltiplos arquivos de chaves para os protocolos 1 e 2 do ssh.
- **-i** – Indica que o servidor `sshd` será executado pelo `inetd`. Isto não é aconselhável porque o servidor gerará a chave aleatória de seção toda vez que for iniciado e isto pode levar alguns segundos. Esta opção pode se tornar viável com o uso do protocolo 2 ou criando chaves pequenas como 512 bytes (no ssh 1), mas a segurança criptográfica também será diminuída. Veja as diferenças entre os dois protocolos em Diferenças nas versões do protocolo, Seção 15.3.7.
- **-k** segundos – Especifica a frequência da geração de novas chaves do daemon `sshd`. O valor padrão é 3600 segundos e 0 desativa este recurso. **ATENÇÃO:** NÃO desative este recurso!!! Esta opção traz a segurança que uma nova chave gerada de servidor será gerada constantemente (esta chave é enviada junto com a chave pública quando o cliente conecta e fica residente na memória volátil), assim mesmo que um cracker consiga obtê-la interceptando as conexões, será praticamente impossível tentar qualquer coisa. Valores menores tendem a aumentar ainda mais a segurança.
- **-p porta** – Especifica a porta que o daemon `sshd` atenderá as requisições. Por padrão é usada a porta 22.
- **-q** – Nenhuma mensagem será enviada ao `syslog` do sistema.
- **-u tam** – Especifica o tamanho do campo de nome do computador que será armazenado no arquivo `utmp`. A opção `u0` faz somente endereços IP serem gravados.
- **-D** – Quando usada não faz o `sshd` iniciar em segundo plano.
- **-V** versão\_cliente – Assume que o cliente possui a versão ssh especificada (1 ou 2) e não faz os testes de identificação de protocolo.
- **-4** – Força o uso do protocolo IP tradicional (IPv4).
- **-6** – Força o uso da nova geração do protocolo IP (IPv6).

A maioria das opções são realmente úteis para modificar o comportamento do servidor `ssh` sem mexer em seu arquivo de configuração (para fins de testes) ou para executar um servidor `ssh` pessoal, que deverá ter arquivos de configuração específicos.

## ssh

Esta é a ferramenta usada para seções de console remotos. O arquivo de configuração de usuários é `~/.ssh/config` o arquivo global `/etc/ssh/ssh_config`. Para conectar a um servidor ssh remoto:

### ssh usuário@ip/nome-do-servidor-ssh

Caso o nome do usuário seja omitido, seu login atual do sistema será usado. O uso da opção `-C` é recomendado para ativar o modo de compactação dos dados (útil em conexões lentas). A opção `-l usuário` pode ser usada para alterar a identificação de

usuário (quando não é usada, o login local é usado como nome de usuário remoto). Uma porta alternativa pode ser especificada usando a opção **-p porta** (a 22 é usada por padrão).

Na primeira conexão, a chave pública do servidor remoto será gravada em `~/.ssh/known_hosts` ou `~/.ssh/known_hosts2`, e verificada a cada conexão como checagem de segurança para se certificar que o servidor não foi alvo de qualquer ataque ou modificação não autorizada das chaves. Por padrão, o cliente utilizará o protocolo ssh **versão 1**, a opção **-2** permite usar o protocolo **versão 2**.

Variáveis de ambiente personalizadas para o `ssh` poderão ser definidas no arquivo `~/.ssh/environment`. Comandos que serão executados somente na conexão ssh em `~/.ssh/rc` e `/etc/ssh/sshrc` caso contrário será executado o `xauth` por padrão.

**OBS:** Para utilizar autenticação Rhosts/Rhosts+RSA (arquivos `~/.rhosts`/`~/.shosts`) o programa `ssh` deverá ter permissões SUID root e conectará usando portas baixas (menores que 1024).

### Exemplos:

# Conectar-se ao servidor remoto usando o login do usuário atual

**ssh ftp.sshserver.org**

# Conectar-se ao servidor remoto usando o login ecouto (via ssh versão 2)

**ssh -2 ftp.sshserver.org -l ecouto**

# Conectar-se ao servidor remoto usando compactação e com o login ecouto

**ssh ftp.sshserver.org -C -l ecouto**

# Semelhante ao exemplo acima, usando o formato "login@ip"

**ssh ecouto@ftp.sshserver.org -C**

# Conecta-se ao servidor remoto usando compactação, e login ecouto, ativando o redirecionamento do agente de autenticação -A e redirecionando a conexão X11 -X.

**ssh ftp.sshserver.org -C -A -X -l ecouto**

### Redirecionamento de conexões do X

O redirecionamento de conexões do X Window poderá ser habilitado em `~/.ssh/config` ou `/etc/ssh/ssh_config` usando as opções **-A -X** na linha de comando do `ssh` (as opções **-a** e **-x** desativam as opções acima respectivamente). Uma variável **\$DISPLAY** é criada automaticamente para fazer o redirecionamento ao servidor **X** local.

Ao executar um aplicativo remoto, a conexão é redirecionada a um **DISPLAY** proxy criado pelo ssh (a partir de **:10**, por padrão) que faz a conexão com o display real do X (**:0**), ou seja, ele pulará os métodos de autenticação `xhost` e cookies. Por medidas de segurança é recomendável habilitar o redirecionamento individualmente somente se você confia no administrador do sistema remoto.

## Exemplo de configuração do arquivo ssh\_config

# permitir redirecionamento de conexões para o próprio computador [nomes de máquinas podem ser especificados também].

**Host 127.0.0.1**

**ForwardAgent yes**

**ForwardX11 yes**

# opções específicas do cliente para conexões realizadas a 10.0.0.2 usando somente o protocolo 2

**Host 10.0.0.2**

# as 2 linhas abaixo ativam o redirecionamento de conexões do X

**ForwardAgent yes**

**ForwardX11 yes**

**PasswordAuthentication yes**

**Port 22**

**Protocol 2**

**Cipher blowfish**

# opções específicas do cliente para conexões realizadas a 10.0.0.3 usando somente o protocolo 1

**Host 10.0.0.3**

# As 2 linhas abaixo desativam o redirecionamento de conexões do X

**ForwardAgent no**

**ForwardX11 no**

**ForwardX11 no**

**PasswordAuthentication yes**

**Port 22**

**Protocol 1**

**Cipher blowfish**

```
# CheckHostIP yes
```

```
# RhostsAuthentication no
```

```
# RhostsRSAAuthentication yes

# RSAAuthentication yes

# FallBackToRsh no

# UseRsh no

# BatchMode no

# StrictHostKeyChecking yes

# IdentityFile ~/.ssh/identity

# IdentityFile ~/.ssh/id_dsa

# IdentityFile ~/.ssh/id_rsa1

# IdentityFile ~/.ssh/id_rsa2

# EscapeChar ~
```

## Ciente ssh para Windows

O **putty** é um cliente ssh Win32 que possui suporte aos protocolos versão 1 e 2 do ssh, aceita compactação além de funcionar também como cliente **telnet**. Seu tamanho é pequeno, apenas um executável e requer 220KB de espaço em disco. Ele pode ser baixado de [este link](#).

Outra alternativa é o **MindTerm**, este é baseado em Java e pode inclusive ser executado como um applet em uma página web. Este programa é encontrado em [este link](#).

## sftp

Permite realizar transferência de arquivos seguras através do protocolo ssh. A conexão e transferências são realizadas através da porta 22 (ainda não é possível modificar a porta padrão). A sintaxe para uso deste comando é a seguinte:

```
sftp usuario@host_remoto
```

Compactação pode ser especificada através da opção **-C**. Um arquivo contendo os comandos usados na seção **sftp** poderá se especificado através da opção **-b arquivo** para automatizar tarefas.

**OBS1:** Para desativar o servidor **sftp**, remova a linha SubSystem sftp /usr/lib/sftp-server (que inicializa o sub-sistema ftp) do arquivo **/etc/ssh/sshd\_config** e reinicie o servidor **sshd**.

**OBS2:** O suporte ao programa **sftp** somente está disponível ao protocolo ssh versão 2 e superiores.

**OBS3:** Algumas opções comuns do cliente **ftp** padrão (como *mget*) ainda não estão disponíveis ao **sftp**. Veja a página de manual para detalhe sobre as opções disponíveis.

## sshd

Este é o daemon de controle da conexão encriptada via protocolo ssh, transferência de arquivos e shell interativo.

**OBS1:** É recomendável que o arquivo `/etc/ssh/sshd_config` seja lido somente pelo dono/grupo, por conter detalhes de acesso de usuários, grupos e intervalo entre a geração de chave de seção.

**OBS2:** Se estiver ocorrendo falhas no acesso ao servidor ssh, verifique as permissões nos arquivos `/etc/hosts.allow` e `/etc/hosts.deny` (o nome do serviço é `sshd`). Mesmo operando como daemon, o servidor utiliza estes arquivos para fazer um controle de acesso adicional.

## Exemplo de `sshd_config` com explicações das diretivas

Abaixo segue um exemplo deste arquivo que poderá ser adaptado ao seu sistema. O objetivo é ser ao mesmo tempo útil para sua configuração e didático:

# porta padrão usada pelo servidor sshd, múltiplas portas podem ser especificadas separadas por espaços.

**port 22**

# especificando o endereço IP das interfaces de rede que o servidor sshd servirá a requisições. Múltiplos endereços podem ser especificados separados por espaços. A opção port deve vir antes desta opção

**ListenAddress 0.0.0.0**

# protocolos aceitos pelo servidor, primeiro será verificado se o cliente é compatível com a versão 2 e depois a versão 1. Caso seja especificado somente a versão 2 e o cliente seja versão 1, a conexão será descartada. Quando não é especificada, o protocolo ssh 1 é usado como padrão.

**protocol 2,1**

# as 4 opções abaixo controlam o acesso de **usuários/grupos** no sistema. Por padrão o acesso a todos é garantido (**exceto o acesso root se PermitRootLogin for "no"**). **AllowUsers** e **AllowGroups** definem uma lista de **usuários/grupos** que poderão ter acesso ao sistema. Os coringas "\*" e "?" podem ser especificados. Note que somente **NOMES** são válidos, **UID** e **GID** não podem ser especificados. As diretivas **Allow** são processadas primeiro e depois **Deny**. O método que estas diretivas são processadas é idêntico a diretiva "**Order mutual-failure**" do controle de acesso do **Apache**; o usuário deverá ter acesso via **AllowUsers** e **AllowGroups** e não será bloqueado, por **DenyUsers** e **DenyGroups** para ter acesso ao sistema. Se uma das diretivas não for especificada, "\*" é assumido como padrão. Estas permissões são checadas após a autenticação do usuário, porque dados a ele pelo **/etc/passwd** e **PAM** são obtidos após o processo de autenticação.

# **AllowUsers** ecouto administrativo

# **DenyUsers** root adm

# **AllowGroup** users

# **DenyGroups** root adm bin

# **Permite (yes) ou não (no) o login do usuário root**

**PermitRootLogin no**

# chaves privadas do servidor (as chaves públicas possuem um .pub adicionado no final do arquivo)



**HostKey /etc/ssh/ssh\_host\_key**

**HostKey /etc/ssh/ssh\_host\_rsa\_key**

**HostKey /etc/ssh/ssh\_host\_dsa\_key**

# tamanho da chave. 768 bits é padrão

**ServerKeysBits 768**

# tempo máximo para login no sistema antes da conexão ser fechada

**LoginGraceTime 600**

# tempo para geração de nova chave do servidor (segundos). O padrão é 3600 segundos (1 hora)

**KeyRegenerationInterval 3600**

# ignora os arquivos ~/.rhosts e ~/.shosts

**IgnoreRhosts yes**

# ignora (yes) ou não (no) os arquivos ~/.ssh/known\_hosts quando for usado para opção **RhostsRSAAuthentication**. Se você não confia neste mecanismo ajuste está opção para **yes**

**IgnoreUserKnownHosts no**

# checa por permissões de dono dos arquivos e diretório de usuário antes de fazer o login. É muito recomendável para evitar riscos de segurança com arquivos lidos todos os usuários

**StricModes yes**

# permite (yes) ou não (no) o redirecionamento de conexões X11. A segurança do sistema não aumentada com a desativação desta opção, outros métodos de redirecionamento podem ser usados

**X11Forwarding yes**

# especifica o número do primeiro display que será usando para o redirecionamento X11 do ssh. Por padrão é usado o display 10 como inicial para evitar conflitos com display X locais.

**X11DisplayOffset 10**

# mostra (yes) ou não (no) a mensagem em /etc/motd no login. O padrão é "no".

**PrintMotd no**

# mostra (yes) ou não (no) a mensagem do último login do usuário. O padrão é "no".

**PrintLastLog no**

# permite (yes) ou não (no) o envio de pacotes Keepalive para verificar se o cliente responde. Isto é bom para fechar conexões que não respondem mas também podem fechar conexões caso não existam rotas para o cliente naquele momento (é um problema temporário). Colocando esta opção como "no" por outro lado pode deixar usuários que não tiveram a oportunidade de efetuar o logout do servidor dados como "permanentemente conectados" no sistema. Esta opção deve ser ativada/desativada aqui e no programa cliente para funcionar.

#### **KeepAlive yes**

# facilidade e nível das mensagens do sshd que aparecerão no syslogd syslogFacility AUTH

#### **LogLevel INFO**

# especifica se somente a autenticação via arquivos ~/.rhosts e etc/hosts.equiv é suficiente para entrar no sistema. Não é muito bom usar "yes" aqui.

#### **RhostsAuthentication no**

# mesmo que o acima com o acréscimo que o arquivo /etc/ssh/ssh\_known\_hosts também é verificado. Também evite usar "yes" aqui.

#### **RhostsRSAAuthentication no**

# especifica se autenticação via RSA é permitida só usado na versão 1 do protocolo ssh. Por padrão "yes".

#### **RSAAuthentication yes**

# permite autenticação usando senhas server para ambas as versões 1 e 2 do ssh. O padrão é "yes".

#### **PasswordAuthentication yes**

# se a PasswordAuthentication for usada, permite (yes) ou não (no) login, para usar sem senhas. O padrão é "no".

#### **PermitEmptyPasswords no**

# Ativa senas s/key ou autenticação PAM NB interativa. Nenhum destes é compilado por padrão junto com o sshd. Leia a página do manual do sshd antes de ativar esta opção em um sistema que usa PAM.

#### **ChallengeresponseAuthentication no**

# verifica se o usuário possui emails ao entrar no sistema. O padrão é "no". Este módulo também pode estar sendo habilitado usando PAM, neste caso cheque a configuração em /etc/pam.d/sshd.

#### **CheckMail no**

# especifica se o programa login é usado para controlar as sessões de shell interativo. O padrão é "no".

#### **UseLogin no**

# especifica o número máximo de conexões de autenticação simultâneas feitas pelo daemon sshd. O valor padrão é 10. Valores aleatórios pode ser especificados usando os campos "inicio:taxa:maximo". Por exemplo, 5:40:15 rejeita até 40% das tentativas de

autenticação que excedam o limite de 5 até atingir o limite máximo de 15 conexões, quando nenhuma nova autenticação é permitida.

### **MaxStartups 10**

#Maxstartups 10:30:60

# mostra uma mensagem antes do nome de usuário/senha

### **Banner /etc/issue.net**

# especifica se o servidor sshd fará um DNS reverso para verificar se o endereço confere com a origem (isto é útil para bloquear conexões falsificadas – spoofing). O padrão é “no”

### **ReverseMappingCheck yes**

# ativa o subsistema de ftp seguro. Para desabilitar comente a linha abaixo

### **Subsystem sftp /usr/lib/sftp-server**