Chapter 8:
Remote access

Learning objectives

Upon completing this chapter, the learner should be able to:

- Access remote systems using ssh
- Configure key–based authentication for SSH
- Securely transfer files between systems
- Configure additional options

Key terms

Secure Shell (SSH)

Key–based login

Public key

Private key

encryption techniques

Digital signatures

SSH tunneling

Passphrase

Table of content

1. Working with Secure Shell (SSH)

Secure Shell (SSH) provides a secure mechanism for data transmission between source and destination systems over IP networks, it uses encryption techniques to secure a communication channel and employs digital signatures for user authentication, it also performs checks on the data being transferred throughout a session to ensure integrity.

SSH includes a set of utilities, providing remote users with the ability to log in, transfer files, and execute commands securely using strong encryption and authentication mechanisms. Due to their powerful security features, Secure Shell is the common method to gain access to other machines over the network.

1.1. Authentication methods

Once an encrypted channel is established between client and server, additional negotiations take place between the two to authenticate the user trying to gain access to the server. Several methods for authentication are offered for this purpose:

- **GSSAPI-Based Authentication:** GSSAPI provides a standard interface that allows security mechanisms such as Kerberos to be plugged in.
- **Host-Based Authentication:** This type of authentication allows a single user, a group of users, or all users on the client to be authenticated on the server.
- **Private/Public Key-Based Authentication:** This method uses a private/public key combination for user authentication.
- **Challenge-Response Authentication:** This method is based on the response(s) to one or more arbitrary challenge questions that the user has to answer correctly in order to gain login access to the server.
- **Password-Based Authentication:** The server prompts the user to enter their password. It checks the password against the stored entry in the shadow (or password) file and allows the user in if the password is confirmed.

1.2. OpenSSH packages

OpenSSH has three packages:

- **The openssh package** provides the ssh-keygen command and some supported library routines
- **the openssh-clients package** includes commands such as scp, sftp, slogin, ssh, and ssh–copy-id, and a client configuration file /etc/ssh/ssh_config
- and **the openssh-server package** contains the sshd daemon, a server configuration file /etc/ssh/sshd_config, and library routines

By default, the openssh-server package is installed during OS installation, allowing users to log in to the system using the ssh command.

1.2.1. Configuration files

OpenSSH has configuration files that define default global settings for both server and client to control how they should operate:

- These files are located in the /etc/ssh directory, and are called sshd_config and ssh_config, respectively
- The server has an additional configuration file called sshd and it is located in the /etc/sysconfig directory
- In addition, the /var/log/secure log file is used to capture authentication messages

A few directives from the sshd_config file are displayed below:

```
root@server1:~# cat /etc/ssh/sshd_config
...
#Port 22
#Protocol 2
...
HostKey /etc/ssh/ssh_host_rsa_key
HostKey /etc/ssh/ssh_host_ecdsa_key
...
Sy slogFacility AUTHPRIV
AuthorizedKeysFile .ssh/authorized_keys
PasswordAuthentication yes
ChallengeResponseAuthentication no
...
GSSAPIAuthentication yes
UsePAM yes
X11Forwarding yes
```

1.2.2. Commands and daemon

The server package also loads the sshd daemon that must run on the system to allow users to be able to connect to the system over the ssh channel.

Command	Description
scp	Secure alternative to the rcp command
sftp	Secure alternative to the ftp command.
slogin	Secure alternative to the rlogin command.
ssh	Secure alternative to telnet and rlogin commands.
ssh–add	Adds RSA/DSA/ECDSA characteristics to ssh–agent.
sh–agent	Authentication Agent. Holds private keys used for the RSA/DSA/ECDSA authentication.
ssh–copy–id	Copies RSA/DSA/ECDSA keys to remote systems.
ssh–keygen	Generates private and public keys.

1.2.3. Per-User client files

OpenSSH allows us to store default values for our ssh communication sessions in a separate file. These per-user settings override the system-wide defaults and they are saved in a file called config in the ~/.ssh directory. The format of this file is the same as that of the ssh_config file. ~/.ssh does not exist by default, it is created when we attempt to connect to the server the first time and it accepts the presented key.

The ~/.ssh directory has a file called known_hosts. This file stores the hostname, IP address, and a copy of the public key of the remote system that we have accessed. A sample entry from this file on server1 is shown below:

```
server2,192.168.0.120
ecdsa-sha2-
nistp256AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAA
ABBBCP4LK2GHWIDv+a/QX/gp0ggl8Q1H85SLExeyeS4ph6o8Nl/ArWWeB
UWSl1PLDUIczGGsLjeUBtUgXdVDhwO/
```

There are three additional files in ~/.ssh that we may be interested in looking at, which are not exist by default:

- Two of them hold private (id_rsa, id_dsa, or id_ecdsa) and public (id_rsa.pub, id_dsa.pub, or id_ecdsa.pub) keys
- and the third one (authorized_keys) is used when we set up a trust relationship between two systems

See the Generating key pair for more details about the contents of those files.

1.3. Setting up SSH connection

To access a server using SSH, two components are needed. The remote server and a client in the local machine to connect to there.

1.3.1. The remote server

On the remote server that you want to access, the sshd service must be running and offering services at port 22, and it should not be blocked by the firewall. If the SSH port is open, you can access it using the ssh command from the command line, which tries to reach the ssh process on the server port 22.

If you have configured the sshd process to offer its services on a different port, use ssh-p followed by the port number you want to connect to, after connecting, you will be prompted for a password if a default configuration is used.

```
[root@server1:~]# systemctl status sshd
● sshd.service - OpenSSH server daemon
   Loaded:     loaded     (/usr/lib/systemd/system/sshd.service;
enabled; vendor preset: enabled)
   Active: active (running) since Sat 2019-12-14 01:50:54 EET;
57min ago
     Docs: man:sshd(8)
           man:sshd_config(5)
 Main PID: 1563 (sshd)
   CGroup: /system.slice/sshd.service
           └─1563 /usr/sbin/sshd -D

Dec  14  01:50:53  server1.localdomain.com  systemd[1]:  Starting
OpenSSH server daemon
...
Accepted password for root from 192.168.114.147 port 58878 ssh2
```

1.3.2. The client machine

When remotely connecting to a Linux server, the SSH client tries to do that as the user account you are currently logged in with on the local machine. If you want to connect using a different user account, you can specify the name of this user on the command line, in the **user@server** format. If, for instance, you want to establish an SSH session as user root to a remote server, type ssh root@remoteserver, or type ssh remoteserver -l root alternatively.

```
[root@client:~]# ssh root@192.168.114.148
The      authenticity      of      host      '192.168.114.147
(192.168.114.148)' can't be established.
ECDSA key fingerprint is
SHA256:OmNQ3I+DyyEGGOlTJmjFRLbc0OYMuDeWrrxHfaKZDfU.
ECDSA key fingerprint is
MD5:9b:81:31:2f:fa:e9:68:81:5f:bf:df:69:a6:d9:97:7f.
Are you sure you want to continue connecting (yes/no)?

[root@server1:~]#
```

You have noticed that while using SSH to connect to the remote server a security message was displayed, this is because the remote server has never been contacted before and therefore there is no way to verify the identity of the remote server. After connecting to the remote server, a public key fingerprint is stored in the file ~/.ssh/ known_hosts.

The next time you connect to the same server, this fingerprint is checked with the encryption key that was sent over by the remote server to initialize contact. If the fingerprint matches, you will not see this message anymore.

In some cases, the remote host key fingerprint does not match the key fingerprint that is stored locally. That is a potentially dangerous situation. Instead of being connected to the intended server, you might be connected to the server of an evildoer.

It does, however, also happen if you are connecting to an IP address that you have been connected to before, but this IP address is now in use by a different server. In that case, you just have to remove the key fingerprint from the ~/.ssh/known_hosts file on the client computer.

Exercise: Using SSH to log in to a remote server

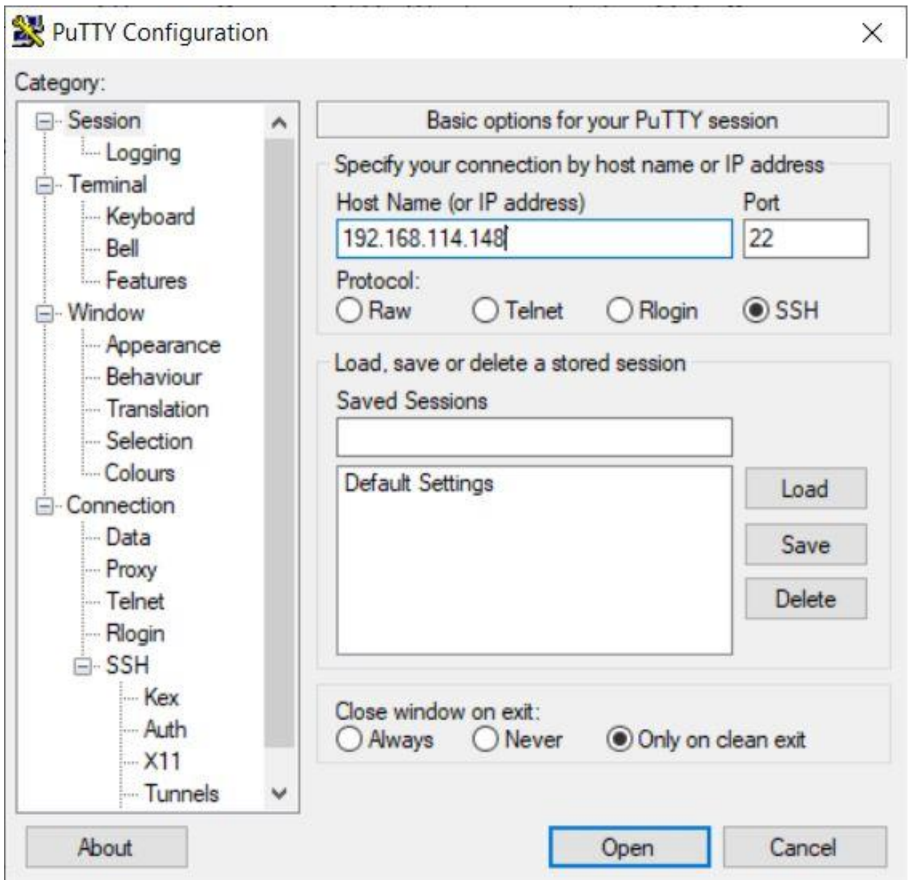
In this exercise, you log in to a remote server using SSH. This exercise assumes that a remote server is available and reachable. In this exercise, server1 is used as the local server, and server2 is the remote server on which the SSH process should be up and running. If you cannot access a remote server to perform the steps in the exercise, you might alternatively replace server2 with localhost. It is obvious that by doing so you will not log in to a remote server, but you still use the ssh command to connect to sshd process.

1. Open a root shell on server2. Type systemctl status sshd. This should show you that the sshd process is currently up and running.
2. To avoid any firewall – related problems, type systemctl stop firewalld.
3. Type ip a | grep 'inet'. (Notice the space between inet and the closing quote.) Notice the IPv4 address your server is currently using. In the rest of this exercise, it is assumed that server2 is using IP address 192.168.122.220. Replace that address with the address that you have found here.

4. Open a shell as a non privileged user on server1.
5. On server1, type ssh 192.168.122.220 –l root. This connects to the sshd process on server2 and opens a root shell.
6. Before being prompted for a password, you see a message indicating that the authenticity of host 192.168.122.220 cannot be established. This message is shown because the host you are connecting to is not yet known on your current host, which might involve a security risk. Type yes to continue.
7. When prompted, enter the root password. After entering it, you now are logged in to server2.
8. Type w. Notice that the SSH session you have just opened shows as just another pseudo terminal session.
9. Type exit to close the SSH session.

1.3.3. Accessing OpenSSH Server from Windows

The ssh command is not a native part of the Windows operating system. If you want to access Linux servers through SSH from a Windows computer, you need to install an SSH client like PuTTY on Windows. From PuTTY, different types of remote sessions can be established with Linux machines. Alternative SSH clients for Windows are available as well, such as MobaXterm, Kitty, mRemoteNG, Bitvise, and also Xshell.



Basic options for your PuTTY session

Specify your connection by host name or IP address

Host Name (or IP address): 192.168.114.148

Port: 22

Protocol: Raw Telnet Rlogin [✓ SSH]

Load, save or delete a stored session

Saved Sessions

Default Settings

Close window on exit: Always Never [✓ Only on clean exit]

The GUI is simple and for quick access all you need is:

- The IP address on the DNS name of the remote machine
- The SSH port that is configured on that machine, default 22
- When hitting open button, you will need to provide your credentials on that machine

1.4. Using other useful sshd options

Apart from the security related, there are some useful miscellaneous options that you can use to streamline SSH performance. In the next two subsections, you read about some of the most significant of these options. Here is the most Useful sshd configuration options:

Option	Use
Port	Defines the TCP listening port.
PermitRootLogin	Allow/disallow root login.
MaxAuthTries	Used to specify the maximum number of authentication tries. After reaching half of this number, failures are logged to syslog.
MaxSessions	The maximum number of sessions that can be open from one IP address.
AllowUsers	Used to specify a space-separated list of users that are allowed to connect to the server.
PasswordAuthentication	Specifies whether to allow password authentication. This option is on by default.

GSSAPIAuthentication	Indicates whether authentication through the GSSAPI needs to be enabled. Used for Kerberos based authentication.
TCPKeepAlive	Set to yes if you do not want to clean up inactive TCP connections.
ClientAliveInterval	The interval in seconds that packets are sent to the client to figure out if the client is still alive.
ClientAliveCountMax	The number of client alive packets that needs to be sent.
UseDNS	If on, uses DNS name lookup to match incoming IP addresses to names.
ServerAliveInterval	The interval in seconds that a client sends a packet to a server to keep connections alive.
ServerAliveCountMax	The maximum number of packets a client sends to a server to keep connections alive.

2. Securing SSH server

2.1. Hardening the SSH server

SSH is an important and also a convenient solution that helps you establish remote connections to servers, it is also a dangerous solution. If your server is visible directly from the Internet, you can be sure that sooner or later an intruder will try to connect to your server, intending to do harm.

Dictionary attacks are common against an SSH server, especially when no additional security measures have been taken. Fortunately, you can take some measures to protect SSH servers against these kinds of attacks, in the following subsections, you learn what is involved in changing these options.

2.1.1. Limiting root access

The fact that SSH servers by default have root login enabled is the biggest security problem. Disabling root login is easy, you just have to modify the PermitRootLogin parameter in /etc/ssh/sshd_config and reload or restart the service.

2.1.2. Configuring alternative ports

Many security problems on Linux servers start with a port scan issued by the attacker, most port scans focus on known ports only, and SSH port 22 is always among these ports. To have SSH listen on another port, you must change the option port 22 into something else in the sshd_config file, where different ports can be used.

2.1.3. Modifying SELinux to allow for port changes

After changing the SSH port, you also need to configure SELinux to allow for this change. Network ports are labeled with SELinux security labels to prevent services from accessing ports where they should not go. To allow a service to connect to a non default port, you need semanage port to change the label on the target port.
Before doing so, it is a good idea to check whether the port already has a label. You can do this by using the semanage port -l command. If the port does not have a security label set yet, use −a to add a label to the port. If a security label has been set already, use −m to modify the current security label. Use, for instance, the command semanage port -a -t ssh_port_t -p tcp 2022 to label port 2022 for access by sshd.

2.1.4. Limiting user access

Many options for sshd can be found by just browsing through the sshd_config file, one of the most interesting options to use is **AllowUsers**. This option takes a space separated list of all users that will be allowed to login through SSH, this limiting login to only these users and you have to include root user as well in the list of allowed users.
When using this parameter, it is easy to add a layer of security by selecting an uncommon username, notice the following about the AllowUsers parameter:

- The AllowUsers option does not appear anywhere in the default /etc/ssh/sshd_config file
- The AllowUsers option is a better option than **PermitRootLogin** because it is more restrictive than just denying root to log in
- If the AllowUsers option does not specify root, you can still become root by using su −after making a connection as a normal user

MaxAuthTries does not lock access to the SSH login prompt after a maximum number of failed login attempts, it just starts logging failed login attempts after half the number of successful login attempts specified here. Still, the MaxAuthTries option is useful for analyzing security events related to your SSH server, SSH writes logs information about failed login attempts to the AUTHPRIV syslog facility. By default, this facility is configured to write information about login failures to /var/log/secure.

Exercise: Applying SSH security options

In this exercise, you learn how to apply common SSH security options. The sshd process should be configured on server1. Use server2 to test access to server1.

1. Open a root shell on server1, and from there, open the sshd configuration file /etc/ssh/sshd_config in an editor.
2. Find the Port line, and below that line add the line Port 2022. This tells the sshd process that it should bind to two different ports, which ensures that you can still open SSH sessions even if you have made an error.
3. Add the line AllowUsers user to the SSH configuration file as well.
4. Save changes to the configuration file and restart sshd, using systemctl restart sshd.
5. Type systemctl status −l sshd. You'll see a permission denied error for SSH trying to connect to port 2022.
6. Type semanage port −a −t ssh_port_t −p tcp 2022 to apply the correct SELinux label to port 2022.
7. Open the firewall for port 2022 also, using firewall−cmd −–add–port=2022/tcp, followed by firewall−cmd −−add–port=2022/tcp −−permanent.
8. Restart the sshd service, and type systemctl status −l sshd again. You'll see that the sshd process is now listening on two ports.
9. Try to log in to your SSH server from your other server, using ssh user@server1. After the user shell has opened, type su −to get root access.

2.2. Key–Based authentication for SSH

By default, password authentication is allowed on RHEL7 SSH servers. If a public/private key pair is used, this key pair is used first, if you want to allow public/private key based authentication only and disable password-based authentication completely, set the **PasswordAuthentication** option to no.

The only thing you need to do to login is to create a key pair, when using public/private key–based authentication, the user who wants to connect to a server generates a public/private key pair. The private key needs to be kept private and will never be distributed, where the public key is stored in the home directory of the target user on the SSH server.

When authenticating using key pairs, the user generates a hash derived from the private key. This hash is sent to the server, and if on the server it proves to match the public key that is stored on the server, the user is authenticated.

2.2.1. Generating key pair

To create a key pair, use the ssh-keygen command as shown below:

```
[root@client]:~# ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):
Created directory '/root/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_rsa.
Your public key has been saved in /root/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:BVsFs2Zo3wZd86mK67oJp4ftytvsiVlSRVj+AyacWW0
root@client.localdomain
The key's randomart image is:
+---[RSA 2048]----+
|        ++=o. o  |
|      ..*= E . o.|
|       =+=B .  ..|
|       .+=oo  .  |
|       .S .oo.   |
|      .   ..o    |
|     ooo . .     |
|    ..@oo .      |
|     B*X+o       |
+----[SHA256]-----+
```

When creating a public/private key pair, you are prompted for a passphrase. If you want maximal security, you should enter a passphrase. You are prompted for that passphrase each time that you are using the private key to authenticate to a remote host. That is very secure, but it is not very convenient.

To create a configuration that allows for maximal convenience, like the example above, you can just press the Enter key twice to confirm that you do not want to set a passphrase. This is a typical configuration that is used for authentication between servers in a trusted environment where no outside access is possible anyway.

```
[root@client:~]# ll .ssh/
total 8
-rw-------. 1 root root 1675 Dec 14 02:10 id_rsa
-rw-r--r--. 1 root root  405 Dec 14 02:10 id_rsa.pub
[root@client:~]# cat .ssh/id_rsa
-----BEGIN RSA PRIVATE KEY-----
MIIEogIBAAKCAQEAulVvCOk+YncwytUJDcpf7smp5BNphyK7UAyL+
...
NEoy8joiVc4D01oueBhYyvFtqvIH6xxRspFXGjK0cCHTEIF7hXg=
-----END RSA PRIVATE KEY-----
[root@client:~]# cat .ssh/id_rsa.pub
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQC6VW8I6T5idzDK1QkNyl
...
/blrqOZf5nT+fjnblOe++1ZylkKCvAQ/QD
root@client.localdomain
```

2.2.2. Transferring public key

After creating public and private keys with ssh-keygen command, we can configure a password-less access between ssh client and server by keeping the private key on client system and sending public key to the destination by using the ssh–copy-id command, which is a tool that uses ssh to log into a remote machine, presumably using a login password, and then adds the keys by appending them to the remote user's ~/.ssh/authorized_keys or creating the file, and directory, if necessary.

```
[root@client:~]# ssh-copy-id -i ~/.ssh/id_rsa.pub
root@192.168.114.148
/usr/bin/ssh-copy-id:  INFO:  Source  of  key(s)  to  be
installed: "/root/.ssh/id_rsa.pub"
The authenticity of host '192.168.114.148 (192.168.114.148)'
can't be established.
ECDSA key fingerprint is
SHA256:SZqxrZQu0SY9DAdNP5u6pjtoH5Q7muG7Pb4F/ezHkOE.
ECDSA key fingerprint is
MD5:be:14:3f:91:d7:5d:14:91:cb:38:a1:af:a9:8b:44:7c.
Are you sure you want to continue connecting (yes/no)? yes
/usr/bin/ssh-copy-id: INFO: attempting to log in with the
new key(s), to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed
-- if you are prompted now it is to install the new keys
root@192.168.114.148's password:
192.168.114.148

Number of key(s) added: 1

Now  try  logging  into  the  machine,  with:      "ssh
'root@192.168.114.148'"
and check to make sure that only the key(s) you wanted were
added.
```

On the server you can see the new authorized_keys file and the public key of the client in it as follows:

```
[root@server1:~]# ll .ssh/
total 12
-rw-------. 1 root root  405 Dec 14 02:17 authorized_keys
-rw-------. 1 root root 1675 Dec 14 01:52 id_rsa
-rw-r--r--. 1 root root  410 Dec 14 01:52 id_rsa.pub
[root@server1:~]# cat .ssh/authorized_keys
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQC6VW8I6T5idzDK1QkNyl/
...
1O46WVp1rWG41yE/blrqOZf5nT+fjnblOe++1ZylkKCvAQ/QD
root@client.localdomain
```

Notice that if multiple users are using keys to log in with that specific account, the authorized_keys file may contain a lot of public keys. Make sure never to overwrite it because that will wipe all keys that are used by other users as well!

2.2.3. Using the passphrases

When you use public/private keys, a passphrase can be used, using this makes the key pair stronger. Not only does an intruder have to get access to the private key, but when he does, he must also know the passphrase to use the key, this is why for establishing client/server connections with public/private keys, it is recommended to use passphrases.

Without further configuration, the use of passphrases would mean that users have to enter the passphrase every time before a connection can be created, and that is inconvenient. To make working with passphrases a bit less complicated, the passphrase can be cached for a session, to do this, you need the ssh-agent and ssh-add commands.

Assuming that the public/private key pair has already been created, this is an easy procedure needs to be repeated for all new sessions that are created:

1. Type ssh-agent /bin/bash to start the agent for the current (bash) shell.
2. Type ssh-add to add the passphrase for the current user's private key, the key is now cached.
3. Connect to the remote server, notice that there is no longer a need to enter the passphrase.

Exercise: Connecting to a remote server with Public/Private keys server

In this exercise, you create a public/private key pair to log in to the server2 host. If no remote host is available, you can use localhost as an alternative to verify the procedure.

1. On server1, open a root shell.
2. Type ssh-keygen. When asked whether you want to use a passphrase, press Enter to use the passphrase-less setup.
3. When asked for the filename in which to store the (private) key, accept the default filename ~/.ssh/id_rsa.
4. When asked to enter a passphrase, press Enter twice.
5. The private key will now be written to the ~/.ssh/id_rsa file and the public key is written to the ~/.ssh/id_rsa.pub file.
6. Use ssh–copy-id server2 to copy the public key you have just created over to server2. You are then asked for the password on the remote server one last time.
7. After copying the public key, verify that it can actually be used for authentication.

To do this, type ssh server2. You should now authenticate without having to enter the password for the remote user account.

3. Using the screen command

The screen command is a useful command that allows you to open multiple terminal sessions, even if you are not in a graphical session. It does not just allow you to run multiple terminals but also allows you to share a session with other users, or to attach and detach to remote terminal sessions.

When screen is called, it creates a single window with a shell in it (or the specified command) and then gets out of your way so that you can use the program as you normally would. Then, at any time, you can create new (full-screen) windows with other programs in them (including more shells), kill the current window, view a list of the active windows, copy text between windows, switch between windows, etc.

Screen manages a session consisting of one or more windows each containing a shell or other program. Furthermore, screen can divide a terminal display into multiple regions, each displaying the contents of a window. All windows run their programs completely independent of each other, and programs continue to run when their window is currently not visible and even when the whole screen session is detached from the user's terminal.

Before you can use screen, you need to install it, to do this, use the yum install-y screen command. Then, type screen to open a screen session. From the screen session, you can start any command you like.

While working with screen, you are working in a specific application. From within this application you can issue specific commands. To use these specific screen commands, start by typing Ctrl+a, ?. This shows a list of all commands that are available.

Every screen command is started with the Ctrl+a key sequence. An important command to remember is Ctrl+a, /, which will close the screen session. Make sure to remove screen sessions that you do not need anymore, otherwise they will stay active until the next time you reboot your server!

3.1. Main commands of screen

It is possible to operate with screen by using some commands, here is a list of the main commands more useful:

- screen -S <screen-name>: start a new window within the screen and also gives a name to the window
- screen -ls: display the currently opened screens including those running in the background
- Ctrl-a followed by c: create a new window
- Ctrl-a followed by w: display the list of all the windows currently opened
- Ctrl-a followed by A: rename the current windows, the name will appear and when you will list the list of windows opened with Ctrl-a followed by w
- Ctrl-a followed by a number from 0 to X: go the windows no X
- Ctrl-a followed by ": choose the windows into which to move on
- Ctrl-a followed by k: close the current windows (kill)
- Ctrl-a followed by d: detach a screen session without stopping it
- Ctrl-a followed by r: reattach a detached screen session

3.2. Basic usage example

The screen command is particularly useful when used from an SSH session. You can start a task that takes a long time from a screen session, detach from it, and attach to it later. The command continues running, even if you are going home and shut down your computer. The next day, you can easily attach to the screen session again to complete the task. To do this, just follow a simple procedure:

1. Open an SSH session.
2. Type screen to open a screen session.
3. Start whichever task you want to start and keep it running.

```
[root@server1:~]# wget
http://mirror.alastyr.com/centos/8.1.1911/isos/x86_64/Cen
tOS-8.1.1911-x86_64-dvd1.iso
--2020-01-16 12:24:56--
http://mirror.alastyr.com/centos/8.1.1911/isos/x86_64/Cen
tOS-8.1.1911-x86_64-dvd1.iso
Resolving    mirror.alastyr.com    (mirror.alastyr.com)...
5.2.80.19
Connecting              to              mirror.alastyr.com
(mirror.alastyr.com)|5.2.80.19|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 7554990080 (7.0G) [application/octet-stream]
Saving to: ‘CentOS-8.1.1911-x86_64-dvd1.iso’

 0% [        ] 10,751,368   244KB/s   eta 8h 36m
```

4. Use the Ctrl+a, d key sequence to detach from the screen session and log out from the SSH session.
5. Use screen ls to see the deattached screen.

```
[detached from 5270.pts-0.server1]
[root@server1:~]# screen -ls
There is a screen on:
        5270.pts-0.server1      (Detached)
1 Socket in /var/run/screen/S-root.
```

6. When you are ready to reconnect, start the SSH session again. It is essential that you are using the same user account that you used before.
7. Attach to the screen session again using screen -r. You can now conveniently finish the work that you have started from the screen session before.

```
[root@server1:~]# screen -r 5270.pts-0.server1
[root@server1:~]#                                                      wget
http://mirror.alastyr.com/centos/8.1.1911/isos/x86_64/Cen
tOS-8.1.1911-x86_64-dvd1.iso
--2020-01-16                                                   12:24:56--
http://mirror.alastyr.com/centos/8.1.1911/isos/x86_64/Cen
tOS-8.1.1911-x86_64-dvd1.iso
Resolving    mirror.alastyr.com    (mirror.alastyr.com)...
5.2.80.19
Connecting              to              mirror.alastyr.com
(mirror.alastyr.com)|5.2.80.19|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 7554990080 (7.0G) [application/octet-stream]
Saving to: 'CentOS-8.1.1911-x86_64-dvd1.iso'

 5% [>                      ] 551,739,648  36.0KB/s   eta
12h 14m
```

Quiz

Chapter review questions

1. To list all existing screen sessions, use the command:
   a. screen – ls
   b. screen – list
   c. screen – ls – all
2. The primary secure shell server configuration file is ssh _ config.
   a. True
   b. False
3. What is the secure shell equivalent for the telnet command?
   a. rsync
   b. scp
   c. ssh
4. Where are the user ssh keys stored?
   a. Sshd _ config
   b. ssh sub – directory
   c. Ssh – keygen
5. What would the command ssh – copy – id do?
   a. make unsecure remote connection
   b. generate pair of keys
   c. install public keys on remote systems
6. What is the use of the ssh – keygen command?
   a. generate authentication keys for ssh use
   b. install public keys on remote systems
   c. make unsecure remote connection
7. List the two encryption techniques described in this chapter.
   a. secure and insecure
   b. symmetric and asymmetric
   c. local and remote

8. What is the default port used by the secure shell service?

a. 22

b. 2222

c. 80

9. Which log file stores the authentication messages?

a. /var/log/dmseg

b. /var/log/httpd

c. /var/log/secure

10. Which of the following is not a common approach to prevent against brute – force attacks against SSH servers?

a. Disable X11 forwarding

b. Have SSH listening on a nondefault port

c. Disable password login

d. Allow specific users only to log in

11. Which of the following successfully limits SSH server access to users bob and lisa only?

a. LimitUsers bob,lisa

b. AllowedUsers bob lisa

c. AllowUsers bob lisa

d. AllowedUsers bob,lisa

12. Which of the following descriptions is correct for the MaxAuthTries option?

a. After reaching the number of attempts specified here, the account will be locked.

b. This option specifies the maximum number of login attempts. After reaching half the number specified here, additional failures are logged.

c. After reaching the number of attempts specified here, the IP address where the login attempts come from is blocked.

d. The number specified here indicates the maximum amount of login attempts per minute.

13. SSH login in your test environment takes a long time. Which of the following options could be most likely responsible for the connection time problems?

a. UseLogin

b. GSSAPIAuthentication

c. UseDNS

d. TCPKeepAlive

14. Which of the following options is not used to keep SSH connections alive?

a. TCPKeepAlive

b. ClientAliveInterval

c. ClientAliveCountMax

d. UseDNS

15. Which file on SSH client computer needs to be added to set the Server − KeepAliveInterval for an individual client?

a. ~/.ssh/ssh _ config

b. ~/.ssh/config

c. /etc/ssh/config

d. /etc/ssh/ssh _ config

16. Which of the following is true about remote access to Linux servers from a Windows environment?

a. Open a shell terminal on Windows and type ssh. The ssh command is available as a default part of the Windows operating system.

b. Configure Remote Access on Windows if you want to access Linux servers running the sshd process.

c. Install the PuTTY program on Windows to access sshd services on Linux from Windows.

d. You cannot remotely access Linux machines from Windows.

17. What is the name of the file in which the public key fingerprint of the SSH servers you have connected to in the past are stored?

a. /etc/ssh/remote _ hosts

b. /etc/ssh/known _ hosts

c. ~/.ssh/remote _ hosts

d. ~/.ssh/known _ hosts

18. Which of the following statements about key - based SSH authentication is true?

a. After creating the key pair, you need to copy the private key to the remote server.

b. Use scp to copy the public key to the remote server.

c. Use ssh – copy - id to copy the public key to the remote server.

d. Use ssh - keygen on the server to generate a key pair that matches the client keys.

19. Which command enables you to generate an ssh public/private key pair?

a. ssh - keygen

b. ssh – copy - id

c. scp

20. useful command that allows you to open multiple terminal sessions

a. scp

b. ssh

c. screen

d. ssh - keygen

Answers to chapter Review Questions:

1. a
2. b
3. c
4. b
5. c
6. a
7. b
8. a
9. c
10. a
11. c
12. b
13. d
14. d
15. d
16. c
17. d
18. c
19. a
20. c