For example:

```
$ ssh mec@192.168.0.1
The authenticity of host '192.168.0.1 (192.168.0.1)' can't be
established.
RSA key fingerprint is 2b:b4:90:79:49:0a:f1:b3:8a:db:9f:73:2d:75:d
6:f9.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.0.1' (RSA) to the list of
known hosts.
Password:


Last login: Fri Sep  3 05:15:21 2010 from 192.168.0.82
mec@proxy-1:~$
```

It will interactively ask for a user password, and upon successful authentication it will return the shell for the user.

> SSH performs a fingerprint verification to make sure that we are actually connecting to the remote computer we want to. This is to avoid what is called a **man-in-the-middle attack**, where an attacker tries to impersonate another computer. SSH will, by default, store the fingerprint the first time we connect to a server and verify that it does not change for future connections.

By default, the SSH server runs at port 22. But certain servers run SSH service at different ports. In that case, use `-p port_num` with the `ssh` command to specify the port.

2. In order to connect to an SSH server running at port 422, use:

```
$ ssh user@locahost -p 422
```

You can execute commands in the shell that corresponds to the remote host. However, when using `ssh` in shell scripts, we do not want an interactive shell as we require to execute several commands and display or store their output.

> Issuing a password every time is not practical for an automated script, hence password-less login using SSH keys should be configured. The *Password-less auto-login with SSH* recipe explains the SSH commands to set this up.

3. To run a command at the remote host and display its output on the local shell, use the following syntax:

   ```
   $ ssh user@host 'COMMANDS'
   ```

   For example:

   ```
   $ ssh mec@192.168.0.1 'whoami'
   mec
   ```

   Multiple commands can be given by using a semicolon delimiter in between the commands as:

   ```
   $ ssh user@host "command1 ; command2 ; command3"
   ```

   For example:

   ```
   $ ssh mec@192.168.0.1  "echo user: $(whoami);echo OS: $(uname)"
   Password:
   user: mec
   OS: Linux
   ```

   In this example, the commands executed at the remote host are:

   ```
   echo user: $(whoami);
   echo OS: $(uname)
   ```

   It can be generalized as:

   ```
   COMMANDS="command1; command2; command3"
   $ ssh user@hostname  "$COMMANDS"
   ```

   We can also pass a more complex subshell in the command sequence by using the `( )` subshell operator.

4. Let's write an SSH-based shell script that collects the uptime of a list of remote hosts. Uptime is the time for which the system is powered on and the `uptime` command is used to display this information.

   It is assumed that all systems in `IP_LIST` have a common user `test`.

   ```
   #!/bin/bash
   #Filename: uptime.sh
   #Description: Uptime monitor

   IP_LIST="192.168.0.1 192.168.0.5 192.168.0.9"
   USER="test"

   for IP in $IP_LIST;
   ```