```
do
  utime=$(ssh ${USER}@${IP} uptime  | awk '{ print $3 }' )
  echo $IP uptime:  $utime
done
```

Expected output:

```
$ ./uptime.sh
192.168.0.1 uptime: 1:50,
192.168.0.5 uptime: 2:15,
192.168.0.9 uptime: 10:15,
```

There's more...

The `ssh` command can be executed with several additional options. Let's go through them.

SSH with compression

The SSH protocol also supports data transfer with compression, which comes in handy when bandwidth is an issue. Use the `-C` option with the `ssh` command to enable compression as follows:

```
$ ssh -C user@hostname COMMANDS
```

Redirecting data into stdin of remote host shell commands

Sometimes, we need to redirect some data into `stdin` of remote shell commands. Let's see how to do it. An example is as follows:

```
$ echo 'text' | ssh user@remote_host 'echo'
text
```

Or

```
# Redirect data from file as:
$ ssh user@remote_host 'echo'  < file
```

`echo` on the remote host prints the data received through `stdin` which in turn is passed to `stdin` from localhost.

Running graphical commands on a remote machine

If you attempt to use this recipe with a command that needs to show some kind of GUI to the user, you will see an error similar to "cannot open display". This is because the `ssh` shell is not able to connect to the X server running on the remote machine. For this you need to set the `$DISPLAY` variable like this:

```
ssh user@host "export DISPLAY=:0 ; command1; command2"""
```

This will launch the graphical output on the remote machine. If you want to show the graphical output on your local machine, use SSH's X11 forwarding option as follows:

```
ssh -X user@host "command1; command2"
```

This will run the commands on the remote machine, but it will bring the graphical output to your machine.

See also

- The *Password less auto-login with SSH* recipe, explains how to configure auto-login to execute commands without prompting for a password.

Transferring files through the network

The major driver for networking of computers is resource sharing, and file sharing is the most prominent shared resource. There are different methods by which we can transfer files between different nodes on a network. This recipe discusses how to transfer files using commonly used protocols FTP, SFTP, RSYNC, and SCP.

Getting ready

The commands for performing file transfer over the network are mostly available by default with Linux installation. Files can be transferred via FTP using the `lftp` command. Files can be transferred via a SSH connection using `sftp`. Further, we can use RSYNC over SSH with `rsync` command and transfer files through SSH using `scp`.

How to do it...

File Transfer Protocol (**FTP**) is a very old file transfer protocol for transferring files between machines on a network. We can use the command `lftp` for accessing FTP-enabled servers for file transfer. FTP can only be used if the FTP server is installed on the remote machine. FTP is used in many public websites to share files and the service usually runs on port 21.

To connect to an FTP server and transfer files in between, use:

```
$ lftp username@ftphost
```

It will prompt for a password and then display a logged in prompt as follows:

```
lftp username@ftphost:~>
```