Install these tools to ease your job

```
$sudo apt-get install openssh-server
$sudo apt-get install putty-tools
```

To start ssh

```
service ssh start
```

To check status

```
service ssh status
```

SSH: Execute Remote Command

Execute a remote command on a host over SSH:

```
$ ssh USER@HOST 'COMMAND'
```

Examples

Get the uptime of the remote server:

```
$ ssh root@192.168.1.1 'uptime'
```

Reboot the remote server:

```
$ ssh root@192.168.1.1 'reboot'
```

SSH: Run Multiple Remote Commands

In the most cases it is not enough to send only one remote command over SSH.

Much more often it is required to send multiple commands on a remote server, for example, to collect some data for inventory and get back the result.

There are a lot of different ways of how it can be done, but i will show the most popular of them.

Run multiple command on a remote host over SSH:

```
$ ssh USER@HOST 'COMMAND1; COMMAND2; COMMAND3'
```

â€“ or â€“

```
$ ssh USER@HOST 'COMMAND1 | COMMAND2 | COMMAND3'
```

â€“ or â€“

```
$ ssh USER@HOST << EOF
```

```
COMMAND1
COMMAND2
COMMAND3
EOF
```

SSH login is too slow? This can be fixed easily! Get rid of delay during authentication!
Examples

Get the uptime and the disk usage:

```
$ ssh root@192.168.1.1 'uptime; df -h'
```

Get the memory usage and the load average:

```
$ ssh root@192.168.1.1 'free -m | cat /proc/loadavg'
```

Show the kernel version, number of CPUs and the total RAM:

```
$ ssh root@192.168.1.1 << EOF
uname -a
lscpu  | grep "^CPU(s)"
grep -i memtotal /proc/meminfo
EOF
```

SSH: Run Bash Script on Remote Server

The equally common situation, when there is some Bash script on a Linux machine and it needs to connect from it over SSH to another Linux machine and run this script there.

The idea is to connect to a remote Linux server over SSH, let the script do the required operations and return back to local, without need not to upload this script to a remote server.

Certainly this can be done and moreover quite easily.

```
$ ssh USER@HOST 'bash -s' < SCRIPT
```

Execute the local script.sh on the remote server:

```
$ ssh root@192.168.1.1 'bash -s' < script.sh
```

Run a GUI Program on a Remote Computerâ€™s Screen
The Basic Procedure

Log into a remote machine using SSH:

```
$ ssh 192.168.1.100
```

Tell GUI applications to be launched on the local screen (so, any graphical program that you run, will be displayed on the remote computerâ€™s screen):

```
$ export DISPLAY=:0
```

Execute GUI Program. For, example lets start Firefox browser that will be launched and displayed on the remote machineâ€™s screen in which we logged in:

```
$ firefox "www.google.com"
```

Use nohup to prevent a process from being stopped after closing SSH session:

```
$ nohup firefox "www.google.com"
```

Secure Against SSH PGP key Autologin

Navigate to /etc/ssh/
```
$sudo leafpad sshd_config
```

Here, change password authentication from yes to no and uncomment.
Now our port is safe.To this port, the hacker will require physical access to your hardware which is impossible.
In case you want to access SSH from another machine then just configure the same key in that PC too and it has access to it.

Secure SSH through Port Redirection

Navigate to
```
$ cd /etc/ssh/
$ sudo nano sshd_config
```

You will find a line saying what ports to use
Type
```
Port 2222
```
This way we have forwarded SSH service from port 22 to port 2222