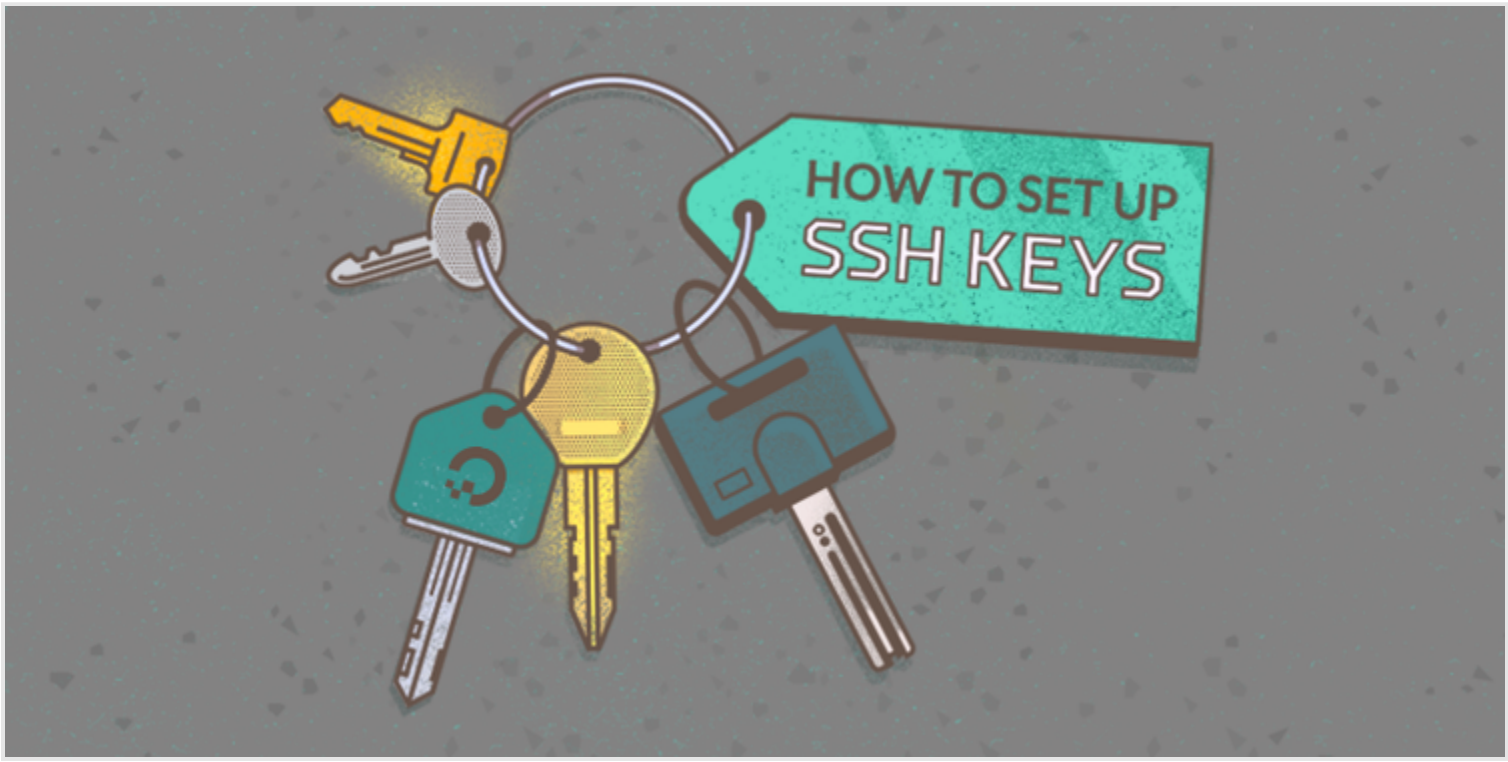

How to Set Up SSH Keys on Ubuntu 18.04

Posted April 27, 2018 82.9k GETTING STARTED LINUX BASICS SECURITY SYSTEM TOOLS UBUNTU 18.04

By: Hanif Jetha

Not using **Ubuntu 18.04**? Choose a different version:

- CentOS 7
- Debian 9
- Ubuntu 16.04

Introduction

SSH, or secure shell, is an encrypted protocol used to administer and communicate with servers. When working with an Ubuntu server, chances are you will spend most of your time in a terminal session connected to your server through SSH.

In this guide, we'll focus on setting up SSH keys for a vanilla Ubuntu 18.04 installation. SSH keys provide an easy, secure way of logging into your server and are recommended for all users.

Step 1 — Create the RSA Key Pair

The first step is to create a key pair on the client machine (usually your computer):

```
$ ssh-keygen
```

By default `ssh-keygen` will create a 2048-bit RSA key pair, which is secure enough for most use cases (you may optionally pass in the `-b 4096` flag to create a larger 4096-bit key).

After entering the command, you should see the following output:

```
Output
Generating public/private rsa key pair.
Enter file in which to save the key (/your_home/.ssh/id_rsa):
```

Press enter to save the key pair into the `.ssh/` subdirectory in your home directory, or specify an alternate path.

If you had previously generated an SSH key pair, you may see the following prompt:

```
Output
/home/your_home/.ssh/id_rsa already exists.
Overwrite (y/n)?
```

If you choose to overwrite the key on disk, you will **not** be able to authenticate using the previous key anymore. Be very careful when selecting yes, as this is a destructive process that cannot be reversed.

You should then see the following prompt:

```
Output
Enter passphrase (empty for no passphrase):
```

Here you optionally may enter a secure passphrase, which is highly recommended. A passphrase adds an additional layer of security to prevent unauthorized users from logging in. To learn more about security, consult our tutorial on How To Configure SSH Key-Based Authentication on a Linux Server.

You should then see the following output:

```
Output
Your identification has been saved in /your_home/.ssh/id_rsa.
Your public key has been saved in /your_home/.ssh/id_rsa.pub.
The key fingerprint is:
a9:49:2e:2a:5e:33:3e:a9:de:4e:77:11:58:b6:90:26 username@remote_host
The key's randomart image is:
+--[ RSA 2048]----+
|     ..o         |
|   E o= .        |
|    o. o         |
|        ..       |
|      ..S        |
|     o o.        |
|   =o.+.         |
|. =++..          |
|o=++.            |
+-----------------+
```

You now have a public and private key that you can use to authenticate. The next step is to place the public key on your server so that you can use SSH-key-based authentication to log in.

Step 2 — Copy the Public Key to Ubuntu Server

The quickest way to copy your public key to the Ubuntu host is to use a utility called `ssh-copy-id`. Due to its simplicity, this method is highly recommended if available. If you do not have `ssh-copy-id` available to you on your client machine, you may use one of the two alternate methods provided in this section (copying via password-based SSH, or manually copying the key).

Copying Public Key Using `ssh-copy-id`

The `ssh-copy-id` tool is included by default in many operating systems, so you may have it available on your local system. For this method to work, you must already have password-based SSH access to your server.

To use the utility, you simply need to specify the remote host that you would like to connect to and the user account that you have password SSH access to. This is the account to which your public SSH key will be copied.

The syntax is:

```
$ ssh-copy-id username@remote_host
```

You may see the following message:

```
Output
The authenticity of host '203.0.113.1 (203.0.113.1)' can't be established.
ECDSA key fingerprint is fd:fd:d4:f9:77:fe:73:84:e1:55:00:ad:d6:6d:22:fe.
Are you sure you want to continue connecting (yes/no)? yes
```

This means that your local computer does not recognize the remote host. This will happen the first time you connect to a new host. Type "yes" and press `ENTER` to continue.

Next, the utility will scan your local account for the `id_rsa.pub` key that we created earlier. When it finds the key, it will prompt you for the password of the remote user's account:

```
Output
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are alre
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to insta
username@203.0.113.1's password:
```

Type in the password (your typing will not be displayed for security purposes) and press `ENTER`. The utility will connect to the account on the remote host using the password you provided. It will then copy the contents of your `~/.ssh/id_rsa.pub` key into a file in the remote account's home `~/.ssh` directory called `authorized_keys`.

You should see the following output:

```
Output
Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'username@203.0.113.1'"
and check to make sure that only the key(s) you wanted were added.
```

At this point, your `id_rsa.pub` key has been uploaded to the remote account. You can continue on to Step 3.

Copying Public Key Using SSH

If you do not have `ssh-copy-id` available, but you have password-based SSH access to an account on your server, you can upload your keys using a conventional SSH method.

We can do this by using the `cat` command to read the contents of the public SSH key on our local computer and piping that through an SSH connection to the remote server.

On the other side, we can make sure that the `~/.ssh` directory exists and has the correct permissions under the account we're using.

We can then output the content we piped over into a file called `authorized_keys` within this directory. We'll use the `>>` redirect symbol to append the content instead of overwriting it. This will let us add keys

without destroying previously added keys.

The full command looks like this:

```
$ cat ~/.ssh/id_rsa.pub | ssh username@remote_host "mkdir -p ~/.ssh && touch ~/.ssh/authorized_keys &
```

You may see the following message:

```
Output
The authenticity of host '203.0.113.1 (203.0.113.1)' can't be established.
ECDSA key fingerprint is fd:fd:d4:f9:77:fe:73:84:e1:55:00:ad:d6:6d:22:fe.
Are you sure you want to continue connecting (yes/no)? yes
```

This means that your local computer does not recognize the remote host. This will happen the first time you connect to a new host. Type "yes" and press `ENTER` to continue.

Afterwards, you should be prompted to enter the remote user account password:

```
Output
username@203.0.113.1's password:
```

After entering your password, the content of your `id_rsa.pub` key will be copied to the end of the `authorized_keys` file of the remote user's account. Continue on to Step 3 if this was successful.

Copying Public Key Manually

If you do not have password-based SSH access to your server available, you will have to complete the above process manually.

We will manually append the content of your `id_rsa.pub` file to the `~/.ssh/authorized_keys` file on your remote machine.

To display the content of your `id_rsa.pub` key, type this into your local computer:

```
$ cat ~/.ssh/id_rsa.pub
```

You will see the key's content, which should look something like this:

```
Output
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAACAQCqql6MzstZYh1TmWWv11q5O3pISj2ZFl9HgH1JLknLLx44+tXfJ7mIrKNxOOwxI
```

Access your remote host using whichever method you have available.

Once you have access to your account on the remote server, you should make sure the `~/.ssh` directory exists. This command will create the directory if necessary, or do nothing if it already exists:

```
$ mkdir -p ~/.ssh
```

Now, you can create or modify the `authorized_keys` file within this directory. You can add the contents of your `id_rsa.pub` file to the end of the `authorized_keys` file, creating it if necessary, using this command:

```
$ echo public_key_string >> ~/.ssh/authorized_keys
```

In the above command, substitute the `public_key_string` with the output from the `cat ~/.ssh/id_rsa.pub` command that you executed on your local system. It should start with `ssh-rsa AAAA...`.

Finally, we'll ensure that the `~/.ssh` directory and `authorized_keys` file have the appropriate permissions set:

```
$ chmod -R go= ~/.ssh
```

This recursively removes all "group" and "other" permissions for the `~/.ssh/` directory.

If you're using the `root` account to set up keys for a user account, it's also important that the `~/.ssh` directory belongs to the user and not to `root`:

```
$ chown -R sammy:sammy ~/.ssh
```

In this tutorial our user is named sammy but you should substitute the appropriate username into the above command.

We can now attempt passwordless authentication with our Ubuntu server.

Step 3 — Authenticate to Ubuntu Server Using SSH Keys

If you have successfully completed one of the procedures above, you should be able to log into the remote host *without* the remote account's password.

The basic process is the same:

```
$ ssh username@remote_host
```

If this is your first time connecting to this host (if you used the last method above), you may see something like this:

```
Output
The authenticity of host '203.0.113.1 (203.0.113.1)' can't be established.
ECDSA key fingerprint is fd:fd:d4:f9:77:fe:73:84:e1:55:00:ad:d6:6d:22:fe.
Are you sure you want to continue connecting (yes/no)? yes
```

This means that your local computer does not recognize the remote host. Type "yes" and then press `ENTER` to continue.

If you did not supply a passphrase for your private key, you will be logged in immediately. If you supplied a passphrase for the private key when you created the key, you will be prompted to enter it now (note that your keystrokes will not display in the terminal session for security). After authenticating, a new shell session should open for you with the configured account on the Ubuntu server.

If key-based authentication was successful, continue on to learn how to further secure your system by disabling password authentication.

Step 4 — Disable Password Authentication on your Server

If you were able to log into your account using SSH without a password, you have successfully configured SSH-key-based authentication to your account. However, your password-based authentication mechanism is still active, meaning that your server is still exposed to brute-force attacks.

Before completing the steps in this section, make sure that you either have SSH-key-based authentication configured for the root account on this server, or preferably, that you have SSH-key-based authentication configured for a non-root account on this server with `sudo` privileges. This step will lock down password-based logins, so ensuring that you will still be able to get administrative access is crucial.

Once you've confirmed that your remote account has administrative privileges, log into your remote server with SSH keys, either as root or with an account with `sudo` privileges. Then, open up the SSH daemon's configuration file:

```
$ sudo nano /etc/ssh/sshd_config
```

Inside the file, search for a directive called `PasswordAuthentication`. This may be commented out. Uncomment the line and set the value to "no". This will disable your ability to log in via SSH using account passwords:

/etc/ssh/sshd_config

```
...
PasswordAuthentication no
...
```

Save and close the file when you are finished by pressing `CTRL + X`, then `Y` to confirm saving the file, and finally `ENTER` to exit nano. To actually implement these changes, we need to restart the `sshd` service:

```
$ sudo systemctl restart ssh
```

As a precaution, open up a new terminal window and test that the SSH service is functioning correctly before closing this session:

```
$ ssh username@remote_host
```

Once you have verified your SSH service, you can safely close all current server sessions.

The SSH daemon on your Ubuntu server now only responds to SSH keys. Password-based authentication has successfully been disabled.

Conclusion

You should now have SSH-key-based authentication configured on your server, allowing you to sign in without providing an account password.

If you'd like to learn more about working with SSH, take a look at our SSH Essentials Guide.

By: Hanif Jetha

Related Tutorials

How To Configure SSH Key-Based Authentication on a Linux Server

How To Ensure Code Quality with SonarQube on Ubuntu 18.04

How To Install and Secure Memcached on Ubuntu 18.04

How To Secure a Containerized Node.js Application with Nginx, Let's Encrypt, and Docker Compose

How to Set Up an Nginx Ingress with Cert-Manager on DigitalOcean Kubernetes

4 Comments

Leave a comment...

Log In to Comment

Choomigo *July 5, 2018*

0 Fantastic tutorial. I was able to follow the instructions and connect to the server using the key-based authentication. One thing though, is this supposed to work for only one client machine?
I used ssh-copy-id without any trouble. I try to follow the same process using a different laptop and i can't connect to the server since it is now asking for a key. I changed the sshd config to allow password authentication but it appears it now defaults to key-based authentication and i'm unable to use ssh-copy-id from the 2nd client (since it doesn't have they key yet). Should be a looking at another process for setting up the 2nd client? I really hope i don't have to manually copy the key...

hjet MOD *July 20, 2018*

0 Thank you for your feedback. You have a couple of options here for adding the second client's public key. You can re-enable password authentication by reversing the steps in Step 4, and then proceed using any of the methods outlined in the guide (`ssh-copy-id` is quickest), or log in to your server using the first client, and copy the key in manually using the last method in Step 2.

Hope this is helpful!!

firevision100 *August 3, 2018*

1 Thanks for this.

Don't we have to save the private key somewhere?

---

mnymic *January 12, 2019*

0 @hjet

I'm just digging my way through the manual, there might be further commentaries, yet here we go.

> ssh-copy-id username@remote_host

nope.no. no way.
more like:

```
 ssh-copy-id -i 666 ***.***.**.xx
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "666.pub"
The authenticity of host 'xxx.xxx.xx.xx (xxx.xxx.xx.xx)' can't be established.
ECDSA key fingerprint is SHA256:xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
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
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is
install the new keys
ak666@xxx.xxx.xx.xx's password:

Number of key(s) added: 1

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

The correct syntax I happened to learn here, however, quite some time was wasted to brute-force the correct flags.

Please edit that code block, trifles like that demotivate a lot.

And thanks for the man, since it is truly monumental :)