Using Multiple SSH ports

Unix Tutorial

This is not the most obvious functionality, hence I decided to share it as a separate post. It's quite easy and perfectly acceptable to specify more than one SSH port for your sshd daemon – useful for debugging or added security (when bound to separate IP addresses).

Adding Extra SSH ports

Simply edit the **/etc/ssh/sshd_config** file and add more port numbers under the existing default port (it's commented out because 22 is used by default):

```
greys@server:~$ sudo vi /etc/ssh/sshd_config
```

Change this:

```
#Port 22
AddressFamily any±
ListenAddress 0.0.0.0
ListenAddress ::
```

to this:

```
Port 22
Port 221
Port 222
AddressFamily any±
ListenAddress 0.0.0.0
ListenAddress ::
```

IMPORTANT: you must uncomment Port 22, otherwise new ports will be the only SSH ports listened on (so SSH port 22 will stop working).

Now restart ssh:

```
greys@server:~$ sudo systemctl restart ssh
```

Confirm each new SSH port

netstat command with grep confirms that all 3 ports are being listened on now:

```
greys@server:~$ netstat -nal | grep 22
tcp        0      0 0.0.0.0:22              0.0.0.0:*
LISTEN
tcp        0      0 0.0.0.0:221             0.0.0.0:*
LISTEN
tcp        0      0 0.0.0.0:222             0.0.0.0:*
LISTEN
```

If we want to, we can even try connecting to a non-standard ssh port like 221 or 222 as per our changes.

Don’t be alarmed about warning:

```
root@server:~# ssh greys@localhost -p 222
The authenticity of host '[localhost]:222 ([127.0.0.1]:222)'
can't be established.
ECDSA key fingerprint is
SHA256:12efZx1MOEmlxQOWKhM5eaxDwJr4vUlLhcpElkGHTow.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '[localhost]:222' (ECDSA) to the
list of known hosts.
greys@localhost's password:
Welcome to Ubuntu 19.04 (GNU/Linux 4.18.0-20-generic x86_64)
```

Hope you enjoy this advice, stay tuned for more!

See Also

- ssh port
- SSH
- ssh command

SSH: Too Many Authentication Failures

Here I was trying to **ssh** from my XPS laptop to MacBook Pro for some quick command, when SSH started giving me the **too many authentication failures error**. I decided to capture findings here as a blog post.

Too Many Authentication Failures

Here's how the error looked from my Ubuntu 19.04 command line:

```
greys@xps:~ $ ssh greys@maverick
Received disconnect from 192.168.1.200 port 22:2: Too many
authentication failures
Disconnected from 192.168.1.200 port 22
```

The weird thing is that this was happening without any passwords asked, so at first it seemed really strange: you get authentication failures but you actually haven't tried authenticating at all.

Why Too Many Authentication Failures Occur

So yes, these errors happen when you attempt to log in using some credentials and you are denied access for a few times in a row due to incorrect credentials.

Something as fundamental as SSH client and server are rarely wrong in such basic things. So thinking about the error a bit more (and Googling around, of course) I realised that authentication attempts were made using SSH keys I have configured on my Ubuntu laptop. There's quite a few and SSH client was offering them one after another to the MacBook's SSH daemon in attempts to log me in.

So I never got asked for a password because my SSH client already offered a few SSH keys and remote SSH server counted each offering as an authentication attempt. So when this maxed out the SSH server limit, I got the error.

MaxAuthTries Setting

Related to the error above is this **MaxAuthTries** setting in **/etc/ssh/sshd_config** file.

This option is set to something fairly reasonable usually, in MacOS Mojave it's set to 6 by default. But because I changed it to 3 in the past for better security, it limited my access when my SSH client was offering more than 3 SSH keys to log in.

Working around the Too Many

Authentication Attemps Problem

There’s a number of approaches, all of them to do with SSH identities used for remote access. They are managed by SSH agent, a special software usually starting automatically with your laptop login that tracks all the usernames and SSH keys you have to try them when accessing things remotely.

Disable SSH agent temporarily

So the easiest fix is to disable SSH agent temporarily and try login again (for password logins it does the trick).

I’ll quickly show the steps but will need to write a separate proper post on **using SSH agent** usage soon.

Step 1: we check user variables for SSH_AUTH_SOCK

This variable will usually confirm if you have SSH Agent. If this variable exists and points to the valid file, that’s the Unix socket used by your SSH agent:

```
greys@xps:~ $ env | grep SSH
SSH_AUTH_SOCK=/home/greys/.ssh/ssh-auth-sock.xps
SSH_AGENT_PID=1661
```

Step 2: we reset the SSH_AUTH_SOCK variable

Let’s set this variable to empty value and check it:

```
greys@xps:~ $ SSH_AUTH_SOCK=
greys@xps:~ $ env | grep SSH
SSH_AUTH_SOCK=
SSH_AGENT_PID=1661
```

That’s it, now logins to Macbook laptop should work again:

```
greys@xps:~ $ ssh greys@maverick
Password:
Last login: Wed Jun 12 12:31:33 2019 from 192.168.1.60
greys@maverick:~ $
```

That’s it for today! Quite a few advanced topics in just one post, so I’ll be sure to revisit it and expand with further posts on the concepts of **SSH ports**, **SSH agent**, **passwordless SSH** and generating **SSH keys**.

See Also

- SSH
- Common SSH settings
- ssh port
- SSH port forwarding
- Check SSH port connectivty