OpenSSH Tutorial

(Linux/Mac/Windows(WSL/MobaXTerm/VirtualBox))

Table of Contents

ssh Client Basics and Simple Login

Help Yourself!

Open a terminal! Enter (to get out, pressing ‘q’)

```
> man ssh
```

```
> ssh -V
OpenSSH_7.9p1 Debian-10, OpenSSL 1.1.1d  10 Sep 2019
```

When you need debugging during the connection attempt:

```
> ssh -v
```

goes up to `-vvv` for more verbosity.

Simple Password-Based Login

Next, try to login! For the tutorial, the following login nodes are available:
`lxlogin1.lrz.de`, `lxlogin2.lrz.de`, `lxlogin3.lrz.de`, `lxlogin4.lrz.de`

`lxlogin8.lrz.de, lxlogin10.lrz.de`

Which one you pick, does not matter! You also got a User ID (we use `<userID>` as placeholder; please replace it by your own ID!) and a corresponding password.

Try a simple Login like this:

```
> ssh -l <userID> lxlogin1.lrz.de
```

or

```
> ssh <userID>@lxlogin1.lrz.de
```

At the first login there you will probably be asked something like this:

```
The authenticity of host 'lxlogin1 (129.187.20.105)' can't be
established.

ECDSA key fingerprint is
SHA256:YmTuVciNdQzoZXpiDC4encMuUa8WIjJuA4NqmXaXgeM.

Are you sure you want to continue connecting (yes/no)?
```

Essentially, that's an authentication message from the remote server. You should not simply answer 'yes'! You should look the LRZ docu page, where these keys are trustworthy (the correct keys), in order to avoid man-in-the-middle attacks. These key fingerprints are stored inside your local `$HOME/.ssh` folder (`~/.ssh/known_hosts`).

Next, you should be asked for a password. This is that password you got for your userID! Enter this password!

To be sure that you are really remotely working, issue:

```
> hostname
```

If it is your own Laptop, you were not successful! Try again! If failing again, call the course instructor!

On ***success***[1], log out again! Either by

```
> exit
```

or by pressing '`Ctrl+D`'.

Sometimes useful options are: `-p` to change the SSH port; `-C` for compressing the SSH communication.

X-Forwarding

If you have a running X-server (under Windows, this might be Xming for MobaXTerm for instance; WSL is more difficult to accomplish!), you can try simple graphics forwarding. Login in again, using one of the option `-X` or `-Y`:

```
> ssh -X <userId>@lxlogin1.lrz.de
```

or

1 `hostname` will show something like `cm2login1`. That's the internal name. We will use it later!

```
> ssh -Y <userId>@lxlogin1.lrz.de
```

The `-Y` option is usually more user-friendly. On ***success***, you should now be able to open a graphical program! For instance, start the PDF viewer *evince*:

```
> evince
```

or

```
> module load paraview
> paraview
```

(Firefox might not work, because it is too clever and opens YOUR LOCAL Browser, if present!) When you finished. Close the graphical window, and log out!

SSH Hopping and ProxyJump

Via SSH, one can connect also to other (SSH server) hosts, which are e.g. behind a firewall (such as ours, represented by the login nodes). Due to lack of access for you accounts, we simulate this by (mis-)using the login nodes. So, we try to login to `lxlogin2` (alias `cm2login2`) through `lxlogin1.lrz.de`. At its simplest, we could accomplish this by the following.

```
> ssh <userID>@lxlogin1.lrz.de
cm2login1> ssh cm2login2
cm2login2>
```

Of course, you now need to enter the password twice. And when you wish to logout, you now must also do it twice – once from `cm2login2`, and once from `cm2login1`. Indeed, so login to `cm2login2` does not require a `<userID>@`, because the current user is used.

This can be done a bit shorter.

```
> ssh <userID>@lxlogin1.lrz.de -t ssh cm2login2
```

Logout is only necessary once, now. No actual login (with a shell etc.) is done on `lxlogin1`. If you like, you can try to extend this – let's go on to `cm2login3 (aka lxlogin3)`.

```
> ssh <userID>@lxlogin1.lrz.de -t ssh cm2login2 -t ssh cm2login3
```

Newer versions of OpenSSH have the so-called 'ProxyJump' option `-J`, which will prove much more convenient, when dealing with SSH tunnels and port forwarding. Here the same example as above (`lxlogin1.lrz.de → cm2login2 → cm2login3`).

```
> ssh -J <userID>@lxlogin1.lrz.de,cm2login2 cm2login3
```

It might be astonishing that we can specify the address `cm2login3`, although it is not visible and reachable from outside (you could test it via `ping cm2login3`). SSH knows but how to resolve it.

Copying Data

scp Client

The OpenSSH suite provides also a client for copying files and folders in a secure fashion. This command-line tool is not always easy to use for beginners, as long as they cannot imagine the structure of local and remote file systems.

As preparation, let's create a local folder and a file inside of it.

```
> mkdir folder && touch folder/file1
```

Without changing the folder, simply execute

```
> scp folder/file1 <userID>@lxlogin2.lrz.de:
```

Please note the colon at the end! It is important. When omitted, you would copy `file1` into a local file named `<userID>@lxlogin2.lrz.de`. To check your success, login to `lxlogin2.lrz.de`, and execute there

```
> ls
file1
```

You can also rename the copied file in the course of copying.

```
> scp folder/file1 <userID>@lxlogin2.lrz.de:file2
```

creates a file named `file2` in the `HOME` directory of your remote account.[2] You can also copy a whole folder. Option `-r` means 'recursive' (also including subfolders would be copied in this course).

```
> scp -r folder <userID>@lxlogin2.lrz.de:
```

Now the complete folder `folder` including the file `file1` should be in your remote `HOME` directory.

You can also copy back from somewhere to your local place.

```
> scp <userID>@lxlogin2.lrz.de:file2 .
```

And similar with folders, etc. Practice a little bit to copy around folders and files until you feel confident to do it correctly. You always can check your success by logging in.

Filezilla

Filezilla is a GUI based SSH client, available for Linux, Mac (Apple), and Windows. Figure 1 shows the client after opening. In the menu line, you must enter the server name to which you want to connect, the userID, the password, and port 22 (usually).

After successful connection, you can copy files and folders via drag-and-drop. This is surely more convenient than SCP.

---

2 The `HOME` directories of your accounts are in a so-called network-shared file system, meaning that they are the same (by content) on each of the login (and actually all cluster) nodes.

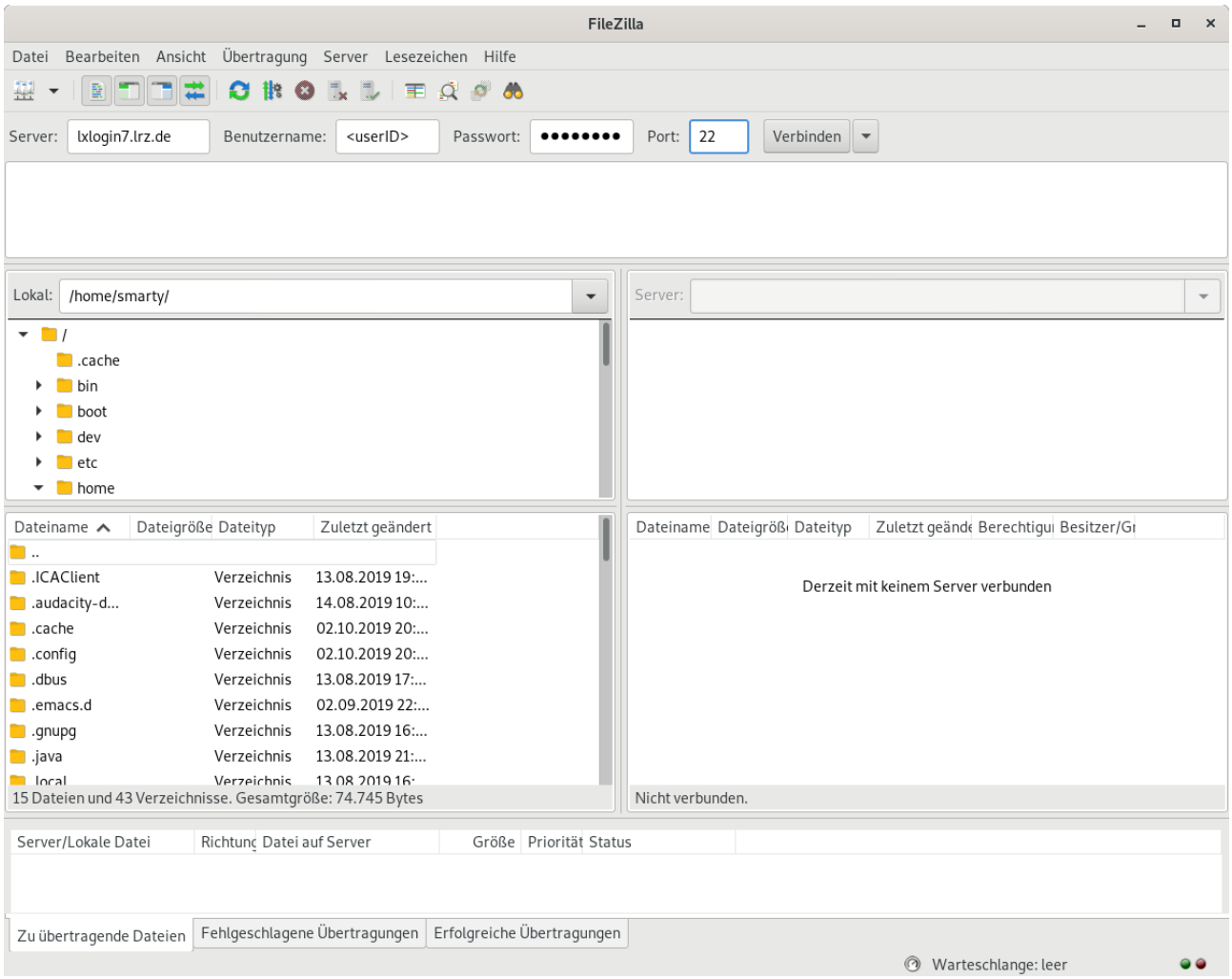

Figure 1: Filezilla SSH Client as Drag-and-Drop File Manager

sshfs

Under Linux, and possibly also Mac (and who knows where else it may work), there is a tool – sshfs – which can be used to *mount* remote file systems locally. This means that it finally appears as if the remote file system were local. The advantage of sshfs is that it works in user space – no root privileges are needed!

First, one needs to create a mount point – a folder, where the remote file system is mounted to.

```
> mkdir mnt
```

Next, we mount the remote `HOME` directory.

```
> sshfs <userID>@lxlogin4.lrz.de mnt
```

Now you can simply cd into `mnt` and issue e.g. `ls`.

```
> cd mnt && ls
file1   file2    folder
```

To unmount the folder, and thus release the SSH connection, leave the folder, and issue `fusermount`.

```
> cd .. && fusermount -u mnt
```

The `mnt` folder could now be removed. It should anyway be empty.

Public Key Authentication

Up to now, we authenticated only via our userID and the corresponding password. This method bears several problems – among others that observers might espy (maybe even only accidentally) your credentials while you hack them into your keyboard. Or if you are inattentive, you might accidentally write it to the bare screen – readable for every eye.

Finally, if you are logging in and out very frequently, possibly over several hosts, and maybe also as different users on different hosts with each having a different password, entering credentials always by hand will drive you crazy sooner or later. Latest now, you should wonder whether a security feature should not be more convenient to useful! And the answer is, of course, 'Yes!'.

Using SSH public-private key pairs is the answer. Their checkout and usage can be automated (looks like password-less login). You don't need to remember all the passwords (except for the cases, where you really need them!). SSH keys are really long. And so much more secure than your password can possibly ever be! And even if some person acquires your SSH key (specifically the private one), he/she cannot guess your password itself – and he/she has only access to the system for which this key was generated. Once you have the suspicion that your account was hacked in this way, just disable your key! You have the full control over your account and the keys – you can do hardly anything wrong!

Finally, there are some systems like Cloud VMs, which can only be access via SSH keys. Here, you even have no other choice!

There are two stages. 1) Creation of keys. 2) Distribution of Keys. 3) Using the keys. For the last to be convenient, we introduce the concept (and tool) of an SSH agent.

Creation and distribution of keys you usually does only seldom. Of course, you could reuse a created key pair for several independent remote hosts (with possibly different user IDs). However, from a security point of view, you should not do this.

Key Generation

To create a key, we use the tool `ssh-keygen`. To remember the command-line options, use `ssh-keygen --help` or `man ssh-keygen`. In its simplest form, just execute the following.

```
> ssh-keygen -t rsa
```

or, if you want more security,

```
> ssh-keygen -t rsa -b 4096
```

Via option -f, you can also determine the key file location and name. But for the sake of this tutorial stick with the defaults! This means (most probably) that files are created inside your `HOME` directory in `.ssh/` – named `id_rsa` (private key[3]) and `id_rsa.pub` (public key).

Once ssh-keygen is started, you get some dialog to follow and answer.

```
Generating public/private rsa key pair.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
```

3 This file should NEVER EVER leave your HOME directory! Also, never change its access permissions!

```
Your identification has been saved in ~/.ssh/id_rsa.
Your public key has been saved in ~/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:Ax7HukEtcb07XVMCkGY8hdNMWnlqzHqxRVPy4YzgNb4
The key's randomart image is:
+---[RSA 4096]----+
| ..o.X==.o.|
|  = @oB Xoo|
|= +o.B * *.|
| o * . * = |
| + S = E . |
|  o   .+ + |
|.       o  |
+----[SHA256]-----+       # ls -l .ssh → id_rsa id_rsa.pub
```

You are asked for a 'passphrase' (twice – once for confirmation). This is NOT any password you already have! It is a new passphrase you should enter. But in contrast to your SSH password, it does not need to be all too strong. Its only purpose is to protect your private key! So, please, don't leave it empty![4] Some basic notion of security must be there in all this convenience! And don't reuse it! (see Bruce Schneier's acticle on creating good passwords/passphrases.)

Next to RSA, you can also choose other keys like ECDSA or ED25519. If servers (for security reasons) do not accept certain encryption method anymore (because, they might be compromised by newer computer development), then you must change to a new/different algorithm. This also means that you should keep your SSH suite up-to-date.

Key Distribution

The next step is to bring the public (!) key to the system you later want to connect. For our tutorial, this is any of the `lxlogin*` nodes. So, issue

```
> ssh-copy-id <userID>@lxlogin2.lrz.de
```

If your public key file got a different name, you must explicitly specify it via

```
> ssh-copy-id -i <public key file> <userID>@lxlogin2.lrz.de
```

If `ssh-copy-id` is not available on your system, copy the content of your public key file into the remote `$HOME/.ssh/authorized_keys`. For this, you must then of course login to the remote host! Go sure to use a correct editor (ASCII), and not to include strange or formating extra characters!

Key Usage

Once your keys are created and correctly distributed, you can test the access. Without anything else, try to login to a `lxlogin*` machine! Instead of being asked for a password, you are asked for a passphrase matching to a private key (`~/.ssh/id_rsa` by default). Just enter the passphrase you set for your key!

4 Keys used to access the LRZ systems from outside MUST be passphrase protected (that's Policy)!

If your pressed enter without entering the passphrase, the next authentication method is used – and your are asked for your account's password (according to your user ID). This fallback mechanism is intended for the case, your keys do not work any more! But if you also forgot this account password, you need external help ;)

Okay! So far, we have effectively only exchanged one password by one passphrase! Where is the convenience? This comes in form of a SSH-agent! This is kind of a daemon that must run in the background. On modern systems, such an agent already runs per default, and you can use it. To check, whether it runs, issue

```
> ssh-add -l
```

If you get a message about that it could not be connected to any SSH agent, or that there does not appear to run such an agent, just start this agent by hand.

```
> eval "$(ssh-agent -s)"
```

It is important to start it that way such that you can connect also from other terminals.

You can add a key via

```
> ssh-add
```

or

```
> ssh-add .ssh/id_rsa
```

or any other possible private key file. You are asked for the passphrase. Enter it! Correctly done so, you should now be able to login to `lxlogin2.lrz.de` without any password or passphrase. Please try!

You can also remove a key again from the agent via

```
> ssh-add -d .ssh/id_rsa
```

or

```
> ssh-add -d
```

if you there is only the default key, or simply

```
> ssh-add -D
```

to remove all keys registered with the agent (for clean up, sometimes a good idea).

Troubleshooting and Recovery

If you fear that your key was compromised, your you simply forgot the passphrase, and want to start anew, it does not suffice to delete the public and private key files (inside `~/.ssh`) and from the agent (`ssh-add -D`). You must also remove the corresponding public key from the remote `.ssh/authorized_keys`!

Exercise: Create keys, and remove them again. Some times should suffice to gain a feeling how it works!

Port Forwarding / SSH Tunneling

The last part deals with some more advanced, and really cool stuff. SSH connections admit it to tunnel through them other protocols like HTTP or VNC (if this sounds familiar to you). Most beginners with SSH don't like to work through a terminal or in a shell. The following may help here to mitigate the first pains.

For the following to work, you must remember what ports are (for what they are useful), and that only one server on a system can acquire a port – and block it from the usage by others. And all ports below 1024 are previliged ports – users can't use them! All other ports up to 65535 can be used.

Preparation

You can use any (or all) of the following examples. If a port is blocked, use another one! Login to one `lxlogin*` node and start one of the following servers.

1) ***Webbrowser: (requires a local Browser)***

```
> module load python
> cat > index.html << EOF
<html>
   <head>
      <title>Hello LRZ!</title>
   </head>
   <body>
      <h1>Welcome to my Webpage!</h1>
   </body>
</html>
EOF
> python -m SimpleHTTPServer 11111
Ctrl+C       # kill the server when finished the exercise
```

2) ***Jupyther Notebook: (requires a local Browser)***

```
> module load python/3.5_intel
> jupyter notebook --port=11111 --no-browser –ip=localhost
The Jupyter Notebook is running at: http://localhost:11111/?
token=3eef5acb44b47f385be6dcdd341b319d4d13fdc3879c6c9f
Ctrl+C       # kill the server when finished the exercise
```

3) ***VNC Server: (requires a local VNC Viewer)***

```
> vncserver -list     # check empty ports
> vncpasswd           # choose a simple password!
> vncserver :5        # for port 5900+5=5905
> vncserver -kill :5  # kill server when exercise finished
```

Next, once one of your servers is running, we try to connect to it. Keep the server running during this time. Remember on which ***host*** and on which ***port*** your server runs! We need this information!

Tunnel and Connection to the same Server

The first scenario is the simpler one. We connect to the SSH Server, where also the HTTP/VNC Server is running. This is maybe the more often case, as `lxlogin*` servers are the login servers, where you may also run such smaller applications (for the larger ones, the Slurm Cluster is dedicated!).

Open a new shell locally and connect to the host you have the server running on with the following command – here, we assume this to be `lxlogin1.lrz.de`.

```
> ssh -L 12345:localhost:11111 <userID>@lxlogin1.lrz.de
```

Here, `12345` is a arbitrary local port.[5] As remote port, we specified here `11111`. Please select the port, you chose before for your server (e.g. `5905` for the VNC server above)!

Depending on the example chosen do the following.

1) Open a Browser with `http://localhost:12345`
2) Open a Browser with the webaddress shown on the servers output, but port `11111` replaced by the local port `12345`. `http://localhost:11111/?token=3eef5acb44b47f385be6dcdd341b319d4d13fdc3879c6c9f` Specifically the token (you have to take your own token, of course) is important to get access to your server!
3) Open the VNC viewer with `localhost:12345`. You must enter the VNC password.

The usage of browsers and VNC is not part of this course. But I'm sure you already recognize what you can use it for.

When finished, let the servers still running, but close the Browser or VNC viewer, respectively, for the next exercise. Also close the SSH connection with the SSH tunnels (in order to release the local for something new).

Tunnel and Connection to different Servers

This final excercise gets now challenging for the beginning. But I hope you will value it somewhen later. We want to guide a SSH Tunnel through a login host (firewall) to another server (where our application server is running). We again simulate this by using different login host of the LRZ Linux cluster.

This exercise is made SSH clients that support ProxyJump. Of course, also for older clients is it feasible to perform such a tunnel forwarding. But why suffering when simply upgrading?!

Imagine our application server is still on `lxlogin1.lrz.de`. You must use your host! We login now through a different host, say `lxlogin2.lrz.de`, and tunnel internally to `cm2login1` with the correct port.

```
> ssh -L12345:localhost:11111 -A -J <userID>@lxlogin2.lrz.de cm2login1
```

5 Must be also unique. If you have several tunnels open, all their local ports must be different! It is sometimes hard to keep an overview. In the worst case, close all tunnels and SSH connections, and start from the beginning!

Remember to use your correct hosts and ports! `-A` makes agent forwarding, meaning that the SSH keys you use are forwarded also for the second host, such that no password request pops up.

The rest of the previous section stays the same!

Another possibility without the ProxyJump works if the very remote host has not the ports blocked (as in this case with `cm2login1`, such that we are forced to login in order to access them).

```
> ssh -L12345:cm2login1:11111 <userID>@lxlogin2.lrz.de
```

As mentioned before, this will not work! If you want to test this case, please ask such that I can setup some server for you on another node, such that you can exercise this mode!

This last tunnel shows that your server is not really protected by itself. SSH only protects the communication. But any other user with access (permission rights) to the system can also setup a tunnel yo your server! Keep this in mind when running a service (use VNC passwords, use tokens for Jupyter Notebook, etc.)!

Please, stop all servers on the login node again!

Appendix

Useful/Entertaining Links

- https://www.schneier.com/blog/archives/2014/03/choosing_secure_1.html
- http://cryptocouple.com/
- http://web.mit.edu/jemorris/humor/alice-and-bob

Advanced Usage – SSH shell

OpenSSH also has a shell mode. You can enter it in different ways, depending on what you want to do. In order to activate it, you must press ~ in such a way that this character does not appear in the shell (possibly you must press enter before trying again; or even setup your terminal shell correctly before).

`~?` shows the SSH shell help

`~C` opens the command line shell; once entered, you can issue `-L12345:localhost:11111` to attach a tunnel to the current SSH connection

`~.` interrupts the current (possibly hanging) SSH connection

SSH local Configuration

In order to simplify life even further, you can make permanent settings inside `~/.ssh/config`. For instance, look at the following config file.

```
ForwardX11 no               # same as ssh -x (opposite of -X)
ForwardX11Trusted no        # same as ssh -x (opposite of -Y)
ForwardAgent yes            # same as ssh -A
```

```
Compression yes             # same as ssh -C
ServerAliveInterval 300     # when problems with connection
User <userID>               # default user name
Host lxlogin1 lxlogin2 lxlogin3 lxlogin4 lxlogin8 lxlogin10
    HostName %h.lrz.de      # parametric for all listed hosts
Host LRZ_GW                 # alias
    HostName lxlogin1.lrz.de
Host rvs2                   # alias for host via a Gateway
    HostName rvs2.cos.lrz.de
    ProxyJump LRZ_GW
```

For more information, please consult `man ssh_config`.

Outlook for real Expert Candidates

That’s not part of this course!

SSH allows for several keys you can have – possibly for each remote system/userID one.

SSH configuration allows also to assign the private key to a host/userID.

SSH allows also a public key restricted execution of certain programs. So, another user could do certain tasks in your name, when he/she has the corresponding private key. But you should seriously test this before exposing your account! This feature might really deteriorate the security of your account, when done careless!

Some systems like SuperMUC-NG do not allow arbitrarily outgoing connections to the Internet. For instance, `git` or `wget` will not work. But this can be circumvented by using so-called *reverse tunnels*. But with `sshfs`, we showed you an alternative!

Don’t use option `-g`, unless you really must and know what you are doing! There are reasons to use SSH. This option might completely negate them!