# Introduction to SSH – Worksheet

## Table of Contents

## 1 Find an SSH Client

1) Check, which SSH Client you have (or prefer) for your operating system! If none is there, install one!

   a) Linux: *sudo apt install ssh* (Debian, Ubuntu, …)
      or use the package manager of your choice!

   b) Mac OS: package manager

   c) Windows 10: OpenSSH should be installed[1] (cmd);
      if not, there are alternatives:
      Windows Subsystem for Linux (WSL), MobaXterm (private variant), Virtual Box – all are effectively Linux with OpenSSH,
      or,
      PuTTY, KiTTY, … (SSH GUI clients)

      Recommendation: Try the native OpenSSH first; if not working or unpracticable try PuTTY (suite) next

2) Figure out the version of your client! Figure out the help information from that client (at least, where to find it)! (Hint: Internet is also a viable option!)

## 2 First Login and Connection Setup Management

1) Using your credentials (user name and password), login to one of the Linux cluster login nodes (lxlogin?.lrz.de, where ? =1,2,3,4)!
   Do NOT accept the fingerprint! Use the doku.lrz.de page to find the SSH server keys (“Linux Cluster Secure Shell Public Keys”), and fill them into you local *.ssh/known_hosts* (OpenSSH)! [Check, where PuTTY is storing this information under Windows!]
   Check with commands, *whoami*, *hostname*, *pwd* that you are working remotely!
   Log out again!

2) For OpenSSH: Create a *config* file in the *.ssh* folder (dot is not a typo!) of your local system, and create a connection shortcut that contains an alias name, user name, and the remote host

1 Not all features of OpenSSH are possibly working with the Windows OpenSSH installation!

name!
For PuTTY: Setup a complete connection inside the GUI (except for the password), and save it!

Try to connect via this saved connection! (Idea: It should be easier, faster, less typing/clicking → convenience)
Hint: editing of the *config* file must be done using an editor like *jedit*, *notepad++*, *emacs*, *vi*, etc. Please, don‘t use a word processor!

## 3 Copying Data

1) Create some folders and files, and copy them to and fro the Linux cluster login nodes (into the HOME directory)! Check the success of that operation by login via SSH and *ls*, *cd*, *cat*, etc.

Hint: As client, you can use:

OpenSSH (command line): *scp*, *sftp*, *rsync*, *sshfs* (Linux only; mounted remote file system)

PuTTY (command line): *pscp*

GUI-based: Filezilla, WinSCP (Windows only)

or, any alternative you know!

Recommendation: Try *scp* first! Try also *Filezilla* as GUI alternative!

Fun-Fact: *ssh* (the OpenSSH client) allows you to copy directly. Under Linux:

```
> mkdir test_folder && touch test_folder/test.txt
> tar cvjf - test_folder | ssh lxlogin1 "tar xfj -"
(same as    scp -r test_folder lxlogin1:)
> ssh lxlogin1 ls test_folder
test.txt
```

## 4 Public Key Authentication

### 4.1 Key Pair Creation and Distribution

This actually only needs to be done once (in a while). But you can always remove the keys again and start out from the beginning, for practicing.
**Always set and remember the (non-empty) passphrase for the keys pairs you create! Otherwise, your keys are not usable!**

OpenSSH:

1) Using *ssh-keygen*, create a 521 bit ECDSA key pair!

2) Using *ssh-copy-id*, copy the public key (ending with *.pub*) to the remote server!

Where is the public key finally copied to? Is the following command equivalent to the *ssh-copy-id* above?

> *> cat ~/.ssh/id_ecdsa.pub | ssh "cat >> .ssh/authorized_keys"*

How else could you copy the public key to the remote server?

PuTTY:

1) Using PuTTYgen, create a 521 bit ECDSA key pair! Save the private one, and copy the public one into the remote *~/.ssh/authorized_keys*!

## 4.2 Key Pair Usage

1) Try to login again! Observe what the SSH client now requires from you! (Hint: If it is not the passphrase, something is wrong! Check that you correctly created the keys, and that you copied the public key to the remote *.ssh/authorized_keys*! In worst cases, check also the *.ssh* folder access permissions! If nothing helps, clean up the local and remote *.ssh* folders, i.e. remove the keys again!)
2) OpenSSH: Check that an *ssh-agent* is running! (*ssh-agent -l* or, *ps aux | grep ssh-agent*)
   If it's not running, try *eval 'ssh-agent -s'*! Take care that at most one such agent is running!
   PuTTY: Start *Pageant*! Check that the tray icon for pageant is visible (computer with a hat).
3) If the SSH agent/Pageant is running, add the ECDSA key created about! Hint: You should be asked for the passphrase!
   Login to the remote server again! You should not need any passphrase or password anymore for this and any subsequent login!
4) With *scp*, i.e. copying files and folders, the key opened in the SSH agent should also remove any extra password/passphrase requirements.

# 5 Port Forwarding / SSH Tunnels

1) Create an SSH connection with tunnel from the login node to your local machine!

For this exercise, login to the Linux cluster, and start a Jupyter Notebook[2] via:

```
remote> module load python/3.8.8-extended
remote> pip install jupyter --user
remote> jupyter-notebook --port=11111 --no-browser --ip=127.0.0.1
    […]
    Copy/paste this URL into your browser when you connect for the first time,
    to login with a token:
    http://127.0.0.1:11111/?token=6160202fb83e5e0cf43a8573761a9cc67d4152f685ad89df
```

The port 11111 might not work. Then, please choose another arbitrary one greater than 1024 (smaller than 64k)!

For an explanation: jupyter starts on the remote login node a web server, which listens on port 11111. If you were to start a browser there, you could simply click on the link presented (btw. the

2 The usage of Jupyter is not part of this course part! It's only an illustrating example!

token is unique, and is there for protecting your jupyter session – each session will have a new token). But unfortunately, you can only start a browser on your local machine, from which you cannot forward any http requests directly to the remote login node – because there is a firewall preventing this.

The simple solution is to redirect the http request through an SSH tunnel. Therefore, setup an SSH connection with a port forwarding tunnel (option *-L* for OpenSSH; "Connection → SSH → Tunnels" for PuTTY (do not forget to press "add"!)). The right-hand side is 127.0.01:11111 (or the port you used above), and the left port can be any arbitrary user port – confusingly, but for simplicity, you can again use the same port number as for the remote port, unless it is already occupied.

Opening now the link presented by the jupyter notebook in your local browser should work! Please, try!

Remark: Instead of the IP address 127.0.0.1, you will often see *localhost*. If everything is setup correctly, both are synonym! On each system, 127.0.0.1 or localhost point to the machine itself. So, opening the web link actually will try to connect to a local(!) web browser at the port specified by you on the left (source port) side! But as there is only the SSH tunnel waiting, this request is forwarded to the remote web server.

2) Actually, you don't need to open a new SSH connection with a tunnel. You can also attach (almost arbitrary many) tunnels to the existing SSH connection. That's done using the escape sequence ~C (Both characters should not be seen in the shell window! If they are seen, press twice <Enter>!). This will bring you to the SSH command shell, prompted by:
   ssh>
   You can now issue "? <Enter>" to get some help. Or, you can setup the tunnel by entering
   ssh> -L11111:localhost:11111 <Enter>
   (Remember: Left port is the local source port; localhost:11111 designate the remote machine and the remote destination port!)

   With PuTTY, the same can be achieved by the session windows pull down menu "Change Settings…", and then "Connection → SSH → Tunnels".

   Please, also try this option for an SSH tunnel setup! Always check the correctness of your tunnel via calling the jupyter web address in a web browser!

3) [Advanced] Can you imagine, what for such tunnels can be used, too? If you need to call remotely some GUI program, you can for instance use VNC. (Do not forget to stop the vncserver, when you finished: *vncserver -kill :<display number>*)