w1_ssh

SSH

Secure shell (SSH) is a protocol to allow you to remotely log in to another computer.

`ssh` is the client, which you run on your machine to connect to another machine.

`sshd` is the server, or *daemon* in UNIX-speak. It runs in the background on the machine you want to connect to, and needs to be installed by the system administrator. Note: SSH uses TCP port 22 by default.

Check your client

Type `ssh localhost` and press ENTER. Several different things could happen:

- If it asks for a password, then the ssh client is working, and a ssh server is running on your current machine. The password would be your user account password, but we don't actually want to log in again so cancel with Control+C.
- If it succeeds without a password, then the client is working and a ssh server is running on your machine and either you do not have a password, or you already have a key set up. Type `exit` and press ENTER to get back to your previous shell.
- If it shows "connection refused", then you have the ssh client correctly working but no server running on your own machine. This is not a problem, as we're trying to log in to the lab machines, so we need a client on our machine and a server on the lab machine.
- If it shows an error that ssh is not found, then you don't have (Open)SSH installed which is very unusual except on windows CMD - in which case please switch to using the windows subsystem for linux.

Connect to the lab

`$ ssh [USERNAME@]HOSTNAME`

The bastion host `seis.bris.ac.uk`. This is reachable over SSH from the internet, and is on a university network that lets you connect further to the lab machines. You should not attempt to do any work on seis itself, as most of the software you would like to use (like compilers) is not installed there. However, you do have a home directory on seis for storing things like SSH keys.

The load balancer `rd-mvb-linuxlab.bristol.ac.uk` connects you to a lab machine.

Prompt: `USERNAME@it######:~$`

Type `ssh USERNAME@seis.bris.ac.uk` . Command `uname -a` to print information about the system. Try `whoami` and `uname -a` to check who you are logged in as, and where; also try `hostname` which just prints the machine name.

Jump:

```
ssh -J USERNAME@seis.bris.ac.uk USERNAME@rd-mvb-linuxlab.bristol.ac.uk
```

Setting up ssh keys

The keys that SSH uses implement digital signatures. Each key comes as a pair of files:

A private key (also known as secret key) in a file normally named `id_CIPHER` where CIPHER is the cipher in use. You need to keep this secure and only store it in places that only you have access to.

A public key in a file normally named `id_CIPHER.pub`. You can share this with the world, and you will need to store a copy of it on any machine or with any service that you want to log in to (for the lab, because the lab machines all share a file system, you only need to store it once - but seis has a separate file system so you need a separate copy there).

Generate keys: command `ssh-keygen -t ed25519` .

On your own machine: `ls -l ~/.ssh`

```
-rw-r--r--.  config
-rw-------.  id_ed25519
-rw-r--r--.  id_ed25519.pub
-rw-r--r--.  known_hosts
```

~/.ssh/authorized_keys and ~/.ssh/id_ed25519.pub:

`ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIDxt6V4EEZ+7knCtSjsSYAtomEsH2WStEOQTE2JlwlHL saquantum@localhost.localdomainssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIHQarE6bKBCgqhXXPyGPWRJaf8I8JZNhwHHfmNQQxFQz soyo@DESKTOP-PF2B1AC`

Set up key access on SEIS

First, we need to upload our public key to the `~/.ssh` directory on seis. Even before this, we need to make sure the directory exists though:

- Log in to seis with ssh and your password.
- Try `ls -al ~/.ssh`. If it complains the folder doesn't exist, create it with `mkdir ~/.ssh`.
- Log out of seis again with `exit`.

Run this from your own machine:

```
scp ~/.ssh/id_ed25519.pub "USERNAME@seis.bris.ac.uk:~/.ssh/"
```

This will ask for your password again. Note two things here: first, to set up access on seis, we are uploading the public key - not the private key! - and secondly, that we put double quotes around the destination. This is because the `~` character meaning home directory is handled by our shell, but we don't want our local shell to expand it, instead we want the shell on seis launched by scp to expand it to our home directory on that machine.

Now log in to seis over ssh and type your password one last time. Then run the following:

```
cd .ssh
cat id_ed25519.pub >> authorized_keys
chmod 600 authorized_keys
```

Setting up keys for lab machines

To connect from your machine to seis, you need a private key on your machine and a public key on seis. To connect from seis to a lab machine, it would seem like you need a public key on the lab machine and a private key on seis. You do not want to upload your private key to seis though for security reasons, so instead we are going to use a SSH feature called *agent forwarding* which means that if you SSH into one machine, then when you try and SSH further into another machine SSH will reuse the same key. The way to do this is to use the `-A` command line flag.

To set this up:

- Log in to seis with `ssh USERNAME@seis.bris.ac.uk`. You should not need a password anymore.
- Log in to the lab machines with `ssh rd-mvb-linuxlab.bristol.ac.uk` and enter your password. Check that the `~/.ssh` folder exists and create it if it doesn't, as you did before on seis, then `exit` again to seis.
- Copy your public key file from seis to the lab machines with `scp ~/.ssh/id_ed25519.pub "rd-mvb-linuxlab.bristol.ac.uk:~/.ssh/"`. This will ask for your password again.
- Log in to a lab machine with `ssh rd-mvb-linuxlab.bristol.ac.uk` and enter your password one last time. On the lab machine, install the public key with the following:

```
cd .ssh
cat id_ed25519.pub >> authorized_keys
chmod 600 authorized_keys
```

- Log out of the lab machine and seis again by typing `exit` twice.

The steps above were necessary because your home directory on seis is not the same as on the lab machines.

From now on, from you own machine, you should be able to get directly into a lab machine with the following command, which should not ask for your password at all:

```
ssh -A -J USERNAME@seis.bris.ac.uk USERNAME@rd-mvb-linuxlab.bristol.ac.uk
```

Setting up a configuration file

SSH reads two configuration files: one for all users at `/etc/ssh/ssh_config` (`/etc` is where POSIX programs typically store global settings) and a per-user one at `~/.ssh/config`.

Create `~/.ssh/config` on your own machine:

```
Host seis
  HostName seis.bris.ac.uk
  User USERNAME

Host lab
  HostName rd-mvb-linuxlab.bristol.ac.uk
  ProxyJump seis
  User USERNAME
```

Running vagrant

- Open a terminal in the folder containing the Vagrantfile.
- Run the command `vagrant up`. This starts the virtual machine configured in the current folder, and if it has not been downloaded and provisioned yet (as is the case when you run `up` for the first time) then it does this for you as well.
- When Vagrant tells you the machine is running, run `vagrant ssh` to log in to your virtual machine. If it asks you for a password, use `vagrant`.
- You should now see the virtual machine prompt `vagrant@debian12:~$`. Try the command `ls /` and check that there is a folder called 'shared' in the top-level folder, along with system ones with names like `usr` and `bin`.
  There are two kinds of errors you might get during `vagrant up`:
- If vagrant complains that it can't find a provider, then you have probably not installed virtualbox, or not rebooted since installing it.
- If you get some odd crash or error message about hypervisors: you cannot run vagrant when another program is already using your processor's virtualisation subsystem, and the page gives instructions how to turn off the other one.
  To exit the virtual machine, type `exit` which will get you back to the shell on the host machine. On the host, `vagrant halt` cleanly shuts down the virtual machine.

The file system

Have a look with the command `ls /`:

`/bin` stands for binaries, that is programs that you can run. Have a look with `ls /bin`: there will be a lot of commands in here, including ls itself. Indeed you can find out where a program is with `which`, so `which ls` will show you `/usr/bin/ls` for example.

`/usr` is a historical accident and a bit of a mess. in the earliest days,

- `/bin` was only for binaries needed to start the system - or at least the most important binaries that needed to live on the faster of several disk drives, like your shell.
- `/usr/bin` was where most binaries lived which were available globally, for example across all machines in an organisation.
- `/usr/local/bin` was for binaries installed by a local administrator, for example for a department within an organisation.

In any case, `/usr` and its subfolders are for normally read-only data, such as programs and configuration files but not temporary data or log files. It contains subfolders like `/usr/bin` or `/usr/lib` that duplicate folders in the root directory. Debian's way of cleaning this mess up is to make its `/bin` just a link to `/usr/bin` and putting everything in there, but in some distributions there are real differences between the folders.

If you have colours turned on (which is the default) you will see some files are green, but others are blue - this indicates the file type, green is an executable program, blue is a link to another file. Have a look with `ls -l /bin`: the very first character of each line indicates the file type, the main ones being `-` for normal file, `d` for directory and `l` for a so-called *soft link*. You can see where each link links to at the end of this listing. For example, `slogin` links to `ssh`. Other links point at files stored elsewhere in the filesystem – you'll see a lot of references to `/etc/alternatives/`.

`/etc` stores system-wide configuration files and typically only root (the administrator account) can change things in here. For example, system-wide SSH configuration lives in `/etc/ssh`.

`/lib` contains dynamic libraries - windows calls these `.dll` files, POSIX uses `.so`. For example, `/lib/x86_64-linux-gnu/libc.so.6` is the C library, which allows C programs to use functions like `printf`.

`/home` is the folder containing users' home directories, for example the default user vagrant gets `/home/vagrant`. The exception is root, the administrator account, who gets `/root`.

`/sbin` (system binaries) is another collection of programs, typically ones that only system administrators will use. For example, `fdisk` creates or deletes partitions on a disk and lots of programs with `fs` in their name deal with managing file systems. `/sbin/halt`, run as root (or another user that you have allowed to do this), shuts down the system; there is also `/sbin/reboot`.

`/tmp` is a temporary filesystem that may be stored in RAM instead of on disk (but swapped out if necessary), and that does not have to survive rebooting the machine.

`/var` holds files that vary over time, such as logs or caches.

`/dev`, `/sys` and `/proc` are virtual file systems. One of the UNIX design principles is that almost every interaction with the operating system should look to a program like reading and writing a file, or in short *everything is a file*. For example, `/dev` offers an interface to devices such as hard disks (`/dev/sda` is the first SCSI disk in the system, and `/dev/sda1` the first partition on that), memory (`/dev/mem`), and a number of pseudoterminals or ttys that we will talk about later. `/proc` provides access to running processes; `/sys` provides access to system functions. For example, on some laptop systems, writing to `/sys/class/backlight/acpi_video0/brightness` changes the screen brightness.

The `/shared` folder is not part of the FHS, but is this unit's convention for a shared folder with the host on Vagrant virtual machines. In previous years we called this folder `/vagrant`, but given the default username on the VM is *also* 'vagrant', this led to a lot of confusion, so we changed it.

Package managers

```
sudo apt install PACKAGE
```

- `sudo` (superuser do) allows you to run a command as root, also known as the administrator or superuser. It is good practice to use sudo for system adminstration instead of logging in as root directly, but if you ever really need a root shell then `sudo bash` gets you one - with `#` instead of `$` as prompt to warn you that you are working as root.
- `apt` is the Debian package manager.
- `install PACKAGE` adds a package, which means download and install it and all its dependencies.

You can also find out information about packages with `apt info PACKAGE` .

You can remove them from the system with `sudo apt remove PACKAGE`.

The repositories that you are using are recorded in `/etc/apt/sources.list`, have a look at this file with `cat` to see where they are, then look up the sites in your browser. There are folders for different Debian versions and package index files for different architectures.

Two commands a system adminstrator should run regularly for security reasons:

- `sudo apt update` fetches the new package list from the repository. This way, apt can tell you if any packages have been updated to new versions since you last checked.
- `sudo apt upgrade` upgrades every package that you already have installed to the latest version in your local package list (downloaded when you do an `apt update`).