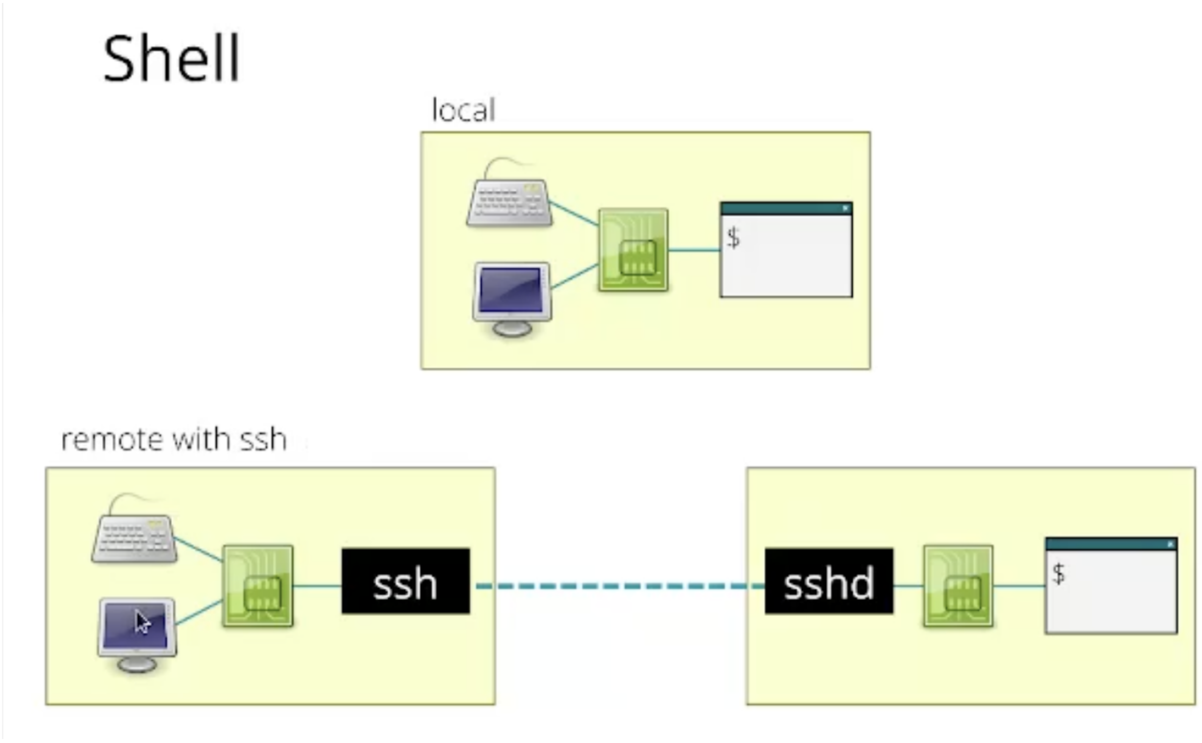
Secure Shell (SSH) (1)

- a **remote shell** is shell that runs on a machine you are not physically present on
- being able to run commands on machiens remotely poses significant **security risks**
- therefore most remote shells have been designed to address these security concerns

Commonly as programmers you will be asked to ssh remotely into a machine and interact with it using the shell



- top image = screen and keyabord interacting with your machine that is running the shell program
- bottom image: same process except your machine is running the **ssh client**
    - your commands are inputted into the client
    - they are sent over an encrypted channel to a **daemon program** at the other end
        - the daemon program handles interaction with the remote machine via a shell program it runs there
            - when output in that shell is displayed, the sshd program will send it back over the encrypted channel to **your** ssh client and then that result is displayed to you on your machine

Important details to note:

1. **in order to ssh into a remote machine, that machine must be running the sshd (daemon)**

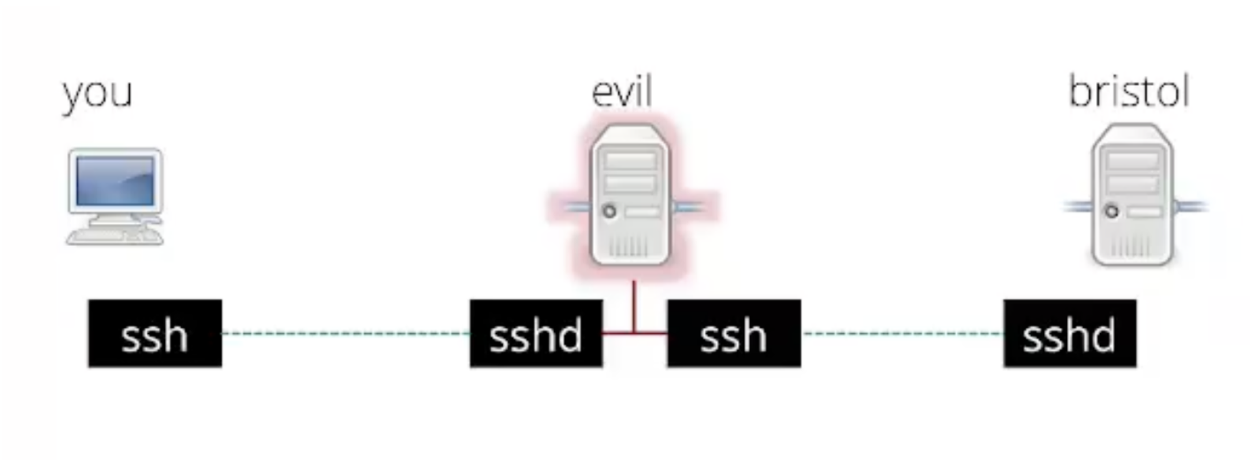
How do we use ssh?

```
ssh [USERNAME@][HOSTNAME]

// example for UoB ssh:

ssh cp20323@seis.bristol.ac.uk
```

- you will then be presented with message chekcing whether you want to still connect to this machine
    - this is a security feature of the secure shell (ssh) to protect you from malicious attacks
        - it is bascially asking you if you trust the server

```
Are you sure [y/n]?
```

Man in the Middle Attack



- attacker inserts itself into the middle of the sshd server you were attempting to reach
    - could even be the person that controls the local network
        - e.g. when connecting at uni you are trusting the uni's network and routers to send and recieve correctly

    Instead these machines could be instructed to direct your request to their own evil version of sshd

        - this machine can then accept your credentials effectively **pretending to be your machine**
            - neither your machine or bristol will be talking to who they think they are talking to

This middle man can then spy on commands you wished to send to Bristol machines or it could take control entirely and do other stuff with the access we have just given it.

Cryptography

- SSH servers (the daemon sshd) use key pairs to dentify the machine that is trying to connect to it
- for the first login however you have to trust the machine, hence the warning message asking if youre sure you want to trust it

Keys

What are SSH keys?

- a pair of **cryptographic** keys used to authenticate to an SSH server as an alternative to a *password-based login*
- **Public key** - safe to share with others and can be freely distributed without compromising security.
    - used to encrypt data and can only be decrypted by the **matching private key**
- **Private key** - must be kept secure and **never** shared
    - used to decrypt data that was encrypted with the public key and to sign data, allowing for the recipiant to verify its authenticity

Benefits of keys:

- we can create our own keys as well with **2 main benefits:**
- **convenient:** allows us to not have to type our password in when ssh'ing into a remote machine
    - e.g. cp202323 password

- **secure**: the keys never leave your machine, even if you connect to an evil ssh server

How to generate and then use a key to login to an SSH server?

1. **Generation:** an SSH key pair is generated by the user, the public key can be placed on any server you wish to access while the private key remains on your **local machine**
2. **Disributing the keys:** the public key is copied to the SSH server and is placed in a special file (`~/.ssh/authorized_keys`) in the users home directory. This tells the server that the corresponding private key (the one on the local machine) should be considered as an authorised key for logging in
3. **Authentication:** when a client tries to SSH into the server, the server will check if the public key it is presented with against a list of authorised keys, if a match is found, the server sends a **challenge** to the client that can only be answered with the **private key**
    1. if the client is successful, it proves it is in possesion of the private key matching the presented public key and is allowed to login to the server without a password

---

Example Questions

Alice is trying to set up key-based login on a service she currently accesses via using ssh
and entering her password. Here are listings of her .ssh directory on her local machine:

```
-rw------- 1 alice alice 389 Jan 19 10:56 authorized_keys
-rw-r--r-- 1 alice alice 395 Feb 21 13:03 id_rsa.pub
-rw-r--r-- 1 alice alice 225 Feb 21 13:01 known_hosts
```

And on the server:

```
-r-------- 1 alice alice 395 Feb 21 13:30 authorized_keys
-rw-r--r-- 1 alice alice 395 Feb 21 13:21 id_rsa.pub
-rw-r--r-- 1 alice alice 225 Jan 19 11:02 known_hosts
```

There's a problem evident with her current setup. Identify which file's presence, absence or
visible details indicates the problem.
A. The problem lies with 'authorized_keys' on the local machine.
B. The problem lies with 'known_hosts' on the server.

C. The problem lies with 'id_rsa' on the local machine.
D. The problem lies with 'id_rsa.pub' on the server.

ANSWER: C

Solution: The absence of a private key on the local machine is a clear problem for key-based
login. The permissions on the files are all correct. Moreover, 'authorized_keys' serves no
purpose locally, and 'known_hosts' and 'id_rsa.pub' don't matter on the server. We went
through this key-based setup process in the SSH labs at the start of Part 1.

- id_rsa would be where the privvate key is stored, which is essential for logging into ssh