Accessing University Servers and using GPUs

SSH or Secure Shell is a network communication protocol that enables two computers to communicate, particularly open a secure tunnel to another computer on which you can execute commands. SSH is the industry standard for accessing and using remove servers, deploying applications and many other uses.

SSH uses either **password protection** or **public/private keypair cryptography** to secure your connection. Using a keypair provides a more secure connection and will save you the hassle of having to write your password every time. A key pair contains a public key which other computers use to authenticate and decrypt your communication, and a private key which you use to encrypt and sign messages. You can either do this manually in the terminal, or using a client application.

Getting Started

UNIX (Bash/Zsh/Fish) and Windows (PowerShell) shells all use the same API for accessing servers. I have written the below for UNIX systems but the same should be applicable for Windows (untested, sorry, your fault for using Windows 😉). Windows documentation is here:

https://learn.microsoft.com/en-us/windows-server/administration/openssh/openssh_install_firstuse?tabs=gui
If you want to login from Windows, you can try MobaXterm.

Generating a key pair

The simplest way to generate a key pair is to run `ssh-keygen` without arguments. In this case, it will prompt for the file in which to store keys. Here's an example:

```
$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key ($HOME/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in $HOME/.ssh/id_rsa
Your public key has been saved in $HOME/.ssh/id_rsa.pub
The key fingerprint is:
```

```
SHA256:PRUjm3X/cVsiEHdpTN2wWyy5Z5SRyAunQE59nHOjU7Y
kag25@m900430.inf.susx.ac.uk
The key's randomart image is:
+---[RSA 3072]----+
|     .o=.BoB=o|
|     o. X &+@=|
|           .+ B.%oX|
|           . o = E*|
|     S o   +.+|
|           .   o |
|                 |
|                 |
|                 |
+----[SHA256]-----+
```

You should now have a directory in `$HOME/.ssh` containing your **public key** `id_rsa.pub` and your **private key** `id_rsa`.

You **should never ever share your private key**. The public key is safe to share.

How to connect to the servers from inside the University network

```
ssh <YOUR_SUSSEX_USERNAME>@<SERVER_NAME>.inf.sussex.ac.uk
```

How to connect to the servers from outside the University network

Connecting is a two-stage process due to a main server acting as a firewall between the computing cluster and the rest of the internet. First stage is to connect to the firewall server:

```
ssh <YOUR_SUSSEX_USERNAME>@unix.sussex.ac.uk
```

You will be prompted to enter your password. Once you're connected, you will see this:

```
            Welcome to the University of Sussex

                        Unix Service

            For IT Support please contact IT Services
```

```
                (A): Shawcross Building
                (E): itservicedesk@sussex.ac.uk
           (W): http://www.sussex.ac.uk/its/help
-bash-4.2$
```

Once on the server you will want to append your `id_rsa.pub` key to the `~/.ssh/authorized_keys` file. If it doesnt exist, you'll need to create it (nano is the default text editor). This will ensure the next time you log in, you use the key pair instead of your password, so you won't have to anymore.

You can now attempt to access the server you've been granted permission to e.g. `vili` (or any other server you have been granted access to).

```
-bash-4.2$ ssh <SERVER_NAME>.inf.susx.ac.uk
```

You will be prompted to enter your password again. If successful you should now have access. If you receive a permissions error, either you entered your password wrong or you do not currently have access and should speak to the relevant person.

Now you are on the server, you should follow the same procedure as before, append your `id_rsa.pub` file to the `~/.ssh/authorized_keys` file. Checkout `man ssh` for more information on using SSH.

One stage process using a config file

To make life much easier, you can configure this as a single stage shorthand process. Create a config file `.ssh/config` on your home computer. Change your username to match.

```
# ~/.ssh/config

Host unix
  HostName unix.sussex.ac.uk
  IdentityFile ~/.ssh/id_rsa
  IdentitiesOnly yes
  User <INSERT_USERNAME_HERE>
Host vili
  HostName vili
  User <INSERT_USERNAME_HERE>
  ProxyCommand ssh -YXCvq unix nc vili.inf.susx.ac.uk 22
Host *
  AddKeysToAgent yes
```

You should now be able to do `ssh vili`. If you have correctly put your public key onto each server, this will go straight through and you will now be on `vili`.

Copying files

The easiest way to get files directly onto the server is using `scp`. You can do this for large packages if you have a reliable connection.

```
scp <SOURCE_DIR>
<TARGET_SERVER_NAME>@<TARGET_SERVER_ADDRESS>:<TARGET_SERVER_DIRECTORY>
```

For example, I can copy a directory onto using the following command:

```
# -r for recursive, -v for verbose
scp -rv /path/to/src vili:/path/to/target
```

For getting code onto , we recommend using `git` so you can easily push and pull from your home computer to the git server, then pull it down to `vili`
`vili`
`vili`.

Running Jupyter Notebook on the server and accessing through the browser locally

If you are running a jupyter notebook on the remote server, served on a specific port `$PORT` e.g. 8888, you can access the interface through your local browser via an SSH tunnel.

```
ssh -vNL <PORT>:localhost:<PORT> vili
```

Then navigate to `http://localhost:<PORT>` in the browser.

Checking GPU status

To check which GPU is currently in command (that means which is an active VGA controller) type in

```
nvidia-smi
```

It will output something like this:

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 530.30.02              Driver Version: 530.30.02    CUDA Version: 12.1     |
|-----------------------------------------+----------------------+----------------------+
| GPU  Name                  Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf            Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                                         |                      |               MIG M. |
|=========================================+======================+======================|
|   0  NVIDIA RTX A6000             Off| 00000000:01:00.0 Off |                  Off |
| 30%   34C P8               19W / 300W|      0MiB / 49140MiB |      0%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+
|   1  NVIDIA RTX A6000             Off| 00000000:23:00.0 Off |                  Off |
| 30%   36C P0               78W / 300W|      0MiB / 49140MiB |      0%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+
|   2  NVIDIA RTX A6000             Off| 00000000:41:00.0 Off |                  Off |
| 30%   34C P0               76W / 300W|      0MiB / 49140MiB |      0%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+
|   3  NVIDIA RTX A6000             Off| 00000000:61:00.0 Off |                  Off |
| 30%   31C P0               69W / 300W|      0MiB / 49140MiB |      0%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+
```

```
|   4  NVIDIA RTX A6000            Off| 00000000:81:00.0 Off |
      Off |
| 30%   35C P0               77W / 300W|       0MiB / 49140MiB |
0%    Default |
|                                       |                      |
      N/A |
+-----------------------------------------+----------------------+------
----------------+
|   5  NVIDIA RTX A6000            Off| 00000000:A1:00.0 Off |
      Off |
| 30%   32C P0               70W / 300W|       0MiB / 49140MiB |
0%    Default |
|                                       |                      |
      N/A |
+-----------------------------------------+----------------------+------
----------------+
|   6  NVIDIA RTX A6000            Off| 00000000:C1:00.0 Off |
      Off |
| 30%   35C P0               72W / 300W|       0MiB / 49140MiB |
0%    Default |
|                                       |                      |
      N/A |
+-----------------------------------------+----------------------+------
----------------+
|   7  NVIDIA RTX A6000            Off| 00000000:E1:00.0 Off |
      Off |
| 30%   35C P0               75W / 300W|       0MiB / 49140MiB |
0%    Default |
|                                       |                      |
      N/A |
+-----------------------------------------+----------------------+------
----------------+

+-----------------------------------------------------------------------------
----------------+
| Processes:
      |
|  GPU   GI   CI        PID   Type   Process name
      GPU Memory |
|     ID   ID
Usage       |
|=============================================================================
================|
|  No running processes found
      |ggyG
+-----------------------------------------------------------------------------
```

```
----------------+
```

Live GPUs in use will show up at the bottom.

Keeping Sessions Alive

If you start processes in a standard terminal, when you disconnect from SSH, your process will exit too. We don't want this! So we create a virtual session using `tmux` to get around this, which you can attach and detach to when you log onto the server. TMUX will preserve your live sessions and ensure your code keeps executing.

TMUX should already be installed, however if its not, you can simply run:

```
sudo apt-get install tmux
```

Create a new TMUX session:

```
tmux new
```

This will open the terminal in a new screen. You can split this into multiple windows each of which is a separate process if you want (look into the online documentation).

You can now run any commands you need that take a long time to execute.

To **detach securely** from the session while leaving it running, hit <CTRL-A D>. Now you can exit the server and your session will be preserved.

When returning to the server and wanting to reattach to an existing session, simply run:

```
tmux attach
```

ANACONDA SET-UP

1. Use wget to to get your preferred anaconda version. For instance:

wget https://repo.anaconda.com/archive/Anaconda3-2020.07-Linux-x86_64.sh

2. Then, go to where you downloaded the file, and type

bash Anaconda3-2020.07-Linux-x86_64.sh

3. Follow the T&C steps. Then, type
export PATH="${PATH}:~/anaconda3/bin:$PATH" >> ~/.bashrc

***Very important to include the ${PATH}. If not included, you'll change the whole $PATH variable and that's a pain to solve.**

4. Now if you try to use conda, it will not work straight away. You'll need to use interactive bash. You can do this by typing **bash -i**

You should now be able to use conda as needed! (you'll need to type bash -i each time you enter vili for using conda)