Use of Pawar lab Computers

July 16, 2018

Contents

1 Introduction

This document is meant to provide guidelines for remotely accessing and using multi-core computers available in the Pawar Lab at Silwood Park. Please feel free to make suggestions that can improve it! If you are really interested, we can provide you with write access to the LaTeX code of this document.

Most computers in our lab run the Ubuntu 64 bit avatar of Linux. Some machines may be running Mac OS X. You will not find Windows anywhere (other than the ones that let light in) except perhaps running sheepishly inside a virtual OS.

2 Off-campus access

If you are accessing any of the lab computers from off-campus, you will first need to `vpn` into the IC secure domain before you can connect to a machine. To see how to set up vpn for IC, Check out `www3.imperial.ac.uk/ict/services/networks/networkconnections/vpnconnection` (a very long url, *a la* ICT (`http://www3.imperial.ac.uk/ICT`) ☺)!

3 Server names and ip addresses

We currently have two servers:

1. William, a 32-core machine with 64Gb RAM, with IP address: `129.31.3.6`
2. Harvey, a 12-core machine with about 50Gb RAM, with IP address: `129.31.3.38`

You may have access to one or both and will have been told which by the names "William" and "Harvey" for the machines. You will have been assigned and given a username, and a temporary password that you can reset.

4 Connecting from a Linux machine

For ssh, read about the command ssh. All you do is tell it what remote machine to connect to, as what user, and it does it. Then the terminal window becomes like a terminal window for the remote machine!

Ubuntu linux also has FTP software built into its Nautilus file management system and SSH from the command line. For ftp, see the dropdown menus on the Nautilus file management system – one option is for connecting to a remote machine. It's straightforward. Then the Nautilus window becomes like a Nautilus window on the remote machine! You can copy files from another Nautilus window that is pointed at a local director into the Nautilus window that is pointed at the remote machine and over the files go.

4.1 Setting up `ssh` between two computers

```
$ ssh-keygen -t rsa
$ scp .ssh/id_dsa.pub  yourname@129.31.3.6:.ssh
$ ssh yourname@129.31.3.6
```

5 Connecting from a Mac

Connection from a Mac is exactly the same as from Linux using ssh. All you do is tell it what remote machine to connect to, as what user, and it does it. Then the terminal window becomes like a terminal window for the remote machine!

6 How to connect from a Windows machine

Download and install SmartFTP and Putty. SmartFTP is for moving files back and forth between your machine and a remote machine. Putty is for logging into a remote machine. There are very many alternatives to these. What you need is FTP software and SSH software. These are choices that seem to work if you are running Windows, and probably work well if you are using another operating system, but depending on your system and preferences you may opt for other software.

To use Putty to connect:

- Open putty
- Type in the IP address
- Chose connection type SSH
- Click open

You may want to save the name of the machine and the IP address and connection type to make it easier to connect in future.

To use SmartFTP to connect:

- Open SmartFTP
- Under File, click New Connection
- Protocol should be SFTP over SSH
- Type in the rest of the information and hit OK

Putty gives you the command line interface in linux – see below for a basic introduction. SmartFTP gives you a fairly transparent GUI for moving files. Depending on your level of computing sophistication these tools may be obvious to you or you may need to do a bit of reading and playing around to get used to them. There are plenty of resources online.

7 Once you are in

Here are a few guidelines once you have successfully managed to log in (e.g., using Putty or whatever in Windows). Please also see the section on acceptable use policy below. The following guidelines are assuming you are in the remote Ubuntu terminal, which will look something like this:

```
pawar@william:~$
```

Here `pawar` is the user name, `william` is the machine name, and the `$` is the linux terminal prompt after which you enter commands. As you will notice, the command prompt is not like Mac or Windows. You will type commands, which the linux shell executes, and then you get another command line. Seems painful but is actually much more powerful than GUI interfaces.

7.1 Reset your temporary password

Once you are in terminal, please change your password from whatever you were provided to something more secure. This is done with the UNIX `passwd` command. You can figure out the details using google.

7.2 The linux directory (folder) and file structure

Linux uses a folder/directory tree just like Max and Windows. To see where you are in the tree type `pwd` and then return. This stands for "print working directory". You should be in your `home` directory, which is `/home/xxx`, where xxx is your login name. Thereafter, the following commands are particularly handy:

`ls`	shows the directories and files in your current directory
`cd MyDir`	changes into the subdirectory `MyDir` — can include a path name
`cd ..`	goes up one directory in the hierarchy
`cat MyFile`	if `MyFile` is a text file, displays it in one shot to the screen (just for looking, not for editing). Text files include `.py`, `.R`, `.csv`, `.tex`, etc.
`cp xxx yyy`	copies file xxx to the new name yyy
`mv xxx yyy`	similar but moves
`mkdir MyDir`	make a directory in the current directory, call it `MyDir`
`rmdir MyDir`	remove a directory

Please use `man command` (or just google it) to get help on a command in the terminal.

8 How to remotely run scripts on a lab machine

You can run R, Python, or whatever is installed on the machine from the linux terminal. You can then copy commands in, but this is a bad way to do it for several reasons. A better way to run your scripts is:

1. First, write your code on your own machine with these features:
    - It loads in all data that are needed automatically.
    - It does all computations and saves all results automatically. Results can be .RData files, or .pdf or .jpg files or a number of other formats.
    - There are constants at the top of the code file that allow for variable lengths of run. E.g., in R, suppose you want to do a simulation a million times, and you know that will take a while and that is why you want to use one of the Pawar lab machines. Write code with a constant at the top called, say, `num.sims`. Set it to 10 initially. Make sure all aspects of your code work on your own machine with `num.sims <- 10`. Then copy the code using one fo the above methods to William or Harvey, then make sure it works there with `num.sims<-10`. Then change `num.sims<-1000000` and run it on William or Harvey.
2. Now, you need to run the script from the command line in linux. For this (using R as an example),
    - Copy the script to an appropriately chosen directory on the machine.

      Important note for Harvey only: this directory should be under $WORK and not in your home directory. I.e., as soon as you log in, type `cd $WORK` to get there. To figure out the path of the $WORK directory, type `pwd` after the previous command.
    - Get the linux working directory to that directory. Suppose the script is called `MyScript.R`
    - Then, at the bash prompt, `R CMD BATCH MyScript.R` runs the script. However, your command line will hang until the script is done (it's waiting for the script to finish). Also, if your connection to the machine is broken, the script stops running.
    - So, instead use `R CMD BATCH script.R &` (The `&` makes it run "in the background" so you get your command line back before the script is done. However, if you logout the job still stops.
    - Using `nohup R CMD BATCH script.R &` will prevent your job from quitting if you log out.
3. Monitor your job:

- Type `top -d 5` at the bash prompt
- This displays allinformation about what jobs are currently running, who owns them, how much resources they are taking up, etc. (use `man top` for more info)
- If you have run a job that takes a while, you should see it taking up 100% cpu time (that's 100% on one of the multiple available cores)
- If you ran a job and realized it is wrong and want to kill it, from within top, type `k` and then enter the job number and it will kill it
- You can run several jobs at once and should see them all listed

9 Some rules

1. Before running a job, run `top` to see what other jobs are running. You can count how many jobs are taking 100% of cpu, or close to it. Those are jobs using a whole core. There are 32 cores on William and 12 on Harvey. Please make sure not to start any jobs that will make it so that more than 31 cores are busy on William or 11 on Harvey. This is so one core is always free for logging in and other admin.
2. It's up to you to police yourself here. If more jobs than cores are run, then the machine slows down for everyone because it is busy switching between jobs. Feel free to email or speak to people about when their jobs will be done.
3. If your jobs are memory intensive, please monitor this as well. The `top` command gives memory stats as well. Please run jobs with reduced load requirements (see the num.sims example above) first to make estimates of memory usage. Don't use all the memory – that makes the machine (glacially) slow, makes it wear out the disk, and is a big pain for anyone else using the machine.

10 Parallelization

The above sections assume that you will you run one script on one core at a time (though you can manually run it on multiple cores through multiple execution commands). To run one script that uses multiple cores, you need *parallelization*.

This document is not really for learning par allelization of your computer code. However, parallelization is how you get the most power out of multi-core machines. Following are some suggestions.

10.1 Parallelization using `R`

There are some nice ways to parallelize code in R on linux. Look into `parallel` package and the command `mclapply` in particular.

10.2 Parallelization using `python`

If you want to parallelize Python code to run on multiple cores, there are a few different options:

- Use the `threading` module: `https://docs.python.org/2/library/threading.html`
- Use the `multiprocessing` module `https://docs.python.org/3/library/multiprocessing.html` to run your code in multiple processes at once. This article may be particularly useful: `http://chriskiehl.com/article/parallelism-in-one-line/`
- Use the `subprocess` module to run multiple python interpreters and communicate between them.

- Google “parallelize python”, and other solutions will pop up.

10.3 Serious parallelization using the Imperial College cluster

Serious parallelization requires a different setup. If the lab machines are too puny for your task, you’ll have to use the Imperial High Performance Computing cluster, wich has 10000+ cores.

11 Getting help

If after giving this document a thorough read and try you find yourself stuck, please ask a lab member or someone else you know has used these machines (For example the nearest MSc/MRes CMEE student!).