Queen Mary
University of London

ECS404U: COMPUTER SYSTEMS & NETWORKS

2020/21 – Semester 1

Prof. Edmund Robinson, Dr. Arman Khouzani

***Lab 1**: Warmp-up: Some useful Skills*

September 21/23, 2020

Instructions:

- Submit a **single PDF** file electronically to QM+ (only) via the link for lab submission of week 1.
- You should not submit your work by emailing it to a member of staff (lecturers or demonstrators).
- The deadline for submission of each lab is **10:00 AM of the Thursday of the following week**. The lab assignments are designed to be finished well within the 2-hours of each lab session. So you should be able to finish and submit your work by the end of the lab session.
- Each lab is worth 1.5%. There are a total of 10 assessed labs, for a sum contribution of 15%.
- The assessment outcome for each lab is "binary" (`Pass/Fail`): either you will get the 1.5% for that lab assignment or zero. A submission that has up to a few small mistakes will still get 1.5%, but one with many mistakes or containing few major mistakes (or no submissions!) will result in a 0%.
- Note that even if you get a `Pass` for a lab assignment, you should still check your feedback (explanation of your mistakes). Both your grade and your feedback will be accessible through *Grades* or *Grades plus* link on the QM+ page of the module (find it on the panel on the right side of the module's page).
- Due to the `Pass/Fail` mode of assessment for each of the lab assignments, there is no opportunity of late submission, except for cases of approved *Extenuating Circumstances* (EC's).
- The solutions of each lab exercise will be posted upon its deadline.
- The single PDF can include scans or just photos of your handwritten material (taken e.g. by your phone) as long as they are legible. If you are taking photos with your phone, make sure your entire solutions are in the frames, the pictures are well-lit, and in focus. Although hand-written solutions are accepted, a typewritten version is encouraged (even better if you use LaTeX!).
- There is a zero-tolerance plagiarism policy at QMUL, and strictly enforced in this module. So make sure your submission is your own work, or have clear citing of your sources (if in doubt, ask!).
- You are nevertheless strongly encouraged to actively engage in asking questions from demonstrators/instructors during the lab sessions, posting questions to the discussion forum on QM+ page of the module, and replying to other students' questions on the forum. You can also email the instructors of the module with your questions or simply request a (virtual) meeting. Also, don't forget about the opportunity to "book" for our weekly on-campus live tutorial sessions on Thursdays at 3-4 PM (Location: Laws:1.00). Note that there is a capacity of 15 students and prior booking is necessary (due to contact tracing requirements.)
- Your answers should be easily mappable to the questions. So please answer in sequence, and label clearly with a question/part number.

Instructions: Week's 1 lab takes place for some of you before the first lecture. Therefore, it will have a different nature than the rest of the labs: we will concentrate on gaining familiarity with some general CS skills. It will also be mostly presented as a tutorial. The lab is still assessed (what you need to upload as your lab assignment is explained at the end). However, the material covered in this lab is **NOT** directly assessed beyond this lab (e.g. in the midterm quiz or the final exam).
The lab is composed of three main parts:

1. Remote log-in to ITL and remote file transfer;
2. Getting to use Linux;
3. Basics of version control with `git`.

Space is provided in the lab manual itself for you to write down your answers. But if you don't have access to a printer, or you are choosing to be green and not print them (thank you!), just type in your answers or hand-write them, but then make sure your answers are easily mappable to the questions; so please answer the questions in sequence and clearly label each of them. Further instructions are provided on the cover page.
Please do ask for help throughout the lab whenever you need, using the public chat, or private chat with any of the moderators. Again, don't forget about the student forum on QM+ either!

1 Remote ssh Log-in to ITL

Learning Objectives This section explains how to get remote access to our machines on campus using `ssh`.

remote log-in with `ssh`

`ssh` stands for *secure shell*, and is a remote log-in programme. From its manual page:

> `ssh` (SSH client) is a program for logging into a remote machine and for executing commands on a remote machine. It is intended to provide secure encrypted communications between two untrusted hosts over an insecure network.

Consider the scenario that you want to connect to an ITL machine from somewhere outside of ITL (on or off-campus).

1. Open a `terminal`, or run `PowerShell` in Windows (as you would run any other desktop app).
2. The basic syntax for using `ssh` is: `ssh user@hostname`.
   The hostname in our case should be `login.student.eecs.qmul.ac.uk`, which is the alias for our login server (whose name is actually `grover`, yes, named after the muppet character!). So you can type in:

   ```
   ssh username@login.student.eecs.qmul.ac.uk
   ```

   or equivalently:

   ```
   ssh username@grover.student.eecs.qmul.ac.uk
   ```

   Remember to replace `username` with your own username, of course [and this is the username you have with the EECS school, not your college username or student ID.]
   The first time, you may be prompted whether you trust the public key of the server (so this is added to the pool of trusted public keys). This is to prevent a man-in-the-middle attack (a rouge server pretending to be grover!), but more on this in your third year.
3. Once prompted for password, enter it. *Note: this is the password for your EECS username.*
   Even though no characters shows up while you are typing your password, they are there – so type in your password and hit `Enter` (this is a security mechanism to hide the length of your password from potential onlookers). You should then be inside grover.

Nice side point: you can issue the command `w` to see the total uptime of the server, its average load, a list of users who are currently logged in, where they have come from, how long they have been logged in, and what are they running.

4. From `grover`, you can then do another "hop" of ssh log-in to any machine in the ITL lab (that is up!). Simply type in:

   ```
   ssh itlnnn
   ```

   where `nnn` is the number of the ITL machine. For instance, to hop into `itl110`, you just issue `ssh itl110`. Note that this is short for `ssh yourusername@itl110.student.eecs.qmul.ac.uk`, but now the default values for the username and domain name are correct, and don't have to be explicitly specified.
5. To check which "node" you are currently logged in, you can issue:

   ```
   uname -n
   ```

   What do you see?
   In order to logout from a machine, you can simply enter `logout` or `exit`. Try it and issue `uname -n` once again. What do you see now? Make sure you have returned to your own local machine!

advanced, optional If you are annoyed at entering your password each time, there is a way to automatically authenticate yourself based on public-private keys. If interested search online for "ssh login without password".

2 Introduction to Using Linux

Learning Objectives The aim of this part is to familiarise you with some basic tasks in Linux, which, in this module or others, you may need to use. In particular, you will learn how to navigate between directories, and creating/copying/moving/renaming/deleting files and directories.

The `terminal`

Once you have remotely logged in to `grover` (or from there to any of the ITL machine), all you have is a Linux "terminal". The terminal provides an interface to the Linux command line interpreter (which is typically `bash`). Simply put, it means now we can enter commands and the OS executes them for us. Details of this will become clearer in the "Operating Systems" module during your second year.
First, some general points:

- Each of you have a storage quota of *5 GB* by default. This is a network filesystem (your home directory is "mounted" from a central server). This means that you can see your files no matter which ITL machine you log in. So if you create or save a file on one ITL machine, it will be there if you log in to another machine too.
- Linux is an open-source operating system and comes in many different "distributions". Find out which distribution your machine has by issuing `lsb_release -d`. Note the underline and then the single "dash" (or "hyphen"). In Linux, commands can typically be used with different "options". These options usually have a short and a long version. The short version is specified by a single dash, and the long version with a double-dash. For instance, the following two commands are equivalent:

  ```
  lsb_release -d
  lsb_release --description
  ```

- In Linux, you can access the "documentation" of an instruction by putting `man` before them (short for "manual"). For instance, to get to the documentation of `lsb_release`, issue: `man lsb_release`. There, you can see a brief description of what the command does, along with how it should be used (its "synopsis"). You can quit the manual and return to the command line by pressing the "`q`" button.
- Besides the "`man`" pages, Linux command also typically (but not always) provide a (shorter) help by issuing them with the "`-h`", or equivalently `--help` after them. For instance, to get a brief help on `lsb_release` by issuing: `lsb_release -h`, or `lsb_release --help`.
- Note that, unlike Windows, Linux is "case sensitive" (differentiates between lower and upper-case letters). So, for instance, `Documents/` is not the same as `documents/`.

Navigating through the filesystem

Using the "`man`" pages (under "description"), or using "`--help`", find out what each of the following Linux commands do. The first one is done for you as an example:

▷ `cd`
change the current directory

▷ `pwd`

▷ `ls`

▷ `mkdir`

▷ `cp`

▷ `mv`

▷ `rm`

A few more points before we move on to the task of this section:

- In Linux, different parts of a path-name are separated by a forward-slash, e.g.: `/homes/khouzani/Documents`, so the root directory is just a single forward-slash: `/`
- The short-name for your "home" directory is a "tilde" sign: $\sim$, so for instance, the command `cd` $\sim$ (note the space) will take me to `/homes/khouzani/`, or the command `cd` $\sim$`/Documents/` will take me to `/homes/khouzani/Documents/`
- There are two more special names when using file-paths in Linux: single-dot: `.`, which represents the directory you are currently in, and double-dot: `..`, which represents the "parent" directory (the immediate directory above the current directory). So for instance, if I am in `/homes/khouzani/Documents/` and issue `cd ..` (again, mind the gap!), my current directory will change to `/homes/khouzani/`
- In the terminal, you can use the `Tab` key for auto-completing.
- You can pass multiple options to a command. For instance, reading the manual, find out what "`ls -l`" and "`ls -t`" do. Then make a guess what "`ls -l -t`", or equivalently, "`ls -lt`" does.

Creating new directories and files

Use what you learned above to accomplish the following tasks:

1. Create a new directory in your `Documents` called `ECS404U`, and inside it, a directory called `week01`.
2. Inside that directory, create a simple text file called `readme.txt` and fill it with a greeting message. (Note that in Linux, files do not need to have an extension, so we could just call it just `readme`, for example). To create a simple text file in Linux, you have many options for your text editor: from the geeked-out `vim`, to the feature-rich `atom` and `sublime`. But a middle-ground is perhaps `nano` (my personal favourite), or `gedit` (not among my favourites!). What is important is that when remotely logged in using ssh, you will not have any graphical user interface (unless you use a `-X` option, but we don't!). So options that rely on a graphical user interface like `atom`, `sublime` and `gedit` go out of the picture, and terminal based options line `vi`, `vim`, and `nano` remain. If you choose to use `nano` like me, you can create a new file by simply typing `nano`, then writing the text, and once done, press `Ctrl+X` to save and exit.

remote file transfer with `sftp` (and optionally, `rsync`)

`sftp`, standing for *secure file transfer program*, is a program for transferring files over `ssh` (and hence, in an authenticated and encrypted way).
Adapted from its manual:

> `sftp` is an interactive file transfer program which performs all operations over an encrypted `ssh` transport. It may also use many features of `ssh`, such as public key authentication and compression. `sftp` connects and logs into the specified host, then enters an interactive command mode.

The syntax of `sftp` is very similar to `ssh`:

```
sftp user@host
```

After authentication, you will enter an interactive terminal. Once there, you can issue the command `help` to get a list of useful commands. The following are the most important ones:

`pwd`	:	print the current working directory on the remote machine
`lpwd`	:	print the current working directory on the local machine
`cd`	:	to change the remote directory
`lcd`	:	to change the local directory
`ls`	:	display remote directory listing
`lls`	:	display local directory listing
`get`	:	download file (from remote machine to your local machine)
`get -r`	:	download a directory (`-r` is for recursive, so recursively all the files in a directory are downloaded)
`put`	:	upload file (from your local machine to the remote machine)
`put -r`	:	upload a directory (recursively all the files in it)
`bye`	:	quit (same as `exit`) back to the local shell

Now, carry out the following task: (note: As before, if your operating system is Windows, you either should use `PowerShell` or something with a friendlier graphical user interface like `FileZilla`, available for free at https://filezilla-project.org/.):

1. Create a small text file on your own machine (call it e.g. `upload_test.txt`. Then upload it the `gover` server, under the directory that you created `~/Documents/ECS404U/week1/`. Hint: the easiest way is if you navigate to the intended target directory on the server (using `pwd, cd, ls`, etc), also navigate to the intended local directory (using `lpwd, lcd, lls`, etc), and then simply issue: `put name_of_the_file_to_uplaod`.
2. After you've done the above, exit the `sftp` program (type in `bye` or `exit`).

optional Another way to do secure file transfer is by using `rsync` (from its manual: "a fast, versatile, remote (and local) file-copying tool"). A difference is that it is no longer interactive. We leave it to you to read up on it if interested.

3 Basics of `git`

`Git` is "by far, the most widely used modern version control system in the world today"[1]
But what does "version control" mean?
It means it can be used to help you track versions of a piece of work and get it back to a previous state if necessary (e.g. if you mess up, or someone else in a group project messes up!) Essentially, it allows you to take "snapshots" of a project as your team develops a project, as historical checkpoints. It also allows you to "merge" changes made by different contributors (as long as the files are text-based, like code!), create different development branches, and many other useful tools that is beyond the intent of this exercise. We highly recommend that you master `Git` (way beyond this simple exercise), as it has become an indispensable skill in the CS workforce (and quite rightfully so!).
Git was originally developed for use in the Linux project circa 2005 by a team led by Linus Torvalds (check his wikipedia page!).
So the true power (and benefit) of a version control system is to help manage (coding) projects that are being worked on concurrently by many different contributors, with the risk that they will make different incompatible changes. That said, the current exercise covers only the basic "single-user" (standalone) scenario, i.e., assumes you are an individual developer of a project.

What's in it for you?

At a basic level version control lets you edit documents more freely. You can delete stuff instead of commenting it out, or marking it up for deletion. You can always get it back from an earlier version. 1It also helps protect against document corruption and editing errors. You can get the previous version back. This is a regular issue for students preparing coursework close to a deadline. But version control does not automatically protect you against disk problems or having your laptop stolen. You should always back up your work on a separate disk. Software houses use version control, very commonly Git. It is an expected skill. You may find it useful to support groupwork. Finally, you may find it useful in the unfortunate event that you are ever accused of plagiarism. A Git archive gives a history of how you developed a document, including timings.

3.1 Create your first `git` repository

Before you start, have a quick read of the main concepts from here: https://git-scm.com/book/en/v1/Getting-Started-Git-Basics. Now:

1. `Git` is already installed on the ITL machines. Invoke its manual: "`man git`" and have a quick read of its first page.
2. Set up Git by telling it about your identity (enter into command line, replacing with your own name and email, this is the identity that your "commits" are going to be registered under):
   ```
   $ git config --global user.name "John Doe"
   $ git config --global user.email johndoe@example.com
   ```
3. To make the outputs colourful:
   ```
   $ git config --global color.ui true
   ```
4. Create a new project. Inside your week01 directory, issue:
   ```
   $ git init
   ```
   That's it: you created your first git repository!
5. Let's check the state of our repository:
   ```
   $ git status
   ```

[1] https://www.atlassian.com/git/tutorials/what-is-git.

You should see that unfortunately, `readme.txt` is untracked. So our repository is actually empty, as of now.

6. Let's "add" `readme.txt` to the "staging area" and check the state of our git again (`git add .` actually adds all files in the current directory – recall that in Linux, a single dot represents the current directory):
   `$ git add .`
   `$ git status`
   What this means is that we are telling `git` that we want it to keep an eye on this file (track its changes). Before that, `git` was ignoring this file. Note, however, that we have not created the snapshot yet. That happens when we "commit" to the current stage.
7. Perform the commit and check Git's status. The `-m"Initial commit"` gives a short message describing the commit. Each commit must have a message describing what changes are being committed:

   ```
   $ git commit -m"Initial commit"
   $ git status
   ```

   Right now, we have successfully created this historical snapshot. We can revert to this version of the files no matter what future changes (or deletions) take place.
8. Check the commit history:

   ```
   $ git log
   ```

9. Change (edit) `readme.txt` (say using: `nano readme.txt`, make sure you save the changes and exit).
10. Now get the status of the repository:

    ```
    $ git status
    ```

    What does the message say?
11. So you should see that Git has realised that the file has been modified (because we "added" it), but the changes are not added to the stage (and definitely, not committed yet). Before we add these new changes and commit to them, we can check what changes have we made:

    ```
    git diff HEAD
    ```

    This should show the detail of the changes (the "difference") from the latest commit (`HEAD` points to the latest commit.)
12. Add and commit to these new changes and display the commit logs again (Perform steps 6–8 – of course with a different commit message). You should now see two commits.
13. Change (edit) `readme.txt` one more time, but instead of using `git add` and then `git commit` separately, let's combine them in a shortcut:
    `$ git commit -a -m"Third change to readme.txt"`
    `$ git log`
14. Let's see who is to "blame" for each of the changes in a specific file (this makes more sense when you are collaborating in a project!):

    ```
    $ git blame readme.txt
    ```

15. Let's say we decide we want the version of `readme.txt` from the previous commit. We are on the "master" branch so we achieve that by:
    `$ git checkout master~1 readme.txt` (that is a "tilde" $\sim$ after `master`)
    Check that this has worked. Be careful with this. It overwrites the current `readme.txt` (so you will have lost all the changes that you made to that file since that commit).

optional If you have time, follow this tutorial as well, which covers branching, and merging too:
https://git-scm.com/docs/everyday#_individual_developer_standalone_individual_developer_standalone

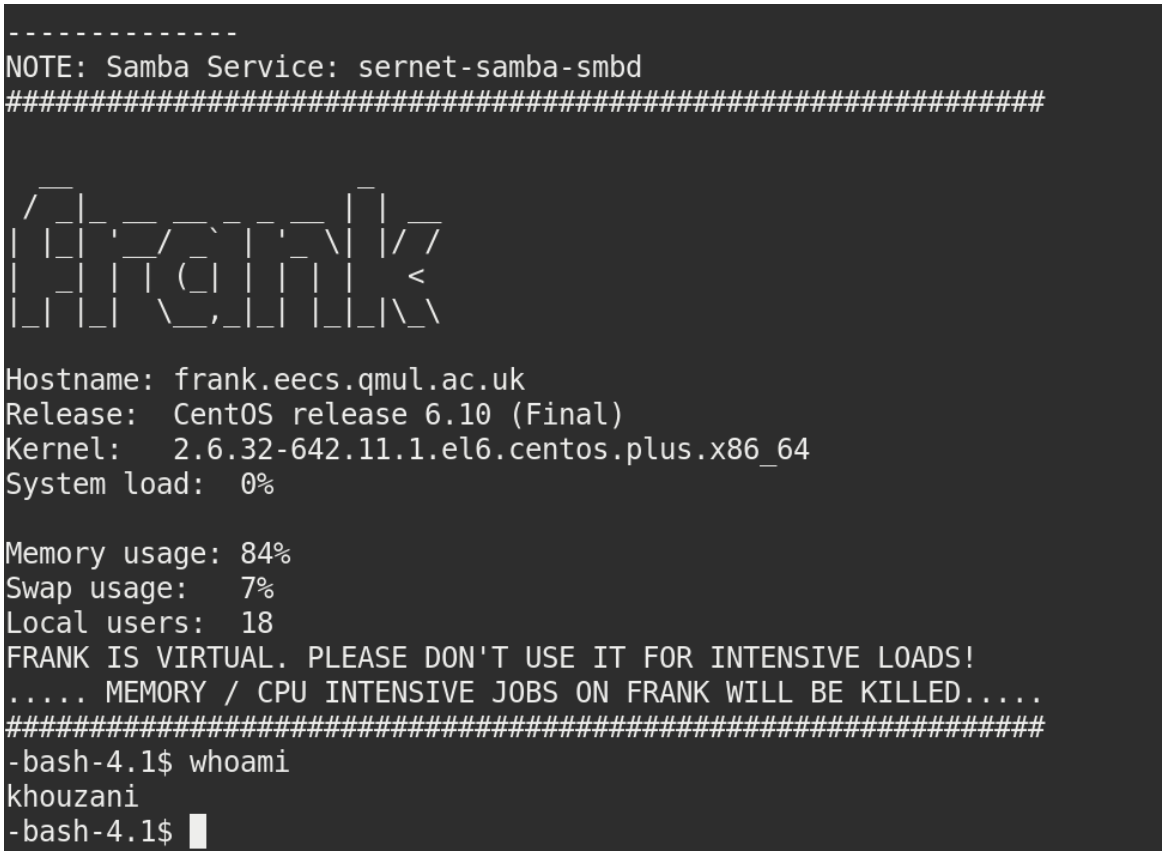
3.2 Other reading resources on `git`

- https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control
- Learn Git with Bitbucket Cloud.

---

4 What you need to upload as your week 1's assignment

1. successfully ssh log in to the `login.student.eecs.qmul.ac.uk` using your EECS username and password. Upon logging in, issue the command `uname -a`, and then the command `whoami` (no spaces between the words!). Then take a screenshot (using the print-screen button, or simply your phone). Attach the print-screen as your proof (that's all!). It should look like the following:



   Pro hint: when taking a print-screen, if you use the `Alt+PrtScr`, then the screenshot is only from the active window. If you hold the `Shift+PrtScr`, then you can choose by mouse the region of the screenshot!

End of questions