

Brief on *NIX at NJIT

Alex. Gerbessiotis
CS Department
NJIT

May 25, 2023

1. OSL (Open System Lab)

The term OSL at NJIT denotes Linux machines physically located on the 2nd floor of the GITC building in a laboratory that is known as Open Systems Laboratory (OSL in short).

The machines located in OSL have DOMAIN names oslX.njit.edu or oslXY.njit.edu, where X,Y are digits mapping to integers from 1 to around 31.

A given domain name maps to a unique IP address starting with 128.234.44.51 for osl1.njit.edu, but note that osl31.njit.edu might map to 128.234.44.47, which is non-intuitive.

In this document, for the sake of the discussion to follow I will be using example machines osl7.njit.edu and osl21.njit.edu. Therefore, X=7 in the former case and X=2 and Y=1 in the latter case.

If you plan to connect to any one of those machines by visiting NJIT's GITC building, skip Section 2 below. If you are inside NJIT, and you are using a wired connection skip Section 2 below. Some URLs (Uniform Resource Locator) of interest are as follows.

0. NJIT computer policies as applicable to the CS Department and other NJIT units are available, as of this writing, at <https://ist.njit.edu/student-computers>
1. The URL for downloading NJIT's copy of MobaXterm, a Windows secure shell client (ssh) is shown below. The URL provides also information about using ssh on Mac OSX and Linux. We do not discuss these options. One may also download a copy of MobaXterm of limited functionality directly from the manufacturer. In that case you may not even need to install MobaXterm; for more see manufacture/publisher's web site. <https://ist.njit.edu/accessing-afs>
2. The URL for 'accessing AFS' which means connecting to a linux machine at NJIT is shown below <https://ist.njit.edu/afs>
3. The NJIT VPN URL with links to downloadable VPN clients for Windows, MacOSX and Linux with instructions is shown below. <https://ist.njit.edu/vpn>
4. An NJIT URL with info on *nix commands is shown below. <https://ist.njit.edu/common-UNIX-commands>

2. VPN

The discussion below uses a client computer that is a WINDOWS 10 machine. This is in accordance with YWCC guidelines. (It also applies to Windows 11 machines.)

NJIT persons have three options in connecting to an OSL NJIT machine:

- (a) Visiting the OSL Lab on the 2nd floor of GITC,
- (b) Using within NJIT a laptop with a wired internet connection,
- (c) Using within NJIT a laptop with a wireless internet connection,
- (d) Using from OUTSIDE of NJIT a laptop.

The discussion to follow applies only to case (d) and in some rare cases to case (c) and both would be referred to as being 'outside of NJIT'. You may skip this section if your situation is under case (a) or case (b).

If you plan to connect to an OSL machine from **outside of NJIT** you must

- (1) Detect if a VPN client has been installed previously on your machine. We expect Computing students can figure out whether a VPN client is preinstalled or previously installed (by you) on their machine (Setting->Programs or Setting->Apps might provide some information). If a VPN client is not installed, first download such a VPN (Virtual Private Network) client though URL 3 of Section 1, and install it. Installation is done once and might require a reboot or a restart.
- (2) Activate VPN if a VPN client is preinstalled on your machine but is currently deactivated (this is a rare case, since by default it is activated at boot time). If you know how to deactivate it, you should know how to activate it or reactivate it.

Instructions below are for a Windows 10 machine but also applicable to a Windows 11 machine.

On my laptop the windows taskbar is at the bottom of the screen, with a Windows icon on the bottom left corner and the time and date information on the bottom right area/corner. In the bottom right area of the taskbar you might see the icon shown below. If not, find an up-arrow in the right area of the task bar, click on it and see if the icon is depicted on the

popped up window. For me, it appears as shown below in Figure 1. This means that the VPN client is **ACTIVATED but it is not in USE** ('not connected to the NJIT' network). If the VPN client were in USE, the icon would have appeared as in Figure 4 instead.

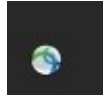


Figure 1.

1. You may click on this icon (shown in Figure 1) and the following pop-up window might appear (Figure 2).

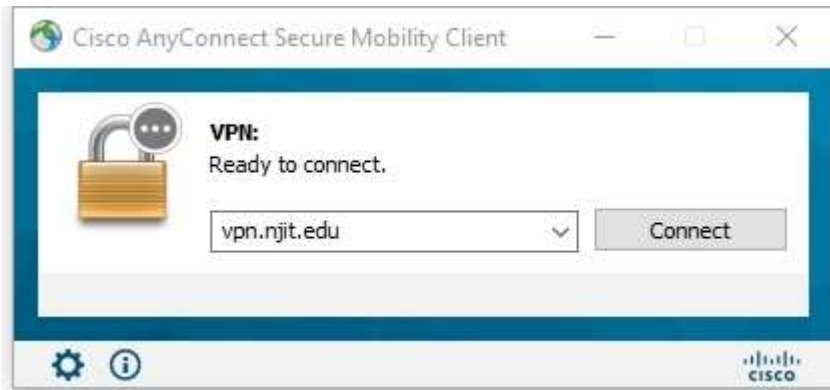


Figure 2.

2. The Connect message (Figure 2) on the button indicates not only that your machine is disconnected (i.e. the VPN client is not in use), but also indicates that your clicking on it will allow a connection to take place and thus you would be able to start using VPN. You are about ready to click Connect. In the next step you need to have ready your myUCID credentials (login and password) which must not have expired. After you click connect the pop up window of Figure 3 is shown.

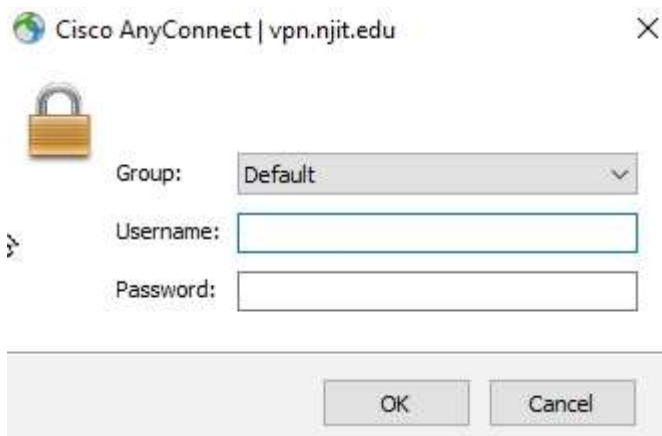


Figure 3.

3. The Username: field might have been pre-populated with your UCID. If not, type in your myUCID login in the Username: field, and then type in your myUCID password in the Password: field. The use of the TAB button (or SHIFT-TAB button) of the keyboard can help you moving around quickly. The Group: option can be left Default or you select an alternative according to the user instructions available during the VPN installation process or other information provided by NJIT.
4. If you have supplied the correct information (correct login name and correct associated password), a connection will be established and VPN would be in use and the popup window of Figure 3 will disappear. At that point if you try to locate the VPN icon using the instructions prior to step 1, the icon has a lock on it as shown below in Figure 4.



Figure 4.

5. If there was an active connection prior to step 2, and there will be an active connection after step 3, the window of Figure 2 would have looked like Figure 5. The button reads Disconnect since VPN is in use vs the Connect in Figure 2 when VPN was not yet in use. You can click on Disconnect to terminate the VPN connection when it is of no need any more. An alternative is to locate the icon shown in Figure 4, click on it, and it will allow you to disconnect and thus terminate the IN-USE VPN session.

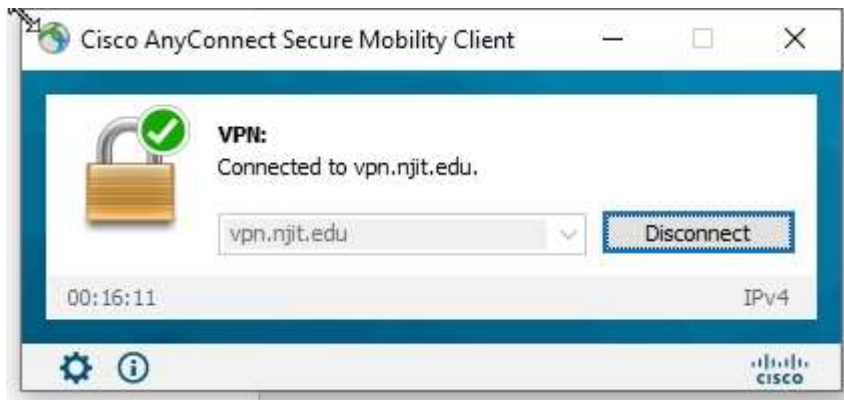


Figure 5.

At this point if you are outside of NJIT you have a VPN session in use. The next section deals with the interaction with a secure shell (ssh) client and how it can be used to connect to an OSL machine.

3. Secure Shell Client

By now, you are either in Case 2(b) or 2(c) or 2(d). If you are in case 2(a) you do not need a ssh client, since you can login directly to an OSL machine and each OSL machine has its own one preinstalled. If you are in Case 2(b) you have skipped the rest of the discussion of Section 2. In case 2(c) which is rare or case 2(d) you have completed successfully step 3, and you have a VPN session that is in-use (and of course active).

This means you have an authenticated, network connection-based presence at NJIT.

Download and Install a Secure Shell client (ssh) utilizing URL 1 of Section 1. An alternative is through URL 2. OSX and Linux machines have one pre-installed. The one available for Windows has file transfer capabilities using a graphical interface. Some NJIT provided machines also have it preinstalled. Thus for a Windows machine you may install the secure shell client known as MobaXterm, available to all at NJIT by NJIT, or go to the commercial MobaXterm web-site and download the limited feature free version found there. NJIT provides some info on MobaXterm through the link below that also includes info for secure shell to OSX users.

<https://ist.njit.edu/how-connect-afs-mobaxterm>

The discussion below is for a Windows client using Mobaxterm.

1. Invoke Mobaxterm. An icon on your desktop might be available for clicking on it. A window as shown in Figure 6 will pop up. Depending on your settings and customizations, it might look different from the one in Figure 6. Click on the button with the message **Start Local Terminal**

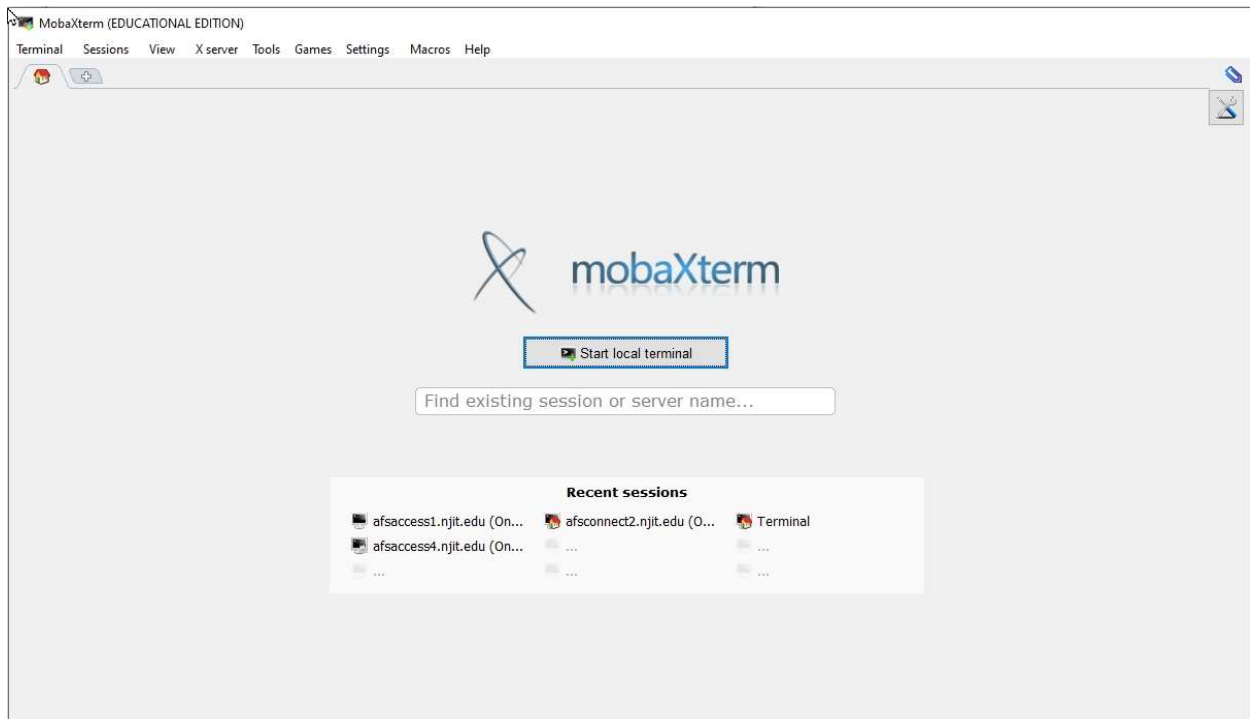


Figure 6.

2. After clicking the button "Start local terminal", a window like the one shown below in Figure 7 will pop up. That window has a **varying prompt (text string)** that ends with a right arrow and then next to the right arrow you may see a blinking cursor in the form of a rectangular box. This is a window running on your local computer, a client application. The prompt is a request by the application for you to provide input. We might call the application a shell. The shell is running on the client machine (your Windows laptop for example).

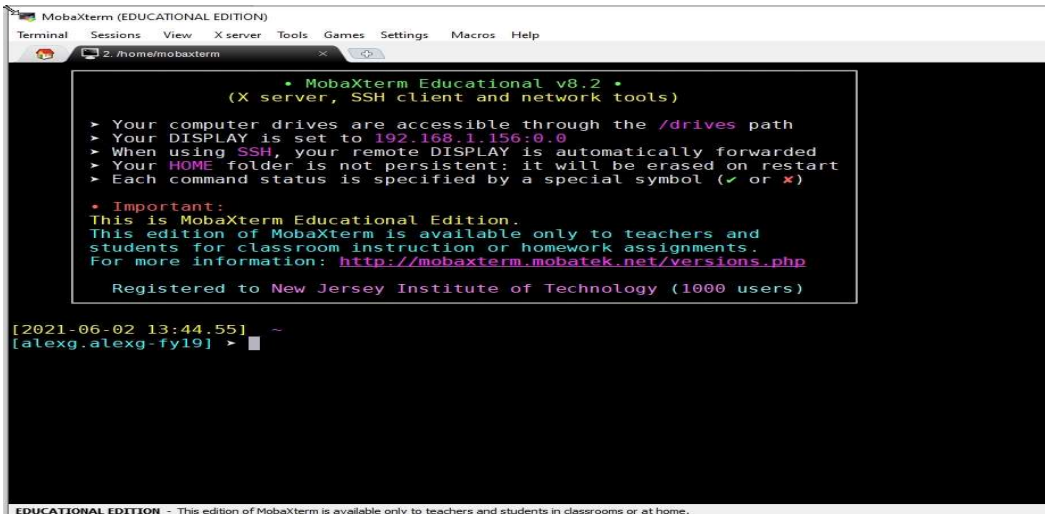


Figure 7.

3. Start writing using the keyboard a shell command that will be read by the client application (shell) of your Windows client machine. The command that you will type will be read by the client application (shell) and executed. The client application will then establish a Secure Shell connection to a remote OSL machine as specified by you in the typed command. As soon as a connection is established you would be writing down commands apparently on the client, but the window would be hosted by the remote host (machine/server), and the commands would be read and executed by the remote host instead.

For the example to follow we pick as a remote host (server) `osl7.njit.edu`.

The syntax of what you should type starting at the cursor's position would be

```
➤ ssh myUCID@host-name
where
```

`ssh` is the name the client program (shell); `ssh` stands for secure shell.

`myUCID` denotes your UCID login name and replace the string accordingly, and

`host-name` is the remote host name to which you want to connect. For this example `host-name` is `osl7.njit.edu`

Do not forget to press ENTER at the end of the typed line and every typed line.

A Warning message might be generated the first time you connect to this host and a password prompt is output for you to provide your myUCID password. The blinking cursor box is waiting for your myUCID password. Type it in and

Do not forget to press ENTER at the end of the typed password.

See Figure 8 below.

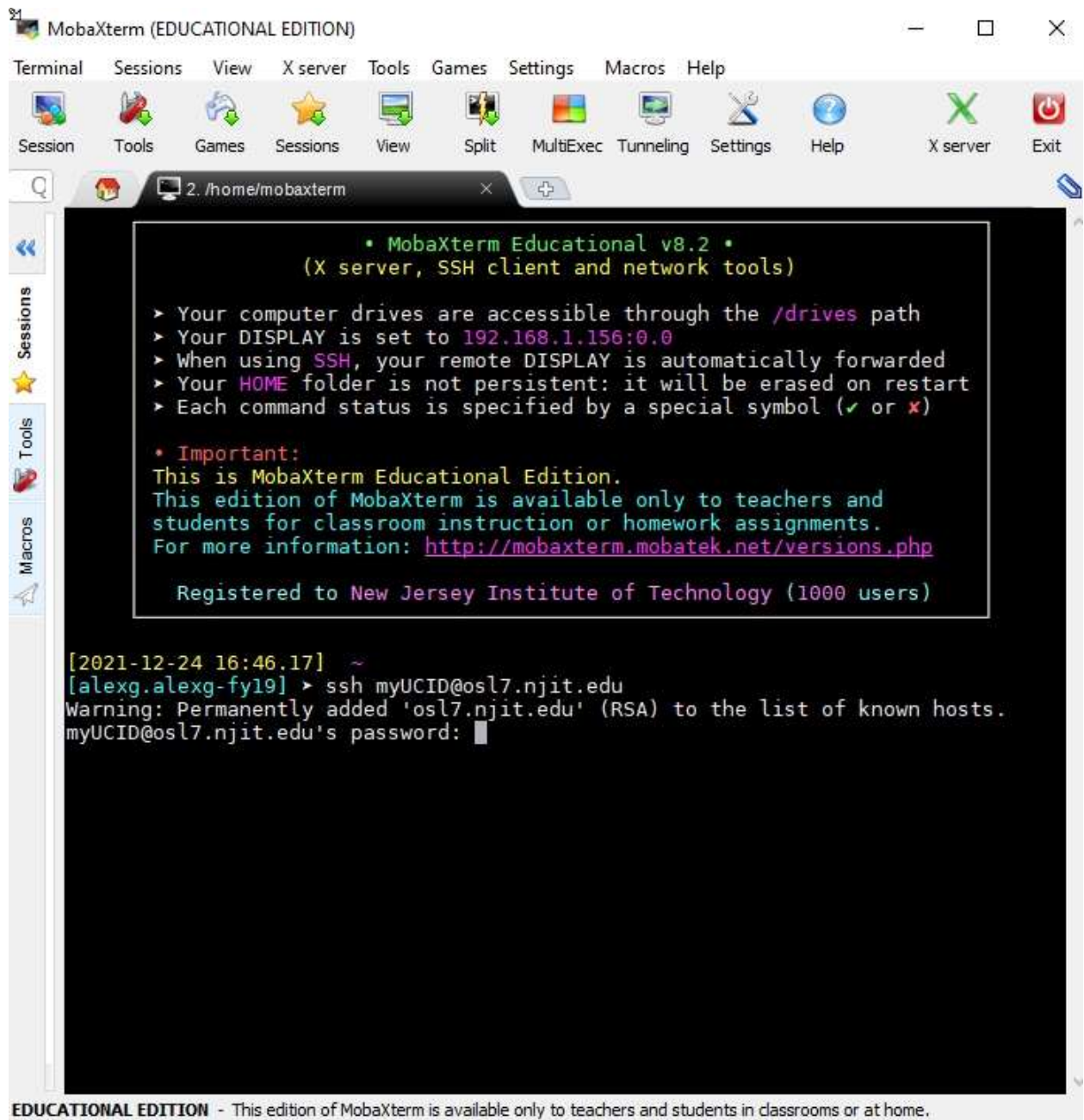
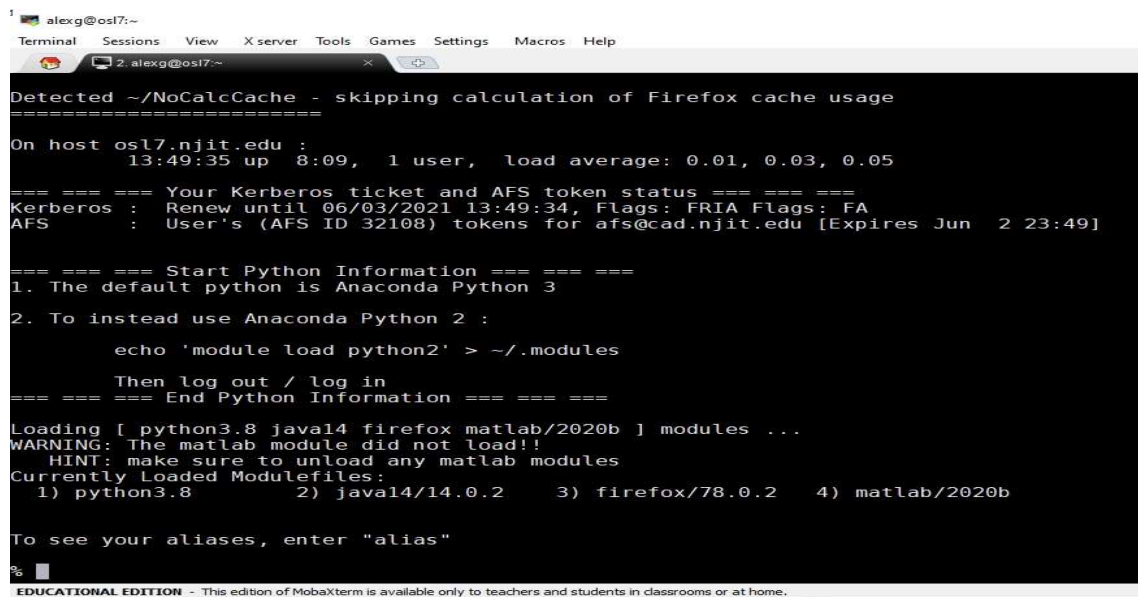


Figure 8.

4. After you typed your password and pressed ENTER you are logged on to the machine in question and of the given host-name). The screen might or might not look like the one in Figure 9. Looking at Figure 9 in the third line from the top you might read a "On host osl7...." verifying that you indeed got connected to osl7.njit.edu as intended.

The window of Figure 9. is a window hosted by the remote host (osl7.njit.edu). Although you will be typing commands in the client machine, your input will be transmitted to, read and executed and interacted by the remote host!

The remote host window has a different prompt as well. At the very bottom of the screen that is Figure 9 you see the prompt which FOR MY OWN SETUP (but maybe NOT FOR YOU) is the percent symbol % and after a space you may see the blinking cursor waiting for your input . The prompt and the cursor are customizable.



```
alexg@osl7:~
Terminal Sessions View X server Tools Games Settings Macros Help
2. alexg@osl7:~
Detected ~/NoCalcCache - skipping calculation of Firefox cache usage
=====
On host osl7.njit.edu :
13:49:35 up 8:09, 1 user, load average: 0.01, 0.03, 0.05

=== === === Your Kerberos ticket and AFS token status === === ===
Kerberos : Renew until 06/03/2021 13:49:34, Flags: FRIA Flags: FA
AFS      : User's (AFS ID 32108) tokens for afs@cad.njit.edu [Expires Jun  2 23:49]

=== === === Start Python Information === === ===
1. The default python is Anaconda Python 3
2. To instead use Anaconda Python 2 :
    echo 'module load python2' > ~/.modules
    Then log out / log in
=== === === End Python Information === === ===
Loading [ python3.8 java14 firefox matlab/2020b ] modules ...
WARNING: The matlab module did not load!!
HINT: make sure to unload any matlab modules
Currently Loaded Modulefiles:
  1) python3.8      2) java14/14.0.2   3) firefox/78.0.2  4) matlab/2020b

To see your aliases, enter "alias"
% 
```

Figure 9.

You might explore other options of MobaXterm. Moreover it is possible to upload files (from the client machine to the remote host) or download files (from the remote host to the client machine). You might see on the left side or the right side of the MobaXterm window, the file system directory area for your account. Read the MobaXterm manual

or instructions as made available by NJIT or through the MobaXterm web-site for more information.

At this point you are ready to start interacting with the remote host.

Type in Unix/Linux commands, when done typing them, press the ENTER button and observe the output.

If you are not familiar with Unix or Linux there are several tutorials or summaries out there. Pick the one you are more comfortable with it.

Finally, to terminate the session type exit. You will 'logout' from the remote host, and you will the move back to the familiar screen of Figure 7.

If you are done with MobaXterm, terminate it by clicking on its top right corner the X symbol that will 'kill' the window. Likewise locate the icon of Figure 4 and stop VPN. The icon of Figure 4 will revert to its form in Figure 1. At that point if you rightclick on Figure 1 you will be able to deactivate VPN. Deactivating VPN is NOT EQUIVALENT to an uninstall.

4. Unix and Linux

Linux is based on the Unix operating system (OS) and environment. The Unix operating system consists of a kernel (the part of the OS that is in main memory all of the time along with the data structures required for its proper operation) plus a variety of services that the operating system provides.

Unix was introduced in the late 60s/early 70s in the then AT&T Bell Labs by K. Thompson and D. Ritchie and was originally developed for a DEC PDP-11 minicomputer. It was originally written in assembly. In 1973 it was rewritten in the programming language known as C. The OS initial interaction with a user is done with some teletype terminals (also known as Video Display Units) to which a keyboard is attached. Thus interaction with a user was done through a device accepting and displaying 80 characters per line of fixed width ASCII output (and a total of 24 such lines). Think of this Courier font being the only option for terminal input (through the keyboard) and output (through the small 8-10inch screen of the terminal).

The UNIX operating system supported timeshared multitasking of user processes. A program in execution is known as a process and the OS's kernel is actively managing processes in main memory as opposed to a program that resides passively in secondary memory, i.e. a hard disk drive for which the OS knows nothing about its contents.

Timesharing means that multiple users have shared access of the CPU (processor) for a limited amount of time in a round-robin fashion. Thus over a period of roughly five seconds, five users have access to the CPU for roughly 1 second per user at a time in a round-robin fashion (1-2-3-4-5-1-2-3-etc). The period of 1second is known as the quantum or timeslice. The operating system's kernel is responsible for efficiently switching from one user to another user's process(es). If a user process has no activity during its timeslice/quantum in timeshared multitasking the OS would switch the usage of the CPU to another process of the same user or some other user. And if one process had to stop using the CPU to do an I/O operation (e.g. printing) then another user's process or another process of the same user could take over the CPU since multitasking is also supported. Whereas the objective of timesharing is to minimize CPU response time for processes, and the objective of multitasking is just to increase CPU utilization, the objective of timeshared

multitasking is to both minimize CPU response time and maximize CPU utilization.

In a Unix system, a user is logged on to the system by providing a set of credentials (login name and an associated password), a process already familiar to you from the earlier sections of this document. At NJIT we call the credentials MyUCID credentials consisting of a myUCID (login name) and a myUCID password.

Immediately after the login has been completed successfully the Unix environment would become available to the logged on user and a program (process) would start executing in the user's environment after the user's login: the Unix shell process.

The UNIX shell process would allow the user to interact with the operating system and start, stop, suspend and resume the execution of services provided by the OS or create and manage user created processes. These services are programs and when run they become processes. The definition of a process is 'a program in execution'. All interaction is done through the terminal and its associated keyboard that was used by the user to gain access to the system: the user types in commands to the shell and after the shell interprets these commands, it invokes services of the OS to execute/realize those user commands as needed and as privileged. The OS might decline to execute some of these commands for safety or security reasons based on the credentials (privileges) of the user.

To keep interaction short UNIX commands to the shell are short and sometimes intuitive. For example command **ls** is short for list, **cat** for catenate (list contents), **mkdir** for make directory, **ps** for process status, **cd** for change directory, **pwd** for print working directoy, **mv** for move, and **cp** for copy.

Moreover, the UNIX shell provides command line editing, and history of interactions with the shell thus allowing for editing a previously lengthy command instead of retyping it before re-execution or repeating a frequently executed command by easily recalling it from a history of prior interactions.

Every command of the shell such as **ls**, **ps**, **mv**, **cp**, **pwd**, **cat**, **cd**, is an executable program residing in secondary memory (disk). It was originally written in C and compiled and assembled

subsequently into the executable file named `ls`, `ps`, etc. Thus typing a command such as

```
% ls
```

would cause the shell (an OS process) to load the program named `ls` from secondary memory into main memory, thus turning it into a process and then executing it as needed.

A reminder: the `%` is the shell prompt. You do not type it. It is output by the shell to remind you that 'I, the shell, have your full attention: please type in your request'. Moreover when you do so and type your request (`ls` in this case) do not forget to tell the shell that you are done when you are done typing your command. You do so by pressing the ENTER key of the keyboard at the end.

At that moment the shell interprets your input (in the example above it is an `ls`), the text between the prompt and the ENTER, and executes it as needed. Every execution of a process in linux (in this case `ls`) by default creates and interacts with three files associated and connected with three devices:

- (a) standard input also known to the user as the file with fd (file descriptor) 0,
- (b) standard output with fd equal to 1, and
- (c) standard error with fd equal to 2.

Unless the shell is instructed otherwise by you, standard input is associated with your terminal's keyboard, and standard output and standard error are both associated with the terminal's screen.

Moreover in Linux, multiple commands can be executed one after the other in the command line. Thus

```
% ls ; date ; echo "Hi"
```

Thus in the above example after `ls` is executed, the current date and time is printed, and afterwards a message gets printed on the standard output, the terminal used by the user.

Moreover in Linux, multiple commands can interact with each other with a mechanism known as an unnamed pipe. A pipe is a FIFO (First In First Out) queue that accepts input from the output of one command and generates output that will become the input of another command. Thus

```
% ls | egrep filename
```

consists of the command `ls` that prints the contents of the current directory (this description makes sense after you read the next section if you are not familiar with any operating system's structure) and directs this output to the unnamed pipe indicated by the pipe `|` symbol. The unnamed pipe indicated receives as input the transmitted by `ls` output and then it generates its own output that is to become the input of the command `egrep`. The command `egrep` filters its input by discarding all lines that do not contain the string/word filename and thus preserving to the output that it will generate the lines that contain the string filename. The combined execution thus prints the output lines of `ls` that contain the string filename.

Pipes can allow multiple cascade communication such as the following one.

```
% ls -l | egrep filename | sort | less
```


5. Unix/Linux filesystem hierarchy

The Unix file system structure is hierarchical. This extends to Linux. The term file system has not been defined yet and it is thus being used generically at this point. In fact there are more than one type of a filesystem in Unix (and Linux) yet the discussion is generic and applies to all of them. The same interface is being used even if internally the systems are different. Moreover, on the osl machines at NJIT, to this hierarchical structure an external file system is further attached (mounted) that is known as AFS (for Andrew File System). It is a distributed file system with certain advanced features that we will not describe here. In summary AFS allows you to access your files independently on whether you are logged on to osl10.njit.edu or osl20.njit.edu or some other machine at NJIT that has access to AFS (including Windows or OSX machines but also Linux or Unix machines).

A Unix filesystem (or its structure) resembles a rooted acyclic directed graph (some people might view it as a tree) whose nodes are files: a filesystem of files! Since at this point we get on into discrete math territory, we won't pursue further those terms.

If there is a term overload let us start from scratch defining them in sequence.

A **hard disk drive** (HDD) can be split into logical subdivisions that are known as **partitions**. We won't discuss the hardware subdivisions of a hard disk drive: sectors, track, clusters, cylinders or other logical formations such as volumes.

A **partition** can be assigned a format. The format of a partition of a hard disk drive is known as a **filesystem**.

A **filesystem** (on a partition) can be created, mounted (to the OS and thus activated and can be viewed through the OS) or

unmounted. One can not destroy (delete) a filesystem directly: creating a new one on top of an older one overwrites the older one. The filesystem describes how a partition is organized logically into files and also describes the areas of the partition that stores information on those files and their data (metadata).

A **filesystem** contains **files** of different **types**.

A **file** is a collection of data on external memory (also known as secondary memory, and colloquially referred to as a hard disk drive). Thus a file is a collection (organized form) of data on a HDD's formatted partition (i.e. a filesystem).

Unix and Linux currently support several common **file types**. The most common ones are listed below. Associated with the file type's name we use a single character to represent and describe the file's type. Types of files in Unix and Linux are as follows.

- **:**(regular) file,
- d** :directory,
- l** :symbolic link, also known as soft-link,
- p** :named pipe,
- b** :block device file,
- c** :character device file, and
- s** :socket (used in networked communication).

In Unix every **file** is identified by a numeric identifier (value) known as its **inode (number)**. Inode stands for index node.

An inode value 10 indicates that the information for a specific file (the one with inode value 10) is available at index 10 of a table that is known as the **inode table**. The inode table was created when a filesystem was created on a partition of a HDD. Index 10 of that table contains information about the file identified with inode value 10 such as the size of the file in bytes, its file type, and other useful information including the locations on the hard disk drive that contain the data (contents) of the file.

Users do not like numbers (inodes) to reference files. An inode number such as 315156789 is difficult to memorize. Users prefer names. The term **filename** would then be established.

A `filename` is a mnemonic name that is associated with a given `inode number`. The association is effected inside a directory and the pair `(filename,inode number)` is recorded in the data of the directory (a directory is a file of type directory). In a given directory there can be only one pair with a given filename and given inode number `(filename, inode number)`. Different directories can however contain the same pair. Furthermore, it is possible that in the same directory we have another association of the the same inode number but with a different file name such as `(filename2, inode number)`. In other words the same file (inodenumber) has two names (aliases).

We furthermore prefer to say that a directory contains a filename rather than it contains a filename and inode number association. And sometimes two different filenames in the such directory such as filename and filename2 map to the same file of the same HDD.

In Unix and Linux we create a regular file by typing

```
% touch myfile
```

Several things happen with this 'command'

- (a) A file in the HDD is created by assigning a currently available inode number say N to the file that is to be established and the space in the inode table index N is initialized appropriately, for example setting the type of the new file to - (regular file), and
- (b) an entry is made in the directory into which the command touch was typed establishing the relationship `(myfile,N)`.

Subsequently the file creation process of myfile is continued and gets completed. (Note that the size of the created file would be zero bytes.)

Implicit in all this discussion is the fact that we, the user, know where we are in the hierarchical structure of the filesystem with which we interact. Thus "in the directory into which the command was typed" needs some explanation.

When a user logs on the system the location of the user in the filesystem is the user's home directory. The user can identify this location in two different ways either by typing

```
% pwd
```

for print working directory (which immediately after login is the user's home directory) or by typing

```
% ls ~
```

where tilde ~ is an alias for the user's home directory location. If a user is lost in the filesystem hierarchy a user can do a

```
% cd ~
```

and this moves him to the user's home directory, the starting location immediately after login. The command `cd ~` literally means change (the current) directory to become the home directory (of mine).

The hierarchical top of a Unix or Linux filesystem is **the root of the filesystem**. It is depicted by a slash symbol / and it is a directory, i.e. a file of type directory. It can contain files of any type including directories. The latter can be referred to as subdirectories since they are subordinate to the root directory /. A parent-child hierarchical relationship can then be established. The root is the parent of its subdirectories, and the subdirectories are the root's children.

Every file in the filesystem is associated with **an absolute path** that describes its location in the hierarchy relative to the root /, the common ancestor of all files (and of all types of files) in existence in the filesystem.

Thus the file with name filename might be associated with the absolute path

```
/afs/cad.njit.edu/u/u/s/user5/filename
```

This is to be read as follows.

- (a) start with the root / directory and read its data, and locate in the data of directory / a filename and inode number association for a file named afs,
- (b) then use the inode number of filename afs, to retrieve information about the file, confirm it is a directory (type) and access its data by reading its data contents and locating in it a filename and inode number association for a file named cad.njit.edu,
- (c) then use the inode number of filename cad.njit.edu, to retrieve information about the file, confirm it is a

- directory (type) and access its data by reading its data contents and locating in it a filename and inode number association for a file named u,
- (d) do the same for u and its file of type directory also named u, and
 - (e) do the same for u and its file of type directory s, and
 - (f) do the same for s and its file of type directory user5, and
 - (g) in directory user5 find the file named filename through the association (filename,inode number). This is the file in question. The inode of filename say 315156789 allows us to determine the type of file with inode 315156789 by going to the inode table and retrieving information about index 315156789.

The absolute path also provides us with some other hierarchical information. For example a 'child' of user5 is filename, or the parent of filename is user5. The parent of user5 is s. The parent of a file is always a directory that contains the file. By the way, the parent of the root / is the root itself, a directory. The root / is the only element of the hierarchy that is the parent of itself! A file with no children is a non-directory file (a file of a type other than directory) or an empty directory (without files i.e. children).

In the absolute path we observe two usages of the slash symbol.

The slash symbol is being used to denote the root (directory) of the file system. Subsequently the slash symbol is being used to separate directories (and arbitrary files) in the absolute path of a file. The absence of a slash symbol at the end of the absolute path for filename also indicates that filename is a file of type OTHER than directory. If it was a directory a slash would have been the last character of the path. (But different programs/commands use this inconsistently.)

A relative path can also describe a file such filename. First by using the command cd (change directory) we move ourselves (the logged on user) to a specific (directory) location in the hierarchical structure that is the filesystem as specified by the absolute path described next to the command cd (that must be a directory)

```
% cd /afs/cad.njit.edu/u/u/s/user5/
```

We can then confirm the current location with the command `pwd` (print working directory)

```
% pwd
```

```
/afs/cad.njit.edu/u/u/s/user5
```

and then file inquiries are relative to this directory

```
% ls filename
```

is then equivalent to an

```
% ls /afs/cad.njit.edu/u/u/s/user5/filename
```

in other words we are looking for information on filename in directory user5.

The **current directory** is denoted by (or aliased to) a dot. Thus

```
% ls .
```

and

```
% ls /afs/cad.njit.edu/u/u/s/user5/
```

are equivalent. The **parent of the current directory** is denoted by (or aliased to) two dots (no space in between) Thus

```
% ls ..
```

is equivalent to

```
% ls /afs/cad.njit.edu/u/u/s/
```

And of course

```
% cd .
```

has no effect as we request that we move to the current directory even if we are in it already. Note that relative paths are allowed when we use `cd`. Thus

```
% cd ..
```

moves to the parent of the current directory thus

```
% pwd
```

```
prints
```

```
/afs/cad.njit.edu/u/u/s
```

and then a relative

```
% cd user5
```

moves us back to the original location.

Note that if you are user5 the OS the moment you login and the shell runs, it automatically does (by itself)

```
% cd /afs/cad.njit.edu/u/u/s/user5/
```

for you. The indicated path (directory) is your **home directory**. The tilde symbol ~ is aliased to your home directory: it saves time typing it!

But beware of the following nuance.

Your home directory is not user5.

Your home directory is user5 of directory s of directory u of directory u of directory cad.njit.edu of directory afs of the root file system!

This is because it is possible that there are multiple user5 directories elsewhere in the file system hierarchy for example

```
/usr/local/user5/
```

```
/user5/
```

```
/bin/user5/
```

```
/user/local/bin/user5
```

```
/afs/cad.njit.edu/u/u/s/user5/user5/my.txt
```

Yes in your home directory there seems to be a directory user5 that contains a file name my.txt! (And the ! is an exclamation mark not part of the file name!)

In order to find information about the file with name filename we can type

```
% ls -l filename
```

and the output might look like as follows.

```
-rw-r--rw- 1 user5 group 1178078 Apr 26 12:14 filename
```

If we type the following the inode number of filename is also output.

```
% ls -li filename
```

```
315156789 -rw-r----- 1 user5 group 1178078 Apr 26 12:14 filename
```

If you want information about the inode number 315156789 stored in the inode table with index 315156789, this can be obtained through filename as follows.

```
% stat filename
```

The `-l` or `-li` is an option: the dash `-` alerts the operating system's shell that an option would be presented, and the `ell` indicates the long option (details about filename). The left most character of the output is the dash (left of the `rw`) that indicates the file type. When `-li` is typed two options follow the dash symbol one after the other with out space in between: the `l` option and the `i` option indicating a request to obtain the inode number associated with filename. The order does not ordinarily matter: we might have typed `-il`.

The nine character `rw-r-----` are the three triads that describe permissions for three entities associated with filename: the user owner of the file, the group of the user owner of the file, and everybody else. We refer to these entities generically as `u`, `g`, and `o` respectively. In this example `u` is `user5`, `g` is the group named `group`, and `o` everybody else i.e. neither `user5` nor anyone in the group named `group`. The permissions assigned by a triad to an entity are `read (r)`, `write (w)`, and `execute(x)` in the presence of the corresponding letter and are positionally dependent (`r` on the left of `w` on the left of `x`). The dash indicates the absence of the corresponding privilege/permission for the corresponding entity for filename. Thus `user5` has `r` and `w` privileges it can read and edit (write) the file named filename. The group (users other than `user5` of the group `group`) has only `r` privileges, and everybody else has none.

Observe another output below.

```
% ls -ld 2021linux
```

```
drwxrwxrwx 3 user5 afs 2048 Oct 14 2021 2021linux
```

It is clear that `2021linux` is a file of type directory. Note that the term `folder` is being used in Windows as an alternative to the term `directory`. This is not the case in Unix. Do not use

the term folder in Unix (including Linux). There is no file of type folder. In fact in Windows a folder can be a non-directory structure. The meaning of privilege x for a directory is different from that a file. All (user5, afs, and everybody else) are allowed to cd into directory 2021linux because of the x privilege. All can delete the directory or write into it i.e. create files (of any type) or delete files. Thus the dangerous

```
% rm -rf 2021linux
```

if executed would delete everything in 2021linux including directory 2021linux and all of its file recursively and completely. The OS won't ask you to confirm your recklessness. You explicitly specified f as an option in -rf to indicate "don't ask". Moreover the r of -rf is 'recursively'!

Directory user5 contains in the data of file/directory user5 a line

```
filename 315156789.
```

Thus the only way to find the alias to 315156789 is by looking inside the directory that contains the relationship between the alias (filename) and the actual name (inode) of the file (315156789). A file (such as 315156789) can have multiple aliases known as hard links. Thus it is possible inside user5 to have another entry

```
myfile 315156789
```

In fact somewhere else (in some other directory) it is possible to have the same. However we do know that this is not the case: this is because in the output of `ls -l filename` on the left side of user5 we see a 1. The 1 indicates one alias exists for 315156789 and that is filename. The operating system keeps track of all the associations with file 315156789.

If you do however a

```
% ln filename myfile
```

and then do a

```
% ls -l filename
```

or an

```
% ls -l myfile
```

or

```
% ls -li filename
```

or

```
% ls -li myfile
```

things would become interesting.

Below we use the symbol sharp #. The # indicates a comment for the shell and thus the remainder of the line is ignored by the shell when it tries to interpret and execute the line.

The commands (in fact executable files) we introduced that manipulate files of a filesystem or traverse the hierarchy of a filesystem are as follows.

```
% pwd          #print current working directory
```

```
% cd path      # change the current working directory to path
```

```
% ls path      # list contents of directory path
```

```
% ls           # list contents of current working directory
```

```
% ls -l        # long list
```

```
% ls -la       # detailed long list (. and .. included)
```

```
% ls -lai      # the inode (numeric name of the file) is also  
listed on the extremest left.
```

```
% rm file      # delete a file
```

or

```
% rm pathTofile
```

but

```
% rmdir directory      # remove an empty current directory
```

```
% rmdir pathTodirectory # remove an empty directory
```

```
% rm -rf file          # A pretty dangerous command... Avoid it.
```

```
% rm -rf directory    # A pretty dangerous command... Avoid it.
```

```
% mkdir directory # Create a directory
% mv oldname newname # rename file
```

6. Filesystem commands

```
% pwd # print working directory
% man pwd # manual page for pwd
% man man # manual page for man
% cd path # go to directory path
% cd / # go to the root of the file system
% cd # go to the home directory of user.
% cd ~ # same as cd
% cd . # go the current directory; do nothing
% cd .. # go one directory up (parent directory)
% cd ../.. # go two levels up (grandaparent)
% ls # list contents of . (current directory)
% ls path # list contents (filenames) of directory path
% ls directory # as above
% ls file # confirm file exists
% ls ~ # list files at home directory
% ls . # same as ls
% ls .. # same as above
% ls -l # detailed listing
% ls -ld # list directories
% ls -li # inode information included
% ls -lia # hidden files included
% ls -liaR # also recursively follow + list directories
% ls file* # list files whose names start with file
% ls *file # list files whose named end with file
% ls file{1,2} # list file1 and/or file2
% ls file[0-9].txt # list files file0.txt , ... , file9.txt
% rm filename # delete filename
% rm -f filename # silent removal
```

```

% rm -rf filename          # Dangerous: does not prompt to confirm
% rm -rf directory        # Even more Dangerous
% rmdir directory         # removes empty dir directory
% mkdir directory         # creates a directory
% mv olddir newdir        # renames olddir into newdir
% mv oldfile newfile      # renames oldfile into newfile
% mv file dir             # move file into dir
% cp file1 file2          # makes a copy of file1; new file named file2
% cp -r dir1 dir2         # copies a full directory
% stat file                # info about file (as found in inode table)
% cat file                 # show contents of file file
% less file                # similar to cat ; controlled output
% more file                # similar to less
% cat a.txt b.txt >>c.txt  # concatenate a.txt, b.txt into c.txt
% paste                    # explore it carefully (joins two files)
% head -10 file            # list first 10 lines of file
% tail -10 file            # list last 10 lines of file
% sort file                # sort lines of file
% sort -nr                 # see below
% sort -n                  # explore numeric based sort
% echo string              # display on screen (std output)
% echo $$                  # print process ID
% echo $?                  # print return value of an ending process (in
BASH shell)
% date                     #
% echo \$date              # observe
% echo ``date``           #
% echo $(date)             #
% chmod newpermissions file # file can be dir or regular file or of
any other type; does not work on AFS for directories
% ls -l file                # get old /current permissions
% chmod 572 file           #r,w,x,- interpret r for 4, w for 2, x for 1, -
for 0 and add ; from sum indicated find unique sum eg 5=4+0+1=r-x,
7=4+2+1=rwx, 2=0+2+0=-w-

```

Change file permission for a file with file name filename. The permissions are the privileges accorded to three entities, the user owner (u) of the file (ls -l indicates it is alexg in the examples of ls), the group (g) to which the user owner belongs (ls -l indicates the group as users), and everybody else (o) i.e. other than alexg and the members of group users.

For file with filename file1 above, the user owner permissions are rw- for read and write and no execute. Equivalently in binary 110 as a r,w,x maps to a 1 and a - to a zero (r maps to the leftmost 1, w to the second from the left and a - or alternatively a 1 to the rightmost 1). 110 in denary is $4+2+0=7$. Equivalently map an r into a 4, a w into a 2 and an x into a 1, and a - into a zero. Then rw- maps to $4+2+0=7$ as well.

```
% who                # who is logged on system
% whoami             # whoami ? what is my UCID (for NJIT)
% date
% hostname
% cal                #calendar for current year;
% cal 5 2023         #calendar for May 2023
% cal 23             #calendar for year 23...
% echo "hello" >out.txt # redirect output to file
% cat file1 >>file2   # add contents at end of file2
% cat file1 file2 >>file3 # at end of file3 add file1 then file2
% cat <file1 >>file2  # same as before last one
% wc file            # num of lines, words, characters

% ps -ef
% ps -ef | egrep root # processes of user root
% ps -ef | egrep root |more # same as above but controlled output
% cat | expand | col -b # Guess! Read man for expand, col
% sleep 10           # sleep for 10 sec; screen freezes
% sleep 10 &        # & is background; process in background
% jobs               # see it in background
% fg %1              # bring in foreground; may omit %1
% CTRL-Z            # suspend it
% bg                 # send it to background (again)
```

```

% module avail gcc          # list gcc versions available
% module load gcc/9.1.0    # activate 9.1.0
% module list              # list currently loaded module files
% module load cuda/9.0.2   # load NVIDIA CUDA module file
% module load texlive      # load Latex (default version is 2018 on AFS)
% gcc -v                   # which gcc version is active (loaded)?
% gcc prog1.c              # compile prog1.c ; exec in a.out
% ./a.out                  # Run (OS Loads+runs) a.out that becomes a
process
% gcc prog1.c -o prog1     # exec is prog1 not a.out
% gcc prog1.c -lm -o prog1 # does your code have math functions? Link with
math library
%./prog1
% which prog1              # seriously ?
% which ls                 # !
% gcc -S prog1.c           # assembly file is prog1.s
% more prog1.s
% gcc prog1.s -o prog1     # exec in prog1
% ./prog                   # runs
% gcc prog1.c -o prog1x
% md5sum prog1x prog1      # same fingerprints?
% ulimit -a                # see max Stack size; max user processes
% gcc -c prog1.c           # object code file prog1.o generated ; no exec
% gcc prog1.o -o prog1     # now generate it!
% ./prog1                  # run it
% echo $?                  # capture the return value of the main function
of prog1
% gcc -S prog1.s
% as prog1.s -o asprog1.o
% gcc -c prog1.s -o gccprogl.a.o
% md5sum asprog1.o gccprogl.a.o # what do you observe
% gcc -c prog1.c           # ?
% gcc agprog1.o -o exeas

```

```
% objdump -D prog1.o >dproglo.txt
```

```
% objdump -D exeas >dexeas.txt
```

BASH SHELL shortcuts CTRL-A means press CTRL key and A key at the same time.

```
CTRL-A    #move to start of bash line
CTRL-E    #move to end
CTRL-K    #delete/kill text from cursor position to end
CTRL-Y    #yank/paste  killed text
CTRL-L    #clear screen (terminal screen)
TAB       # autocomplete command or file name
UP DOWN   # move up or down command history
!cmd      # recall + execute more recent command starting
          # with text cmd
```

7. Tar, zip and gzip usage

How to create a tar archive on an OSL machine ?

TAR is a Tape ARchiver program. It takes as input multiple files and collates them into a large .tar file. The latter .tar file is in a format that is fit to be copied to a tape (and this was the intent long time ago). A .jar file in Java is in fact a .tar file containing Java class files. Therefore tar is versatile and universally useful. The following commands packs three files into archive file.tar

```
tar cvf file.tar a.txt b.txt c.txt
```

The command archives into archive file.tar the three files listed afterwards a.txt, b.txt c.txt. The following command


```
tar xvf file.tar
```

extracts from archive file.tar all the files stored in the archive. In our example these are files a.txt, b.txt and c.txt.

```
Options:  c for create,  
          v for verbose,  
          f for archive filename follows,  
          x for extract.
```

How to create a zip archive on an OSL machine?

```
zip file.zip a.txt b.txt c.txt
```

archives into file.zip a.txt b.txt c.txt

```
unzip file.zip
```

extract all the files archived in file.zip

```
FOOD for THOUGHT  
% cp abc.docx abc.zip  
% unzip abc.zip  
% mv def.xlsx def.zip  
% unzip def.zip
```

.xlsx .pptx .docx are in fact .zip archives!!!

Dealing with gzip on an OSL machine.

The tar archiver is just an archiver. It does not compress files. The zip program however archives and compresses files, and unzip dearchives and decompresses the archived compressed files. One way to compress file.tar is by doing what will turn file.tar into a compressed file.tar.gz.

Compress a file using gzip.

```
% gzip file.tar
```

Decompress a file using gzip (the d option in -d is for decompress)

```
% gzip -d file.tar.gz
```

and then of course if after decompression we are dealing with a .tar file we can dearchive by typing.

```
% tar xvf file.tar
```