KIT304 Server Administration and Security Assurance

Tutorial 1

Goal

In this tutorial you will explore the basic lab setup and familiarise yourself with some basic Unix commands that you will need over the coming weeks. Many students will have some familiarity with these commands from KIT104.

Introduction

Welcome to the first tutorial in KIT304. This unit will have a total of 24 tutorials – many more than you typically have in other units. The goal of all of these sessions is to not only equip you with knowledge, but also with practical skills in systems administration. The tutorials are arranged into four topics: Unix, Windows, Web & Database, and Network Security.

The first of these topic areas, and where you start today, is **Unix Administration**. In previous units you have been exposed to Unix commands and shell scripting. Today you will briefly revisit those skills before building on them for your subsequent tutorials. In the second tutorial of week 4 you will have a practical exam covering the material from these sessions, and a MyLO quiz that will cover the material in the weekly lecture and the readings on MyLO.

> ➢ **Note**: When completing this and future tutorials, be sure to read through any introductory material *before* attempting specific tasks. This material forms an important part of the content of the unit, and is often referred to in the subsequent tasks.
>
> ***Students who skip the background reading and head straight to the tasks will often miss important contextual information, which may be assessed.***

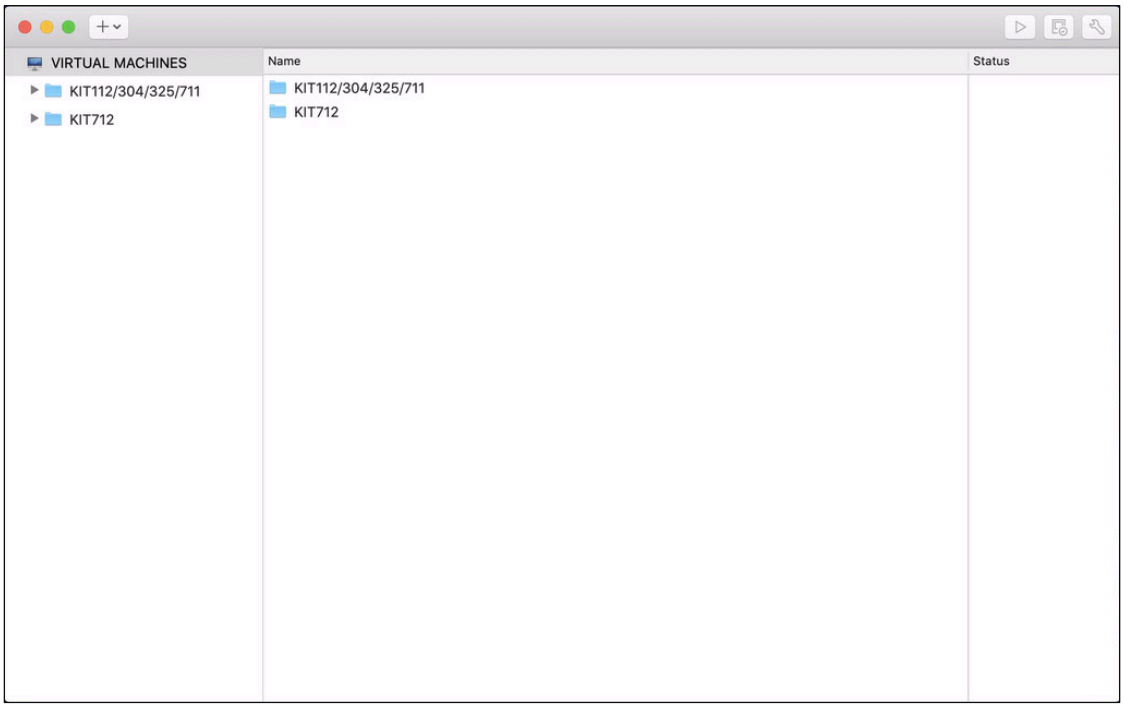
Objective 1 – Connecting to a Remote Lab System

In this unit, you will be using servers and clients running both Unix-derived and Windows operating systems. To do this easily on our lab machines, all of the servers and the clients are virtualised. This enables each computer in the lab to run multiple and different operating systems at the same time.

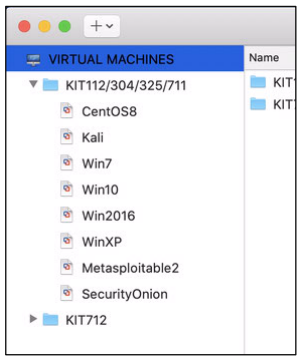
In the Unix Administration module, the main operating system that you will be using is the CentOS distribution of Linux. This distribution is derived from the same source base as Red Hat Enterprise Linux (RHEL). It was first released in 2004 and is a no-cost platform that users are free to redistribute. It is supported by an active community including system administrators, network administrators, managers, core Linux contributors, and Linux enthusiasts from around the globe. RedHat and CentOS together make up a very substantial portion of the enterprise server market share, which is why we have chosen it to be our Unix server in this unit. Additionally, Red Hat is the operating system that is used in the discipline for the ICT server **alacritas**.

For the first part of this session you'll cover LabShare and getting familiar with some of the virtual machines that you will be using throughout the semester. Prior to this tutorial you should have already downloaded the **ICT Labshare System** document in MyLO, installed the relevant VPN and remote desktop connection software, and tried to allocate yourself a system.

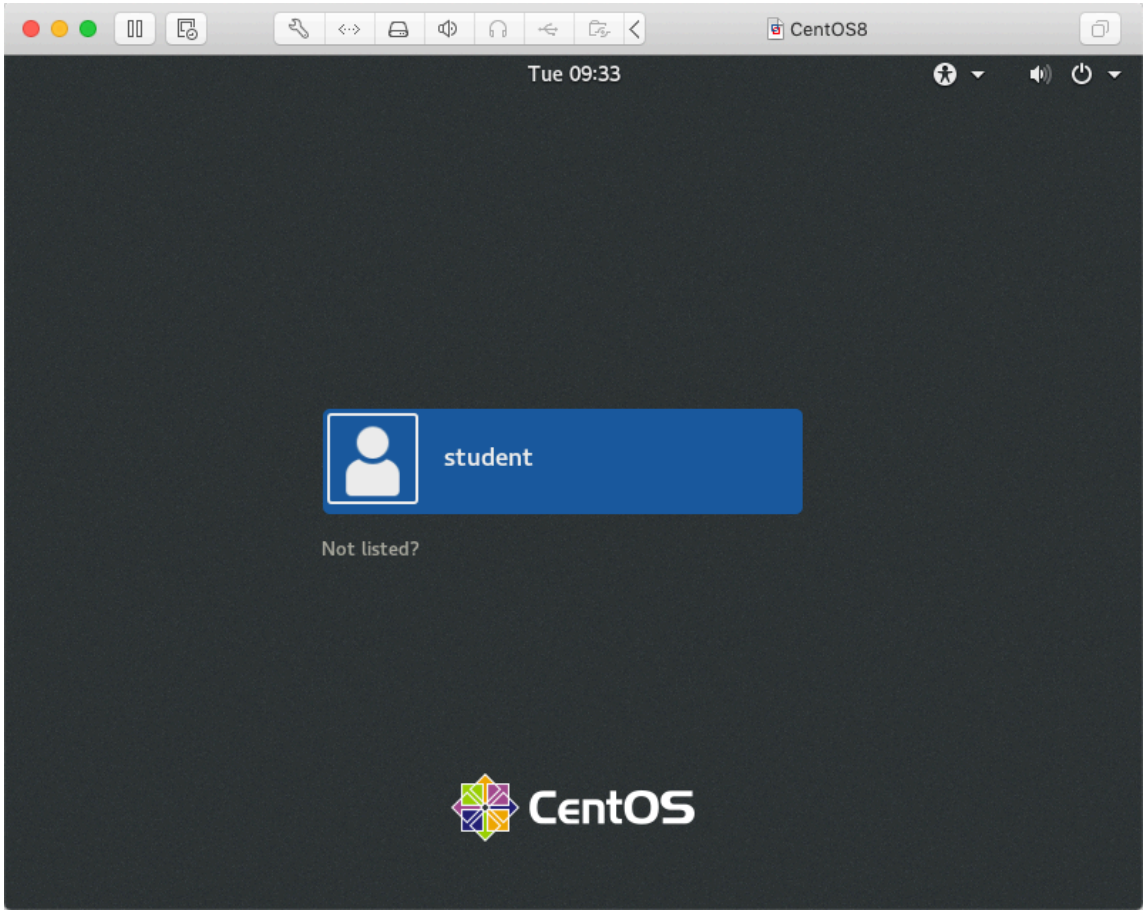
1. Using the **Labshare** system, allocate yourself a remote system and connect to it. When you're prompted for a connection password in the remote desktop client, **check that the username value is set to cyber** (if it's not, change it) and then enter the password shown on the LabShare web page. This password is unique to your current session, and will be changed at the end of your allocation. After a short pause, you'll see the remote connection window, and the remote system will automatically launch VMware Fusion – the virtualisation platform that you will be using to run the virtual machines used in this unit. You'll see a list of the supported virtual machines, as follows:



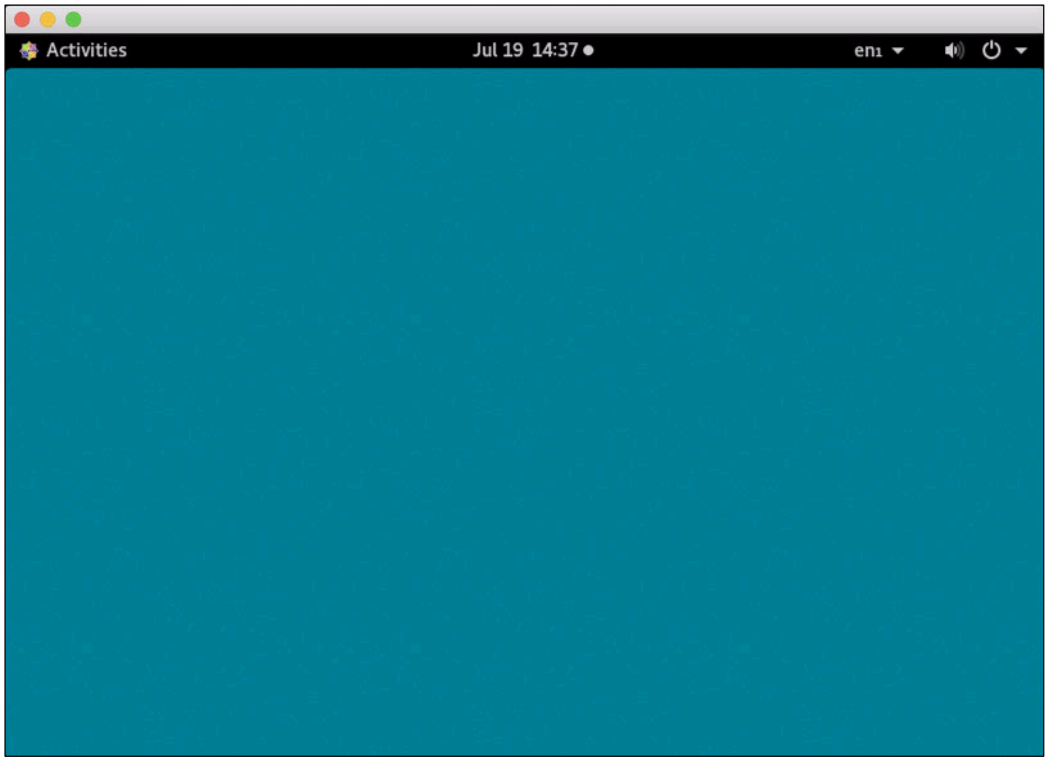
Click the triangle to the left of the KIT112/304/325/711 entry to expose the list of virtual machines you need, as shown on the right:



2. For the first part of this tutorial you will be using the **CentOS8** virtual machine (VM). While there a several ways that you can start a VM, the simplest is to just double-click the one you want to run. You could also select it with a single click, and then click the "play" button in right-hand side of the library window's title bar. After the virtual machine has booted (which takes around 90 seconds) you will be presented with a login screen which lists a single user named **`student`**:



Click on the username, and you will be prompted to enter a password. The password for the student account is also **`student`**. Type this and then either click the **Sign In** button or press **enter** to log on. When the account is ready, you'll see the CentOS desktop:



- Most of the VMs that you will be using in this course are fully integrated with the macOS desktop environment, and support a shared keyboard and mouse, as well as more advanced functions like copy-and-paste (for text), and drag-and-drop (for files). But for some operating systems (and for most operating systems during their boot-up and shut-down phases), there is no such integration, and when you activate a VM, it *captures* the mouse and keyboard. To release these from the VM so that you can access other VMs, or even other parts of the macOS user interface, press **Control-⌘** (if you're connecting from a Mac) or **Control-ALT** (if you're connecting from a Windows-based system).
- After it's running, you should be able change the size of the VM window (by dragging any corner or edge) to give you more desktop space in that VM. You cannot do this usefully with text-only VMs (like Metasploitable2, which you'll be using later in this tutorial).
- You can double-click the title bar of a VM window to expand it to fill the entire display, and double-click it again to restore it to its original size.

Obviously you would understand that for most VMs, you can resize them by dragging an edge or corner – but more useful is that you can alternate a window between its original size and an almost full-screen size by double-clicking the title-bar. This is a very useful feature that most students don't use, or don't know about – it is extremely useful when you have multiple virtual machines on screen at the same time. You can also **Merge All Windows** of running VMs into separate tabs in a single window via the **Windows** menu in VMware Fusion.

Objective 2 – Unix Command Line Basics

For the next part of this session you'll learn some basic Unix commands that you will need over the coming weeks.

3. In the CentOS virtual machine window, open a **Terminal** as follows:

   - click on the **Activities** item in the upper-left of the CentOS screen
   - click the **Terminal** icon in the Dock that appears on the left edge of the display

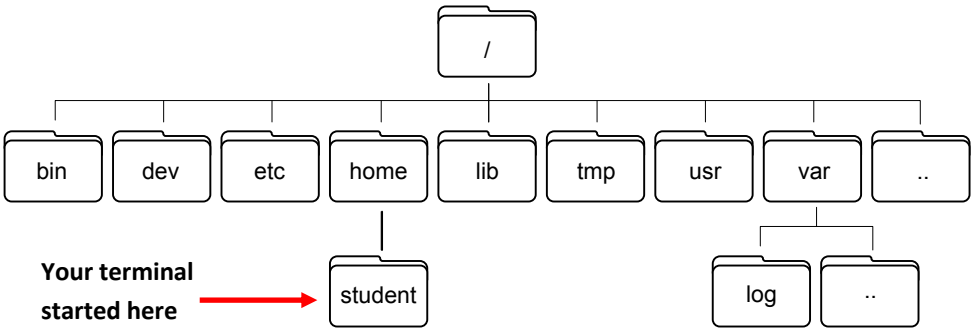
   Every process on a Unix system, including the Terminal, and the Unix shell that is running inside it, has the concept of a **current** or **working directory**. When you open a new Terminal window, the Unix shell running inside it will always start at your **home directory**. You can print your current directory within the Unix file system by entering the command **`pwd`** (which stands for **p**rint **w**orking **d**irectory).

   Enter the **`pwd`** command - what is the full path to your home directory?

   _____

   This shows that your home directory (**student**) is inside another directory (**home**), which in turn is at the *root* of the file system (denoted by the leading slash).

4. All operating systems have a default folder structure that stores all of the various system files. The general directory structure for Unix-derived operating systems is shown below.



   The primary function of each of these can be described as follows:

   - **`bin`** – binaries – the executable files that implement commands you enter
   - **`dev`** – representations of devices such as drives, printers, and screens
   - **`etc`** – system wide configuration files
   - **`home`** – a location that holds all of the user home directories
   - **`lib`** – libraries used by the binaries in **`/bin`**
   - **`tmp`** – temporary files, cleared at reboot

- **usr** – read only storage for user data and applications
- **var** – files that change in size. i.e., logs, email drafts, and print queues

You can use the **tree** command to show the tree-structured nature of the filesystem, e.g.

```
cd /
tree -dL 1
tree -dL 2
```

5. It is often useful to do a listing of a directory (with **ls**). Try the following variations:

ls displays a list of the files and folders in the current directory
ls / displays a list of the files and folders at the top (root) of the file system
ls /bin displays a list of files in the **bin** directory (which is at the root of the file system)

Write down the command to list the contents of the **log** directory, which is inside the **var** directory at the top of the filesystem:

_____

6. Within the Unix file structure, you can navigate around using the **cd** (change directory) command. For example, try this command:

cd /tmp change (move) from where you are now to the **/tmp** directory

With **cd** and many other commands, you can also refer to other parts of the directory structure with a few shortcuts:

- **~** (tilde) refers to your home directory
- **.** (period) refers to the current directory
- **..** (two periods) refers to the parent of the current directory
- **/** refer to the root of the file system

Experiment with these shortcuts and the **cd** command to move around the filesystem. How can you quickly get back do your home directory?

_____

7. Try entering each of the following commands to see what output they generate:

cd /tmp change (move) from where you are now to the **/tmp** directory
pwd confirm that you've changed your directory

`ls -al`	display a list of files and folders with additional information, like size and date *with this command, the very first character on the line indicates if an entry is a* ***directory*** *(starts with a* ***d****) or an ordinary file (starts with a -)*
`cd ..`	changes to the *parent* of the current directory
`pwd`	confirm your new directory location
`man ls`	display the **man**ual page(s) for the **ls** command

Those commands are simple, basic Unix commands that you should aim to become familiar with.

Write down the names of the ***directories*** that exist in the home directory of the student account:

_____

8. Next you will create a file. There are a number of tools that you could use to do this, but the two that it is recommended that you familiarise yourself with are the editors **`vi`** and **nano**.

   A "cheat sheet" of vi commands is attached to this tutorial sheet. **`vi`** can be challenging for beginners, as it operates in different modes (such as command mode and insert mode) and there are very few clues on screen to indicate what the current mode is.

   nano is simpler to use and has visual clues relating to what the commands are (used in conjunction with the control key). However, it is worth noting that some system files ***must*** be edited with **`vi`**, or **`vi`**-based editors, so it's very important that you have at least a passing familiarity with **`vi`**, as it is usually provided with most Unix-based operating systems.

   Follow these steps to create a short text file in your home directory using **`vi`**:

   - Type **`cd`** and press return, to ensure that you're back in your home directory
   - Enter the command **`vi textfile`** to create a new file named **`textfile`**, and start editing
   - **`vi`** starts up in **command mode**. In this mode you can use the arrow keys to move around and use various command keystrokes to give commands to **`vi`** (since there is no text in the file, you can't move around just yet).
   - Press the **`i`** key to enter **insert mode** (**`vi`** confirms this at the bottom left of the screen). In insert mode, you can enter any new text (at the location the cursor was at when you pressed **`i`**). In most modern versions of **`vi`**, you can also move around with the arrow keys while in insert mode (this is not the case for very old versions of **`vi`**).
   - Enter five or six lines of text so there's something to work with.
   - To get out of insert mode, press the **ESC** (escape) key. This always takes you back to **command mode**.
   - In command mode, in addition to the **`i`** (insert) command you used above, you can use the following commands (try each of these):

     a) **`A`** (*append*) enter insert mode at the end of the current line

b) **o** (*open*) open a new line after the current line, and enter insert mode
c) **O** (*open*) open a new line in front of the current line, and enter insert mode
d) **x** delete the single character at the cursor
e) **dd** delete the entire line at the cursor
f) **u** undo the last edit
g) **:x** save your edits and exit back to the Unix shell
h) **:q!** cancel all of your edits, and quit back to the Unix shell without saving

- There are many other commands you can use in **vi** (see the cheat sheet at the end of this document for more), but you've just covered those that will let you carry out most editing tasks. Editing in **vi** is mostly a cycle of switching between insert and command mode as you move around the document and make changes.

9. Using **vi**, create a few different text files, opening (and re-opening) them for editing, and saving your changes. ***You need to master this skill – get comfortable using basic vi!***

   When you're back at the Unix command line, you can remove files using the **rm** command, or copy them from one location to another with **cp**. You will be undertaking tasks in the coming weeks that require that you be confident with each of these commands, so be sure you're familiar with all of them by following these steps:

   a) Create a file (with **vi**) called **names.txt** and enter the names of 5 of your friends into it, one per line, then save and exit back to the Unix shell.
   b) Make a copy of **names.txt**, naming the new copy **friends.txt** – you would do this by entering the command **cp names.txt friends.txt**
   c) Decide that you don't really like the third person on the list, so edit **friends.txt** (in **vi**) and remove the third line.
   d) Rename **names.txt** to **colleagues.txt** – you would do this with by entering the command **mv names.txt colleagues.txt**
   e) Move the colleagues file into your **Documents** folder – you would do this by entering the command **mv colleagues.txt Documents**

   Note that the **mv** command (an abbreviation for **move**) can be used to both *rename* a file, and *move* a file to another folder (or subdirectory) – you used both variants above.

10. Directories are obviously different to files, and you use a different set of commands to create and manipulate them. To create a directory use the command **mkdir** and to delete a directory use the command **rmdir**.

    - Make a directory named **TextFiles**
    - Copy **friends.txt** into the new **TextFiles** folder
    - Try deleting the **TextFiles** folder with the **rmdir** command. What error did you see?

_____

Can you find a command that will remove the **TextFiles** folder even though there are files in it? If you're not sure, ask your tutor for a hint.

_____

11. It is often useful to do a listing of a directory (with **ls**) that is somewhere else in the file system than where you currently are. Are you able to figure out the syntax needed to do this? Write down the command to list the contents of the **log** directory, which is inside the **var** directory at the top (also known as the *root*) of the filesystem:

_____

12. In the terminal, navigate to the **/etc** folder, and generate a file listing of what is stored in it. You should be able to view the contents of most of these files using the **cat**, **head** and **tail** commands. What are the differences between these three commands?

_____

_____

_____

13. Try using the **cat** command to display the contents of the **shadow** file. This results in an error, which can be overcome with the **su** command. What is the purpose of the **su** command? Write your answer below. (Hint: use the **man** command to learn about it).

_____

14. Now that you know how to use the **su** command, use it to become the root user (this will require you to enter the root password when prompted, which is **toor**), then use the **cat** command again to view the content of the **shadow** file. What does the file contain? Can you find out what the different columns are for?

_____

_____

15. The shadow file itself is quite long. As a system administrator, you will often want to search through long files to find content that is of interest. **grep** is one command that you can use to do this. Are you able to use **grep** to find just the line that contains the information for your account?

You should experiment with **grep** and become confident with its use. It is very powerful, and extremely useful.

16. Use the **man** command to learn more about the **find** command, which allows you to search for files by name (and other criteria) across all or part of the Unix file system. Provide an example of the find command to locate the file **friends.txt** starting at the root of the filesystem.

---

17. If you've been following these steps as written, you should currently be running commands as the root user (from the **su** command, back at step 11). You can revert to the **student** account from **root** by typing **exit** (or just control-D). If you search the entire file system as the **student** account, you will see quite a few error messages saying that you don't have permission. Try this command to filter these out:

    ```
    find / -name media 2>/dev/null
    ```

    To explain briefly, the **2>/dev/null** portion of this command tells the system to silence any error messages by redirecting them to the **/dev/null** device, where they're effectively ignored. All commands run with two output streams – stream 1 is **standard output**, and stream 2 is **standard error output**. The **>** symbol redirects an output to the path listed after it - **/dev/null** in this case. So the sequence **2>/dev/null** tells the shell to redirect any error messages generated by the command to the /dev/null "sink".

18. As you already know, within the Unix directory structure you can navigate around using the **cd** (change directory) command. However, you can also refer to parts of the directory structure with a few shortcuts:

    - **~** (tilde) refers to your home directory
    - **.** (period) refer to the current directory
    - **..** (two periods) refers to the parent of the current directory
    - **/** refers to the root of the file system

    Experiment with these shortcuts and the **cd** command to move around the filesystem.

19. Your final command to consider today is **history**. This shows you the commands that you've already entered earlier. You can repeat an earlier command by typing a **!** (the exclamation mark, sometimes pronounced "bang") followed by the history number of that previous command, and then pressing return.

Conclusion

Hopefully the experience that you have had with Unix in earlier units in your course has come back to you during this tutorial, and you are comfortable navigating around the Unix file system, creating and opening files, and using such tools as **grep**.

In future tutorials you will be using many of the commands you've covered today, and it will be assumed that do know what they are, and how to use them. A cheat sheet that can serve as a quick look-up guide for many of these commands is attached to this tutorial, however you should take your own notes on how each command works as this will be far more useful in your practical tests.

In this tutorial we re-introduced you to the **man** and **info** tools. In coming weeks, you will need to research how to use commands by looking them up yourself. As the semester progresses you will be reading documentation from other sources so that you can complete tasks across the range of operating systems you are using. One of the sub-goals of this unit is to make you self-reliant, and able to use technical documentation to improve your skillset and knowledge.

> ➢ To shut down all of the virtual machines, and log out of the remote Macintosh, just release your allocation in the LabShare web interface. There is no need to manually shut down the VMs, and there is no need to log out of the remote Mac.

Skills to Master

To progress through this unit, and to be a system administrator of any Unix system, you need to master the following skills:

- list files in directories with **ls** and **ls -al**
- create and edit simple text files with **vi**
- move around the file system with **cd**
- understand and use relative paths (those that use **../** as part of the path name)
- **find** files in the filesystem
- display files with **cat**, **head** and **tail**
- filter files with **grep**
- understand the purpose of the main subdirectories, including **/etc**, **/bin**, **/usr**, **/var**, and **/home**
- redirect output with the **>** symbol

Anything identified in this skills section could be part of an assessment item in this module.

VI Quick Reference -- Basic Commands

Notations and Conventions

Commands you type are in `fixed` font.
`Italics` = substitute desired value.
<CR> means press RETURN key.
^X means press CONTROL and X keys together. Boxed commands switch to insert mode; press ESC key to end new text.

Beginning Your Edit Session

`vi` *`file`* <CR>	edit or create *file*
`vi -r` <CR>	show rescued files
`vi -r` *`file`* <CR>	recover rescued *file*

Ending Your Edit Session

`:q!` <CR>	quit and discard changes
`:wq` <CR> or `ZZ`	quit and save changes
`:wq` *`new`*<CR>	save as *new* and quit

Controlling Your Screen Display

`^R`	Eliminate @ lines
`^L`	Repaint screen after interruption
`:set wm=`*`x`* <CR>	Auto word wrap at *x* chars before line end
`:set nu` <CR>	Show line numbers on screen (not added to file)
`:set nonu` <CR>	Stop showing line numbers on screen

(put set commands into .exrc file for automatic settings each time you start vi)

Moving the Cursor

`h`	one position left
`k`	one line up
`j`	one line down
`l`	(letter "ell") one position right
`0`	(zero) beginning of current line
`$`	end of current line
`w`	forward one word
`W`	... including punctuation
`b`	backward one word
`B`	... including punctuation
`e`	forward to end of current word
`E`	... including punctuation
`-`	up one line, 1st non-blank char
`+` or <CR>	down line, 1st non-blank char
`H`	beginning of first screen line
`M`	beginning of middle screen line
`L`	beginning of last screen line

Paging Through Text

`^F`	forward one screen
`^B`	backward one screen
`^D`	scroll down half screen
`^U`	scroll up half screen
`n``G`	move screen to line number *n*
`G`	move screen to last line

Searching Through Text

`/`*`pattern`*	forward search for *pattern* (regular expression syntax)
`?`*`pattern`*	backward search for *pattern* (regular expression syntax)
`n`	repeat search for next occurrence
`N`	repeat search, reverse direction

Creating Text

Press ESC key to end new text. To enter a control character as text, precede it with ^V

`a`	append text after cursor
`A`	append text at end of current line
`i`	insert text before cursor
`I`	insert text before first non-blank character on current line
`o`	open new line after current line
`O`	open new line before current line

`:r` *`file`* <CR> insert contents of *file* after current line

Modifying Text -- Simple Changes

(* = can be preceded with a repeat count, e.g., `5x` or `16dd`. Count starts at cursor.)

`x`	delete character at cursor *
`dd`	delete line with cursor *
`dw`	delete current word *
`D`	delete to end of line.
`rc`	change character at cursor to *c*
`cw`	replace current word with new text *
`cc`	replace entire current line *
`C`	replace line from cursor to end
`J`	join current line with next
`~`	change case of current character
`u`	undo last text change
`.`	repeat last text-change command (could be at new location)

Modifying Text -- Operators

(Can precede with repeat count. Double to affect whole lines , e.g., `5yy`. Follow with one of the cursor movement or searching commands to select affected text, e.g., `dw` or `c5w` or `y/Geology<CR>`).

`d`	delete
`c`	change (replace with new text)
`y`	yank (copy) to buffer
`!`	filter selected text through command typed on status line, then replace with command output
`<<`	shift line(s) left one tab position
`>>`	shift line(s) right one tab position

Moving Text Around

(Use these commands to insert a copy of text from buffer. Deleted, replaced (old), or yanked text goes to unnamed buffer by default. Or use named buffer a thru z by prefixing command with " operator and buffer name, e.g., `"a5yy` and then `"ap)`

`p`	copy buffer text after cursor or line
`"ap`	... using named buffer a (or b,c, etc)
`P`	copy buffer text before cursor or line
`"aP`	... using named buffer a (or b,c, etc)
`xp`	transpose characters

Global Text Substitution

`:`*`n,m`* `s/`*`old`*`/`*`new`*`/g`<CR>

Change all occurrences of regular expression *old* to text *new* on all lines *n* thru *m*. Can use symbolic line numbers "." (current line) or "$" (last line).

Linux/Unix Command Line Cheat Sheet - GettingGeneticsDone.blogspot.com

Command	Description
pwd	prints working directory (prints to screen, ie displays the full path, or your location on the filesystem)
ls	lists contents of current directory
ls -l	lists contents of current directory with extra details
ls *~/user/*.txt*	lists all files in /home/user ending in .txt
cd	change directory to your home directory
cd ~	change directory to your home directory
cd */scratch/user*	change directory to user on scratch
cd -	change directory to the last directory you were in before changing to wherever you are now
mkdir *mydir*	makes a directory called mydir
rmdir *mydir*	removes directory called mydir. mydir must be empty
touch *myfile*	creates a file called myfile. updates the timestamp on the file if it already exists, without modifying its contents
cp *myfile myfile2*	copies myfile to myfile2. if myfile2 exists, this will overwrite it!
rm *myfile*	removes file called myfile
rm -f *myfile*	removes myfile without asking you for confirmation. useful if using wildcards to remove files ***
cp -r *dir newdir*	copies the whole directory dir to newdir. –r must be specified to copy directory contents recursively
rm -rf *mydir*	this will delete directory mydir along with all its content without asking you for confirmation! ***
nano	opens a text editor. see ribbon at bottom for help. ^x means CTRL-x. this will exit nano
nano *new.txt*	opens nano editing a file called new.txt
cat *new.txt*	displays the contents of new.txt
more *new.txt*	displays the contents of new.txt screen by screen. spacebar to pagedown, q to quit
head *new.txt*	displays first 10 lines of new.txt
tail *new.txt*	displays last 10 lines of new.txt
tail -f *new.txt*	displays the contents of a file as it grows, starting with the last 10 lines. ctrl-c to quit.
mv *myfile newlocdir*	moves myfile into the destination directory newlocdir
mv *myfile newname*	renames file to newname. if a file called newname exists, this will overwrite it!
mv *dir subdir*	moves the directory called dir to the directory called subdir
mv *dir newdirname*	renames directory dir to newdirname
top	displays all the processes running on the machine, and shows available resources
du -h --max-depth=1	run this in your home directory to see how much space you are using. don't exceed 5GB
ssh *servername*	goes to a different server. this could be queso, brie, or provolone
grep *pattern files*	searches for the pattern in files, and displays lines in those files matching the pattern
date	shows the current date and time
anycommand* > *myfile	redirects the output of anycommand writing it to a file called myfile
date > *timestamp*	redirects the output of the date command to a file in the current directory called timestamp
anycommand* >> *myfile	appends the output of anycommand to a file called myfile
date >> *timestamp*	appends the current time and date to a file called timestamp. creates the file if it doesn't exist
***command1*	*command2***	"pipes" the output of command1 to command2. the pipe is usually shift-backslash key
**date	grep *Tue***	displays any line in the output of the date command that matches the pattern Tue. (is it Tuesday?)
tar -zxf *archive.tgz*	this will extract the contents of the archive called archive.tgz. kind of like unzipping a zipfile. ***
tar -zcf *dir.tgz dir*	this creates a compressed archive called dir.tgz that contains all the files and directory structure of dir
time *anycommand*	runs anycommand, timing how long it takes, and displays that time to the screen after completing anycommand
man *anycommand*	gives you help on anycommand
cal -y	free calendar, courtesy unix
CTRL-c	kills whatever process you're currently doing
CTRL-insert	copies selected text to the windows clipboard (n.b. see above, ctrl-c will kill whatever you're doing)
SHIFT-insert	pastes clipboard contents to terminal

*** = use with extreme caution! you can easily delete or overwrite important files with these.

Absolute vs relative paths.

Let's say you are here: /home/turnersd/scripts/. If you wanted to go to /home/turnersd/, you could type: **cd /home/turnersd/**. Or you could use a relative path. **cd ..** (two periods) will take you one directory "up" to the parent directory of the current directory.

.	(a single period) means the current directory
..	(two periods) means the parent directory
~	means your home directory

A few examples	
mv myfile ..	moves myfile to the parent directory
cp myfile ../newname	copies myfile to the parent directory and names the copy newname
cp /home/turnersd/scripts/bstrap.pl .	copies bstrap.pl to "." i.e. to dot, or the current directory you're in
cp myfile ~/subdir/newname	copies myfile to subdir in your home, naming the copy newname
more ../../../myfile	displays screen by screen the content of myfile, which exists 3 directories "up"

Wildcards (use carefully, especially with rm)

*****	matches any character. example: **ls *.pl** lists any file ending with ".pl" ; **rm dataset*** will remove all files beginning with "dataset"
[xyz]	matches any character in the brackets (x, y, or z). example: **cat do[or]m.txt** will display the contents of either doom.txt or dorm.txt