An Introduction to UNIX, SSH & VNC

For many of you this may be the first time that you have used the UNIX operating system. Whereas it has been vastly improved to work in a similar way to Mac OS X and Windows operating systems, the terminal has remained an extremely powerful way to manipulate file systems and software. In this workshop, anything which is contained in ⟨⟩ brackets should be replaced by what is described in the brackets, e.g. ⟨FileName⟩ = iLOVEcheese.txt.
Before we begin on our UNIX journey, we need to understand the general format of a UNIX command line. A typical command line looks like this:

```
user@computer /directory > command -options targets
```

The user and computer is pretty self explanatory. The directory describes the current folder that you are located in. To the right of the >, there is a typical UNIX command. The command is the name of a built-in shell command followed by its 'arguments' (the options and the target filenames/expressions).
The first command which you should memorise is **man** because it is EXTREMELY USEFUL. It shows the manual for any UNIX command:

man ⟨UNIXcommand⟩ = reveals all you need to know about a UNIX command!! Use it!

1 Terminal Navigation

The next step is to start navigating about your terminal. Below are some essential commands to get you started. These also include useful arguments which make the command work slightly differently:

ls	= lists the files in your current directory
ls −a	= lists files and folders
ls ⟨FolderName⟩	= lists files in the folder
ls −lh	= a detailed list, and human readable
ls −l *.txt	= lists .txt files only (.txt can be replaced by any end of a file name)

cd ⟨FolderName⟩	= used to change directory to FolderName
cd /	= goes to the root of the system
cd ..	= goes up one folder, e.g. cd ../../ goes up two.

** Tip: if your folder has spaces use "" around the folder name. An easier way is to put _ instead of spaces when creating files. **

du −h = disk usage of folders, human readable

du −ah = disk usage of files and folders, human readable

** Tip: the **du** command is useful for finding large files in your home area, don't overload your to your home area otherwise the network will suspend your account. Use **df -h** to see how full your area is. **

pwd = prints the working directory, super helpful to know where you are!

top = shows the current processes occuring on your system

1.1 Mozilla Caching

An important thing to do is to manage your home area data allocation. A big problem is caused by Mozilla caching data. If your home area becomes full then your account will stop working. It is advised to put any large downloads onto your scratch disk (e.g. /scratch/username) on the local computer.
To free up space, you need to delete the mozilla cache and then stop mozilla caching data. This should only need to be done once, so do it now!
To delete the cache, use the command cd to go to this directory:

```
cd  /.mozilla/firefox/
```

Here there should be some random folders and profiles in here. Use `rm -r FolderName` to delete the folders with the random assortment of letters and numbers at the end. This should delete the cache. An unfortunate side-effect of this is it will delete your history but as it only needs to be done once then it is worth it.
You now need to stop mozilla caching. Open up mozilla and click on Settings, then Advanced and finally Network. There should be a tick box with Override Cache Management. Click on this and set the Limit Cache box to 0. This will now prevent caching!
Now that you have got a hand on the basics, have a go at exercise 1 (see the end of the document).

2 File Manipulation

So you now know the basics of moving around the UNIX system. The next step is to manipulate those files! Here are some commands for you to do just that:

cat⟨FileName⟩	= shows the contents of a file
head−n ⟨no.ofLines⟩ ⟨FileName⟩	= reads the file from the top
tail−n ⟨no.ofLines⟩ ⟨FileName⟩	= reads from the bottom

Nb: the **–n ⟨no.ofLines⟩** portion is optional.

mkdir elephant	= creates folder called elephant in current working directory
mkdir elephant/banana	= creates folder called banana in elephant

cp banana.jpg apple.jpg	= copy & renames file banana.jpg to apple.jpg
cp banana.jpg ⟨FolderName⟩/	= copies banana.jpg to a folder
cp −R ⟨FolderName⟩ ⟨FolderName2⟩	= copies & renames FolderName to FolderName2
cp *.txt ⟨FolderName⟩/	= copies all .txt files to a folder (.txt can be replaced)

** There is other copying procedures, namely **rsync** and **scp**, which are used for file transfers across secure networks. They are introduced in sections 3 and 4.

mv banana.jpg Folder1/	= moves banana.jpg to Folder1
mv ⟨FolderName⟩ ⟨FolderName2⟩/	= moves folder1 into folder2
mv banana.txt apple.txt	= renames file

rm ⟨FileName⟩	= deletes a file
rm −i ⟨FileName⟩	= deletes after asking for confirmation
rm −f ⟨FileName⟩	= forces deletion of the file
rm −r ⟨FolderName⟩	= deletes a folder

It is also possible to include links from one file or directory to another using the **ln** command. This has the form:

ln ⟨FileName⟩ ⟨LinkName⟩	= creates a hardlink between the file and link
ln −s ⟨FileName⟩ ⟨LinkName⟩	= creates a symbolic/soft link

Note that if the linkname of filename is change, the change will be reflected in the other. There are a few subtle differences between soft and hard links. A soft link can be created to a file which does not exist and can be used to link different physical disk devices/partitions whereas a hard link cannot.

Now do Exercise 2.

3 Researching Files

The next step in becoming a linux master is finding files and searching its contents. This is especially useful if you have a large amount of files or large text files/ databases. UNIX has a few commands which you will find helpful for doing this. You can use the command **locate** but this is very very slow. It is advised to use **find** and here are some useful arguments:

find −name "⟨FileName⟩"	
find −name "text"	= search for files that start with the word text
find −name " ∗ text"	= " " " end " " "
find −size +10M	= find files larger than 10Mb (can be replaced)*1
find −name "⟨filetype⟩"−atime−5	= search last access
find −type d	= search for only directories
find −type f	= search for only files

*1 M represents megabyte and can be replaced by K or G. The + represents more than and can be replaced by - for less than. No sign means the file is exactly that size.
One fo the most important commands is **grep**. This is essential if you would like to find certain phrases etc. in a long file.

grep ⟨text⟩ ⟨FileName⟩	= search for text within a file
–i	= don't consider upper case words
–I	= don't consider binary files
grep −r text⟨FolderName⟩	= search for files names with occurence of text
grep −E text⟨FileName⟩	= search start of lines
grep −E ⟨0−4⟩ ⟨FileName⟩	= show lines with no. 1-4
grep −E ⟨a−zA−Z⟩⟨FileName⟩	= retrive lines with alphabetical letters

In fact, grep is so important it has its own exercise you may do if you like. See the end of the document.

4 File Permissions

Changing file permissions is important if you would like to protect your data from intruders or allow it to be used by other people on the network. Where the user of the file cannot be changed without admin, the **chmod** command can let you change the permissions of your own files. Below are a list of arguments for chmod and an example.

u	= user
g	= group
o	= other

r	= read permissions
w	= write permissions
x	= execute, useful for scripts
+	= add a permission
−	= remove a permission
=	= affect a permission

For example, I have a file called banana.py and I would like anyone to be able to run this script. I can use **chmod** to add a permission to execute for other users i.e.:
chmod o+rx banana.py
This can be also expressed as a sequence of 3 octal digits. Each octal digit represents the access permissions for the user, group and other respectively. The mapping of the permissions onto these digits is as follows:

Permission	Digit
—	0
–x	1
-w-	2
-wx	3
r–	4
r-x	5
rw-	6
rwx	7

The previous example can now be restated as:
chmod 005 banana.py

Now see Exercise 3

5 More Advanced UNIX: Piping, Redirecting and Process Control

A process is a program in execution. Every time you invoke a system utility or an application program from a shell, one or more "child" processes are created by the shell in response to your command. All UNIX processes are identified by a unique process identifier or PID. This section shall follow a few examples to make it easier to understand.

5.1 Piping

The pipe ('|') operator is used to create concurrently executing processes that pass data directly to one another. It is useful for combining system utilities to perform more complex functions.

```
cat banana.txt | sort | grep "banana"
```

In this example, the output of cat is passed onto sort which in turn is passed onto grep to find all instances of the word banana. The output of grep is shown on the terminal.

5.2 Redirection

To redirect standard output to a file instead of the screen, we use the > operator:

```
echo hello
```

hello

```
echo hello > output
cat output
```

hello

In this case, the contents of the file output will be destroyed if the file already exists. If instead we want to append the output of the echo command to the file, we can use the » operator:

```
echo bye » output  cat output
```

hello bye

To capture standard error, prefix the > operator with a 2 (in UNIX the file numbers 0, 1 and 2 are assigned to standard input, standard output and standard error respectively), e.g.:

```
cat nonexistent 2>errors
cat errors
cat:  nonexistent:  No such file or directory
```

You can redirect standard error and standard output to two different files:

```
find .  -print 1>errors 2>files
```

or to the same file:

`find .  -print 1>output 2>output` or `find .  -print >& output`

Standard input can also be redirected using the < operator, so that input is read from a file instead of the keyboard:

`cat < output` hello bye

You can combine input redirection with output redirection, but be careful not to use the same filename in both places. For example:

```
cat < output > output
```

will destroy the contents of the file output. This is because the first thing the shell does when it sees the > operator is to create an empty file ready for the output.

5.3 Controlling Processes

An important lesson to learn with UNIX is how to control processes in your terminal. As stated before, each process has a unique PID number. This is essential to know if you would like to change how this process works. To find the PID of current processes on your current terminal you can use the command **ps**. This gives the PID of each command on your terminal.
To kill a command you can used the command **kill** with the PID as the argument. E.g. `find` has a PID of 27503 and we would like to kill it:

```
kill 27503
```

Sometimes this may not kill the process so you may need to force it to shut down. This should be used as a last resort. To do this the argument **-9** should be placed after the kill order.

```
kill -9 27503
```

Now see Exercise 4

6 Tips and Tricks

There are some tips and tricks to get your UNIX skills up to the ranks of a wizard. The following section shall show some simple shortcuts to make your life a whole lot easier.
Probably the most important shortcut is the use of the TAB key. TAB allows you to autocomplete a command or file location within the terminal e.g. if I wanted to move to my scratch folder I could write `cd /scratch/radcliff/`. Alternatively I could type `cd /scratch/rad TAB`. This would autocomplete

to radcliff. Note that if there is two files starting with rad, a list of options will be given and you need to write more of the word until it is unique.

CTRL + L	= Clear the Terminal
CTRL + D	= Logout
CTRL + A	= Cursor to start of line
CTRL + E	= Cursor to end of line
CTRL + Z	= Stops current command. Resumed with fg or bg
CTRL + C	= Intterupts and kills the current process

The background is useful for freeing up your current terminal. You can add a & after the command to instantly take it to the background. If you suspend a program using CTRL+Z, you can resume it using the command **fg** or **bg** to send it to the foreground and background respectively. If a program runs in the foreground, the terminal you are using becomes unusable until the job finishes.
That's the end for the basics of UNIX, now it is time to get down to grips with some more useful commands which carry over a network such as ssh and VNC

7 SSH

SSH, which is an acronym for Secure SHell, was designed and created to provide the best security when accessing another computer remotely. Not only does it encrypt the session, it also provides better authentication facilities, as well as features like secure file transfer, X session forwarding, port forwarding and more so that you can increase the security of other protocols.
To ssh into another system, you need to know its system address. If this is within the local network, it is the name of the computer. A typical ssh command to the computer hegemone looks something like this:

```
radcliff@server1 > ssh hegemone
```

This will ask you for you password and you will get access to the file system of the other computer. An important variant of ssh is called session forwarding. This allows you to access files on other systems and open them locally. A typical command for port forwarding is:

```
radcliff@server1 > ssh -X hegemone
```

It is possible to set up ssh to remember your password on systems that you have used before. This is called ssh-keys. An excellent guide for setting this up is found on:
`https://www.digitalocean.com/community/tutorials/how-to-set-up-ssh-keys--2`
Files can also be transferred across an ssh protected network using a command called **rsync**. Rsync works similarly to cp in that its syntax is as follows:

rsync ⟨FileName⟩ ⟨Destination_Folder_Name⟩/ = copies file to a destination folder

To send the file across networks we change the syntax to be:

rsync ⟨FileName⟩ User@destination_system : /path/Destination_Folder_Name

The addition of argument **-ar** or **-R** allows folders to be sent across the network. For example, we want to copy folder elephant in almap7's raid2 drive to the nas drive in the computer hegemone. We are currently located in the root. The syntax would be:

```
 radcliff@almap7 / > rsync -R /raid2/elephant/ radcliff@hegemone:/nas/
```

Likewise we can swap the argument, if we would like to copy this back. The manual for rsync is especially useful.

8 VNC

8.1 VNC within a Network

A VNC session is an interactive virtual desktop of a system on the network streamed to your home computer. It is useful when accessing machines with no desktop output, such as the ALMA computers or when you need systems with a higher computational power. Here is a short guide for setting up a VNC on other systems within the local network. In the example, hegemone is the home computer and almap7 is the system we want to get a vnc session on:

1. Before you start you need to find the size of your monitor. This can be done on the monitors' menu or use the command **xdpyinfo | grep 'dimensions:'**. Note this down.

   ```
   radcliff@hegemone / > xdpyinfo | grep 'dimensions:'
   dimensions:  2880x1200 pixels (762x318 millimetres)
   ```

2. Next we need to create a vnc server. To do this, ssh into the computer you want to use.

   ```
    radcliff@hegemone > ssh almap7
   ```

3. Once done, use the command **vncserver** and the argument **-geometry** ⟨**dimensions**⟩ replacing dimensions with the values found in step one. The output should look similar to what is stated below. Make a note of the address of the vnc system. In the example this is **almap7.jb.man.ac.uk:10**

   ```
   radcliff@ALMAP7   > vncserver -geometry 2180x1100
   New 'almap7.jb.man.ac.uk:10 (radcliff)' desktop is almap7.jb.man.ac.uk:10
   Starting applications specified in /home/radcliff/.vnc/xstartup
   Log file is /home/radcliff/.vnc/almap7.jb.man.ac.uk:10.log
   ```

   It is advised to use a slightly smaller geometry than your actual desktop monitor so that the vnc window can fit within the monitor.

4. The final step is to return your home computer and input the command **vncviewer** and the address found in step 3 as an argument. Enter your network password and viola! A vnc session is born.

   ```
   radcliff@hegemone / > vncviewer almap7.jb.man.ac.uk:10
   ```

   If you find that the geometry is wrong and does not fit your desktop, you will need to kill the vnc session and restart it from step 2. To kill a session, ssh into the system which the vnc session is on and use the command **vncserver -kill :no.of_vnc_session**. For example:

   ```
    radcliff@ALMAP7   > vncserver -kill :10
   ```

That is all for UNIX, and you've earnt a break. Go to exercise 4 for your reward!!

EXERCISE 1

Try the following commands and see what they do!

- echo hello world
- passwd
- date
- hostname
- arch
- uname -a
- dmesg | more(you may need to press q to quit)
- uptime
- who am i
- who
- id
- last
- finger
- w
- top (you may need to press q to quit)
- echo $SHELL
- echo con,present,fers,ed
- man "automatic door"
- man ls (you may need to press q to quit)
- man who (you may need to press q to quit)
- who can tell me why i got divorced
- lost
- clear
- cal 2000
- cal 9 1752(do you notice anything unusual?)
- bc -l(type quit or press Ctrl-d to quit)
- echo 5+4 | bc -l
- yes please(you may need to press Ctrl-c to quit)
- time sleep 5
- history

EXERCISE 2

1. Try the following command sequence:
   - cd
   - pwd
   - ls -al
   - cd .
   - pwd (where did that get you?)
   - cd ..
   - pwd
   - ls -al
   - cd ..
   - pwd
   - ls -al

- cd ..
- pwd (what happens now)
- cd /etc
- ls -al | more
- cat passwd
- cd -
- pwd

2. Continue to explore the filesystem tree using cd, ls, pwd and cat. Look in /bin, /usr/bin, /sbin, /tmp and /boot. What do you see?
3. Explore /dev. Can you identify what devices are available? Which are character-oriented and which are block-oriented? Can you identify your tty (terminal) device (typing who am i might help); who is the owner of your tty (use ls -l)?
4. Explore /proc. Display the contents of the files interrupts, devices, cpuinfo, meminfo and uptime using cat. Can you see why we say /proc is a pseudo-filesystem which allows access to kernel data structures?
5. Change to the home directory of another user directly, using cd username.
6. Change back into your home directory.
7. Make subdirectories called work and play.
8. Delete the subdirectory called work.
9. Copy the file /etc/passwd into your home directory.
10. Move it into the subdirectory play.
11. Change into subdirectory play and create a symbolic link called terminal that points to your tty device. What happens if you try to make a hard link to the tty device?
12. What is the difference between listing the contents of directory play with ls -l and ls -L?
13. Create a file called hello.txt that contains the words "hello world". Can you use "cp" using "terminal" as the source file to achieve the same effect?
14. Copy hello.txt to terminal. What happens?
15. Imagine you were working on a system and someone accidentally deleted the ls command (/bin/ls). How could you get a list of the files in the current directory? Try it.
16. How would you create and then delete a file called "$ SHELL"? Try it.
17. How would you create and then delete a file that begins with the symbol # ? Try it.
18. How would you create and then delete a file that begins with the symbol -? Try it.
19. What is the output of the command: echo con,present,fers,ed? Now, from your home directory, copy /etc/passwd and /etc/group into your home directory in one command given that you can only type /etc once.
20. Still in your home directory, copy the entire directory play to a directory called work, preserving the symbolic link.
21. Delete the work directory and its contents with one command. Accept no complaints or queries.

22. Change into a directory that does not belong to you and try to delete all the files (avoid /proc or /dev, just in case!)
23. Experiment with the options on the ls command. What do the d, i, R and F options do?

Exercise 3

1. Describe three different ways of setting the permissions on a file or directory to r–r–r–. Create a file and see if this works.
2. Team up with a partner. Copy /bin/sh to your home directory. Type "chmod +s sh". Check the permissions on sh in the directory listing. Now ask your partner to change into your home directory and run the program ./sh. Ask them to run the id command. What's happened? Your partner can type exit to return to their shell.
3. What would happen if the system administrator created a sh file in this way? Why is it sometimes necessary for a system administrator to use this feature using programs other than sh?
4. Delete sh from your home directory (or at least to do a chmod -s sh).
5. Modify the permissions on your home directory to make it completely private. Check that your partner can't access your directory. Now put the permissions back to how they were.
6. Type umask 000 and then create a file called world.txt containing the words "hello world". Look at the permissions on the file. What's happened? Now type umask 022 and create a file called world2.txt. When might this feature be useful?
7. Create a file called "hello.txt" in your home directory using the command cat -u > hello.txt. Ask your partner to change into your home directory and run tail -f hello.txt. Now type several lines into hello.txt. What appears on your partner's screen?
8. Use find to display the names of all files in the /home subdirectory tree. Can you do this without displaying errors for files you can't read?
9. Use find to display the names of all files in the system that are bigger than 1MB.
10. Use find and file to display all files in the /home subdirectory tree, as well as a guess at what sort of a file they are. Do this in two different ways.
11. Use grep to isolate the line in /etc/passwd that contains your login details.
12. Use find and grep and sort to display a sorted list of all files in the /home subdirectory tree that contain the word hello somewhere inside them.
13. Use locate to find all filenames that contain the word emacs. Can you combine this with grep to avoid displaying all filenames containing the word lib?
14. Create a file containing some lines that you think would match the regular expression: (ˆ[0-9]1,5[a-zA-Z]+$)|none and some lines that you think would not match. Use egrep to see if your intuition is correct.
15. Archive the contents of your home directory (including any subdirectories) using tar and cpio. Compress the tar archive with compress, and the cpio archive with gzip. Now extract their contents.
16. On Linux systems, the file /dev/urandom is a constantly generated random stream of characters. Can you use this file with od to printout a random decimal number?
17. Type mount (with no parameters) and try to interpret the output.

9 Exercise 4

1. Archive the contents of your home directory using tar. Compress the tar file with gzip. Now uncompress and unarchive the .tar.gz file using cat, tar and gzip on one command line.
2. Use find to compile a list of all directories in the system, redirecting the output so that the list of directories ends up in a file called directories.txt and the list of error messages ends up in a file called errors.txt.
3. Try the command sleep 5. What does this command do? Run the command in the background using &.
4. Run sleep 15 in the foreground, suspend it with Ctrl-z and then put it into the background with bg. Type jobs. Type ps. Bring the job back into the foreground with fg.
5. Run sleep 15 in the background using &, and then use kill to terminate the process by its job number. Repeat, except this time kill the process by specifying its PID.
6. Run sleep 15 in the background using &, and then use kill to suspend the process. Use bg to continue running the process.
7. Startup a number of sleep 60 processes in the background, and terminate them all at the same time using the pkill command.
8. Use ps, w and top to show all processes that are executing.
9. Use ps -aeH to display the process hierarchy. Look for the init process. See if you can identify important system daemons. Can you also identify your shell and its subprocesses?
10. Combine ps -fae with grep to show all processes that you are executing, with the exception of the ps -fae and grep commands.
11. Start a sleep 300 process running in the background. Log off the server, and log back in again. List all the processes that you are running. What happened to your sleep process? Now repeat, except this time start by running nohup sleep 300.
12. Multiple jobs can be issued from the same command line using the operators ;, && and ||. Try combining the commands cat nonexistent and echo hello using each of these operators. Reverse the order of the commands and try again. What are the rules about when the commands will be executed?
13. What does the xargs command do? Can you combine it with find and grep to find yet another way of searching all files in the /home subdirectory tree for the word hello?
14. What does the cut command do? Can you use it together with w to produce a list of login names and CPU times corresponding to each active process? Can you now (all on the same command line) use sort and head or tail to find the user whose process is using the most CPU?

** Exercises used from http://www.doc.ic.ac.uk/ wjk/UnixIntro/ **

Exercise 5

1) Snake Game
At the Terminal prompt type in emacs, then press enter. You'll see a bunch of text come up. Once it does hold down the ESC key and press X. If you timed your presses right, you'll notice the cursor has moved to the bottom of the page next to the letters M-x. Now type in snake and enjoy!

2) Tetris
Follow the same steps as for the Snake game. At the Terminal prompt type in emacs, then press enter. You'll see a bunch of text come up. Once it does hold down the ESC key and press X. If you timed your presses right, you'll notice the cursor has moved to the bottom of the page next to the letters M-x. This time, type in tetris. Use the arrow keys to move and rotate the blocks, and press the space to make the blocks fall.

10 TO ADD

JBCA tea coffee, stationary, pub, snack club, archiv emails,
Aliases