Anglia Ruskin University

MOD002712
Computer Systems and Servers

Lab Guide – Student Version

Trimester 1, 2017 – 2018

MSc Information and Communication Technology (Conversion)
Anglia Ruskin University, Cambridge Campus

Page Intentionally Blank

Contents

Labs with a * are assessed (compulsory) labs which must be submitted for assessment in the form of a logbook. Please make sure you **read the assessment information section on page 3**.

Introduction

This lab guide contains the lab exercises for the logbook assessment component of the *Computer Systems and Servers* module. The purpose of these labs is to give you "hands-on" experience with Linux, allowing you to practice fundamental techniques.

You are recommended to read through the week's lab **BEFORE** coming to the lab session, and indeed this is what is expected. You will be much better prepared and therefore will make more efficient use of the lab time. Please do ask for help in the lab sessions if you are stuck.

Ian Oxford (ian.oxford@anglia.ac.uk) is the member of staff leading the practical sessions, to whom all queries about the labs should be addressed.

Assessment information

It is in your best interests to read this section carefully, along with the information in the module guide and any other additional information circulated in the lectures, labs, VLE and email. You are expected to attend the lectures and labs, and to check the VLE and your Anglia email account regularly.

You are required to submit for assessment a logbook containing your write up for the lab chapters detailed in Table 0.1 in the box below. Submitting just these should be sufficient (if done correctly and well) to get a pass or merit grade.

Higher marks (aiming towards a distinction) may be obtained by also attempting and completing the optional lab exercises and providing additional insights (which include, but are not limited to, relevant comments and reflections on the exercises and the lab as a whole).

Your logbook mark constitutes 50% of the total module mark; the remaining 50% is from an examination, which will test your knowledge and understanding of the theory and concepts covered in the lectures. In order to pass this module, you must achieve at least 40% in the module overall and obtain the qualifying mark (30%) in the logbook component **and** obtain the qualifying mark (30%) in the examination.

Logbook assessment summary

Table 0.1: Assessed labs and mark weightings for the logbook

Lab	Topic	Marks available
3	* Linux systems and shell scripting	10
4	* Daemons and processes	10
6	* Email under Linux	10
7	* Apache HTTP server and PHP	10
9	* Network traffic analysis	10
10	* Further UNIX tools	10
Logbook presentation		10
Additional insights and value-added work		30
LOGBOOK TOTAL		**100**

Submission deadline: **Monday 18th December 2017 at the iCentre**

Table 0.2: Logbook marking criteria

Grade Approximate mark range	**Criteria**
Fail 0 – 39%	Very few exercises completed correctly. No or very little effort and understanding. Presentation is very poor. The layout is messy and there are frequent and significant spelling and grammar issues, which make reading and understanding the logbook difficult.
D 40 – 49%	Few exercises completed correctly. Little effort and/or understanding. Presentation is poor. There are several noticeable formatting issues and/or spelling and grammar mistakes, which affect the overall coherence of the logbook.
C 50 – 59%	Some exercises completed correctly. Adequate effort and understanding. Presentation is acceptable. There are some noticeable formatting issues and/or spelling and grammar errors.
B 60 – 69%	Most exercises attempted and completed correctly. Good effort and understanding. Presentation is good. There are some formatting issues and/or spelling and grammar mistakes, but these do not detract from the overall coherence of the logbook.
A 70 – 79%	All exercises attempted and completed correctly. Very good effort and understanding. Presentation is excellent, with little or no formatting issues or spelling and grammar errors.
A+ 80 – 100%	All exercises attempted and completed correctly. Excellent effort and understanding. Presentation is professional, as if it were a publishable document.

The assessment details and mark weightings are described in Table 0.1 above. Table 0.2 details the marking criteria.

How to format your logbook

The purpose of the logbook is to keep a record of the tasks carried out during the weekly lab sessions. For **assessed labs**, a good pass mark will be awarded if all of the compulsory lab exercises are well presented, completed correctly and in sufficient detail. To achieve high marks, completion of the optional exercises and the inclusion of additional insights and further relevant work above and beyond the requirements of the exercises are expected. **Your logbook should be sufficiently detailed such that someone else could repeat what you have done. You will therefore need to include evidence in your logbook, such as clear screenshots and the relevant code and commands executed** (and fully annotated to show your understanding). **Your logbook must be concise and neat. Logbooks which do not meet these criteria will lose marks.** You will also see that some exercises in the later labs are more open-ended and this will allow you to discuss why you have chosen to complete the exercise in a certain way. Further advice on how to format your logbook will given in the lab sessions.

You are expected to reference appropriately (see http://libweb.anglia.ac.uk/referencing/referencing.htm) and be aware on the rules regarding collusion and plagiarism. If you are unsure about any of these, please seek advice from the Module Tutor.

1 Getting started with Slackware and Linux (Part 1)

1.1 Installing and setting up Slackware

You will be setting up a Slackware 13.37 VMware virtual machine on a Microsoft Windows 10 host machine. Further details on how to do this will be distributed separately in the lab session.

After you have set up your Slackware machine, please go through the Linux Tutorials. Depending on your prior knowledge and experience, they may take several lab sessions to complete. You do not need to complete all of them in this session, and you will be advised which tutorials are necessary for each lab.

Before starting Lab 3 (* Linux systems and shell scripting), you will need to complete the following Linux Tutorials:

- Linux Tutorial: Basic Linux
- Linux Tutorial: vi
- Linux Tutorial: File permissions

Before starting Lab 4 (* Daemons and processes), you will need to have completed Lab 3 (and the above Linux Tutorials). It is also recommended, though not necessary, that you complete the remaining Linux Tutorials:

- Linux Tutorial: Further file handling
- Linux Tutorial: Redirection
- Linux Tutorial: Wildcards, filename conventions, and getting help

From Lab 5 (Samba) onwards, it is expected that you have completed all of the Linux Tutorials.

In order to take screenshots, you will need to press the [Ctrl] and [Alt] keys together to move the focus from the Slackware virtual machine to the Windows host machine. You can then take screenshots using the Print Screen button or the Snipping Tool. If you are using a Mac host machine, you will need to press the [Ctrl] and [Cmd] keys together and take screenshots by pressing [Cmd] + [Shift] + [4], and then either dragging out the area or pressing [Space Bar] and clicking the window to be screenshot. Please note that although you are free to complete the lab exercises on your own machine rather than the lab machines, such use is unsupported – if you encounter problems, the Module Tutor will be unable to help you. (A common problem with using Mac host machines is the use of the # key. Slackware by default thinks you are using a Windows keyboard – you will therefore need to use [Shift] + [3] if Slackware thinks you are using a US keyboard, or the [\ ] key for a UK keyboard. The normal key combination to type a hash in Mac OS X, [Alt] + [3], will NOT work.)

To power off your Slackware machine, issue the command `init 0` as root.

1.2 Linux Tutorial: Basic Linux

If you are familiar with UNIX or Linux you may know how to do some or all of the following tasks. If you are new to UNIX and Linux, these exercises will get you up and running with most of the tools you will need to successfully complete the lab exercises for this course. Some further tools will be introduced as we need them in later labs. Read through the entire section before you start working, so that you know what to expect.

Log in to the Slackware machine as root.

1.2.1 Listing files and directories: ls

When you first log in, your current working directory is your home directory. Your home directory has the same name as your username, for example, **ee91ab**, and it is where your personal files and subdirectories are saved.

To find out what is in your home directory, type

```
ls
```

then press [Return] (or "Enter"). The ls command (short for list) lists the contents of your current working directory.

There may be no files visible in your home directory, in which case, the UNIX prompt will be returned. Alternatively, there may already be some files inserted by the System Administrator when your account was created.

ls does not, in fact, cause all the files in your home directory to be listed, but only those ones whose name does not begin with a dot (.). Files beginning with a dot (.) are known as hidden files and usually contain important program configuration information. They are hidden because you should not change them unless you are very familiar with UNIX!

To list all files in your home directory including those whose names begin with a dot, execute

```
ls -a
```

ls is an example of a command which can take options: -a is an example of an option. The options change the behaviour of the command. There are online manual pages that tell you which options a particular command can take, and how each option modifies the behaviour of the command. You can view the man (manual) pages by executing

man *command*

for example, man ls will show the manual page for the command ls. Use the [Up] and [Down] keys to navigate and press [q] to exit.

1.2.2 Making directories: mkdir

We will now make a subdirectory in your home directory to hold the files you will be creating and using in the course of this tutorial. To make a subdirectory called **unixstuff** in your current working directory execute

```
mkdir unixstuff
```

To see the directory you have just created, execute

```
ls
```

1.2.3 Changing to a different directory: cd

The command cd directory means change the current working directory to 'directory'. The current working directory may be thought of as the directory you are in, i.e. your current position in the file-system tree. To change to the directory you have just made, execute

```
cd unixstuff
```

Execute ls to see the contents (which should be empty).

Exercise 1.1
Make another directory inside the **unixstuff** directory called **backups**.

1.2.4 The directories . and ..

Whilst you are still in the **unixstuff** directory, execute

```
ls -a
```

As you can see, in the **unixstuff** directory (and in all other directories), there are two special directories called . and ..

In UNIX, . means the current directory, so executing

```
cd . [Note that there is a space between cd and .]
```

means stay where you are (the **unixstuff** directory). This may not seem very useful at first, but using . as the name of the current directory will save a lot of typing, as we shall see later in the tutorial.

.. means the parent of the current directory, so executing

```
cd ..
```

will take you one directory up the hierarchy (back to your home directory). Try it now.

Note: executing cd with no argument always returns you to your home directory. This is very useful if you are lost in the file system.

1.2.5 Pathnames

Pathnames enable you to work out where you are in relation to the whole file system. For example, to find out the absolute pathname of your home directory, execute

```
cd
```

to get back to your home directory. To print the current working directory, execute

```
pwd
```

The full pathname will look something like this:

```
/home/abc123
```

which means that **abc123** (your home directory) is in the directory **/home** – the system's home directory, under which almost everyone's home directories are located. (The home directory for the root account is sometimes located at **/root** rather than **/home/root** in some Linux distributions.)

Exercise 1.2

Use the commands ls, pwd and cd to explore the file system. (Remember, if you get lost, execute cd by itself to return to your home directory.)

Hint: To view commands that you have recently executed, press the [Up] arrow key, which will cycle through your command history starting with the most recent command executed. You can use this to repeat a command or to slightly edit one you have recently used.

1.2.6 More about directories and pathnames

1.2.6.1 Understanding pathnames

First execute cd to get back to your home directory, then

```
ls unixstuff
```

to list the contents of your **unixstuff** directory. Now execute

```
ls backups
```

You will get a message like this:

```
/bin/ls:  cannot access backups:  No  such  file or directory
```

The reason that this error occurs is that **backups** is not in your current working directory. To use a command on a file (or directory) not in the current working directory (the directory you are currently in), you must either cd to the correct directory, or specify its pathname. To list the contents of your backups directory, you must execute

```
ls unixstuff/backups
```

1.2.6.2 The home directory: ~

Home directories can also be referred to by the tilde (~) character. It can be used to specify paths starting at your home directory. So typing

```
ls ~/unixstuff
```

will list the contents of your **unixstuff** directory, no matter where you currently are in the file system.

Exercise 1.3

Write down (on paper) what you think `ls ~` and `ls ~/..` would list. Execute those commands to check if you were right.

1.2.6.3 Relative and absolute pathnames

The top-level directory, known as the root directory, can be accessed by executing

```
cd /
```

All pathnames relative to / are known as absolute pathnames (e.g. **/path/to/this/file**). Pathnames that are relative to the current working directory are known as relative pathnames (e.g. if the current directory is **/path/to**, the file can be accessed using the path **./this/file**).

Exercise 1.4

Change directory to your home directory. What is the absolute pathname of the **unixstuff** directory? What is the pathname of the **unixstuff** directory relative to your homedirectory?

1.2.7 Summary

`ls`	list files and directories in the current directory
`ls -a`	list all files and directories in the current directory
mkdir	make a directory
cd *directory*	change to named directory
cd	change to home directory
cd ~	change to home directory
cd ..	change to parent directory
pwd	display the path of the current directory

1.3 Linux Tutorial: vi

There are many ways to edit files in UNIX/Linux: you may wish to choose another way you are comfortable with. For this tutorial, we are going to edit our files using the text editor vi.

There are two modes while working with the vi editor:

1. **Command mode**: This mode enables you to perform administrative tasks such as saving files, executing commands, moving the cursor, cutting (yanking) and pasting lines or words, and finding and replacing. In this mode, whatever you type is interpreted as acommand.

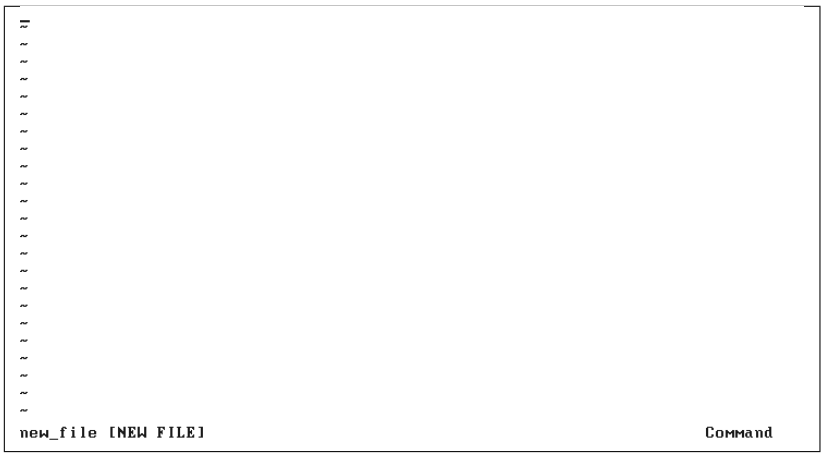
2. **Insert mode**: This mode enables you to insert text into the file. Everything that is typed in this mode is interpreted as input and finally it is put in the file.

1.3.1 Creating a new file in vi

Navigate to your **unixstuff** directory and execute

```
vi new_file
```

You should see a screen something like as follows:



You will notice a tilde (~) on each line following the cursor. A tilde represents an unused line. If a line does not begin with a tilde and appears to be blank, there is a space, tab, newline, or some other non-viewable character present.

So now you have opened one file to start with.

vi always starts in command mode. To enter text, you must be in insert mode. To switch to insert mode, press the **[i]** key. To get out of insert mode, press the [Esc] key, which will put you back into command mode.

Hint: If you are not sure which mode you are in, press the [Esc] key twice, and then you'll be in command mode.

1.3.2 Saving files and exiting vi

The command to quit out of vi is

```
:q
```

i.e. when in command mode, press the **[:]** key, and then [q], followed by the [Return] key.

If your file has been modified in any way, the editor will warn you of this, and not let you quit. To ignore this message, the command to quit out of vi without saving is

```
:q!
```

The command to save the content of a file is

```
:w
```

You can combine the above command with the quit command, or

```
:wq
```

An alternative command to :wq is

```
ZZ [Note that you must use a capital Z and that you don't have to press [Return] afterwards.]
```

You can specify a different file name to save to by specifying the name after the :w. For example, if you wanted to save the file you were working on as another filename called **filename2**, you would type

```
:w  filename2
```

and then press [Return].

Note: If you normally use Microsoft Windows, you are probably used to seeing files having file extensions (e.g. txt, log, exe) and therefore the filename should be **new_file.txt**. There is no requirement in UNIX/Linux based operating systems for files to have file extensions. Unlike Windows, the file extension does not determine the file type in Linux – the content of the file itself (more specifically, its MIME type) determines its type. To determine the file type of a file, use:

```
file filename
```

File extensions can still be useful in UNIX/Linux – for example ls *.png will output the filenames of all files with the file extension **png** (an image file extension) in the current directory. However, a file with the extension **png** may not actually be an image – there is nothing stopping you creating a text file with the extension **png** (or any other file extension).

Exercise 1.5

While **new_file** is still opened, type some meaningful text (at least 25 lines, note that a very long line still counts as one line so you need at least 24 carriage returns), then save and exit the file. You should include the following paragraph from a BBC News website article:

"A controversial theory that the way we smell involves a quantum physics effect has received a boost, following experiments with human subjects. It challenges the notion that our sense of smell depends only on the shapes of molecules we sniff in the air."

Reopen **new_file** and verify if your text is still there.

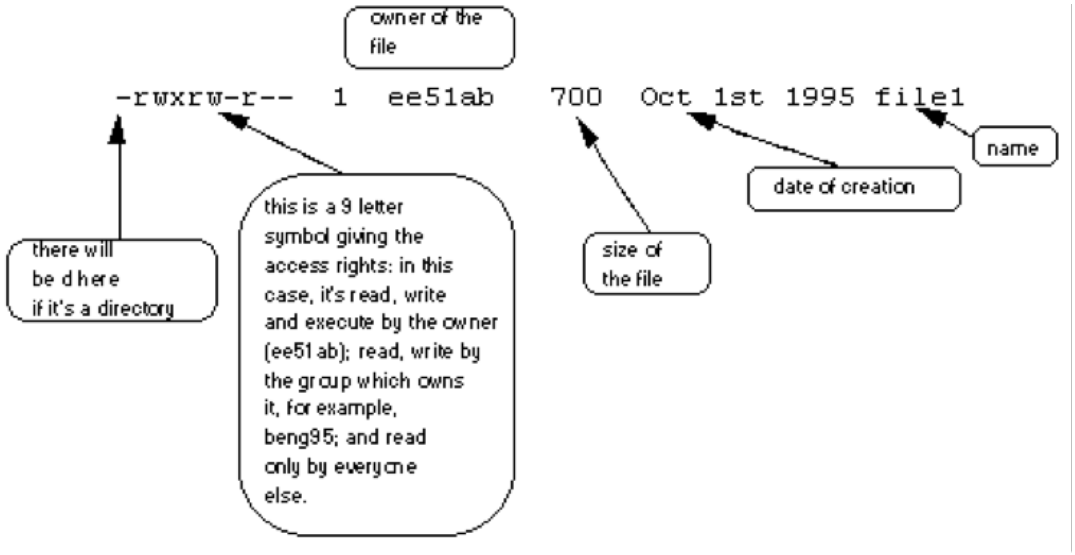
1.4 Linux Tutorial: File permissions

1.4.1 File system security: access rights

In your **unixstuff** directory, execute

```
ls -l
```

The -l option stands for "long listing". You will see that you now get lots of details about the contents of your directory, similar to the example below.



Each file (and directory) has associated access rights, which may be found by executing `ls -l`. You can also see which group owns the file (**beng95** in the following example):

```
-rwxrw-r-- 1 ee51ab beng95 2450 Sept 29 11:52 file1
```

In the left-hand column is a 10 symbol string consisting of the symbols d, r, w, x, -, and, occasionally, s or S. If d is present, it will be at the left hand end of the string, and indicates a directory: otherwise - will be the starting symbol of the string.

The 9 remaining symbols indicate the permissions, or access rights, and are taken as three groups of 3.

- The left group of 3 gives the file permissions for the user that owns the file or directory (**ee51ab** in the above example).
- The middle group gives the permissions for the group of people to whom the file or directory belongs (**beng95** in the above example).
- The rightmost group gives the permissions for all others.

The symbols r, w, etc., have slightly different meanings depending on whether they refer to a simple file or to a directory.

Access rights on files

- r (or -), indicates read permission (or otherwise), that is, the presence or absence of permission to read and copy the file.
- w (or -), indicates write permission (or otherwise), that is, the permission (or otherwise) to change a file.
- x (or -), indicates execution permission (or otherwise), that is, the permission to execute a file, where appropriate.

Access rights on directories

- r allows users to list files in the directory.
- w means that users may delete files from the directory or move files into it.
- x means the right to access files in the directory. This implies that you may read files in the directory provided you have read permission on the individual files.

So, in order to read a file, you must have executable permissions on the directory containing that file, and hence on any directory containing that directory as a subdirectory, and so on, up the tree.

Examples

-rwxrwxrwx	a file that everyone can read, write and execute (and delete).
-rw-------	a file that only the owner can read and write: no-one else can read or write and no-one has execution rights (e.g. your mailbox file).

1.4.2 Changing file mode: chmod

Only the owner of a file can use chmod (change file mode) to change the permissions of a file. The options of chmod are as follows:

u	user
g	group
o	other
a	all
r	read
w	write (and delete)
x	execute
+	add permission
–	remove permission

For example, to remove read, write and execute permissions on the file **biglist** for the group and others, execute

```
chmod go-rwx biglist
```

This will leave the other permissions unaffected. To give read and write permissions on the file **biglist** to all, the command is

```
chmod a+rw biglist
```

Exercise 1.6

Try changing access permissions on the file **new_file** that you created earlier, and on the directory **backups**. Use `ls -l` to check that the permissions have changed.

2 Getting started with Slackware and Linux (Part 2)

2.1 Linux Tutorial: Further file handling

2.1.1 Copying files: cp

The command which makes a copy of **file1** in the current working directory and calls it **file2** is

```
cp file1 file2
```

What we are going to do now is to take a file stored in an open access area of the file system, and use the cp command to copy it to your **unixstuff** directory. First, cd to your **unixstuff** directory.

```
cd ~/unixstuff
```

Then at the UNIX prompt, execute

```
cp new_file /var/tmp/new_file
```

The above command means copy **new_file** from the current directory ('**.**', i.e. **~/unixstuff/**) to the directory **/var/tmp/**, keeping the filename the same. (Note: The **tmp** directory is an area of the filesystem to which everyone has read and write access.)

Exercise 2.1
Create a backup of your **new_file** file by creating a copy of it with the filename **new_file.bak**

2.1.2 Moving files: mv

To move a file from one place to another, the mv command is used. This has the effect of moving rather than copying the file, so you end up with only one file rather than two. It can also be used to rename a file, by moving the file to the same directory, but giving it a different name. The syntax is the same as the cp command:

```
mv file1 file2
```

moves or renames file1 to file2.

We are now going to move the file **new_file.bak** to your **backup** directory. First, change to your **unixstuff** directory (can you remember how?). Then, inside the **unixstuff** directory, execute

```
mv new_file.bak backups/
```

Type ls and ls backups to see if it has worked. Note the use of / in the above command – this can be useful to distinguish between a file and a directory of the same name (which should be avoided!).

2.1.3 Removing files and directories: rm

To delete (remove) a file, the rm command is used. As an example, we are going to create a copy of the **new_file** file, then delete it.

Inside your **unixstuff** directory, execute

```
cp new_file tempfile
```

Use ls to check if it has created the file, then execute

```
rm tempfile
```

Use ls to check if it has deleted the file.

Removing directories can be done using the rmdir command. This command will fail if the directory is not empty. To test this, try removing the **backups** directory. Non-empty directories can be removed by using rm with the option -r. Be careful about doing this!

Exercise 2.2

Create a directory called **tempstuff**, then remove it.

2.1.4 Displaying file contents: cat, less, head and tail

Before you continue further, you may like to clear the terminal window of the previous commands so the output of the following commands can be clearly understood. At the prompt, execute

```
clear
```

This will clear all text and leave you with the prompt at the top of the window.

The command cat (concatenate) can be used to display the contents of a file on the screen. Change to the **unixstuff** directory and execute

```
cat new_file
```

If the file is longer than than the size of the window, only the bottom part of the contents can be seen. The whole file can be scrolled through by using the command less. Execute

```
less new_file
```

Press the [Up] and [Down] keys to scroll through the file, type [q] if you want to quit reading. As you can see, less is used in preference to cat for long files. The more command is similar to the less command, however, you can only scroll forwards with more using the [Space Bar] and less also allows you to search (see later).

The head command writes the first ten lines of a file to the screen.

First clear the screen then execute

```
head new_file
```

The tail command writes the last ten lines of a file to the screen. Clear the screen and execute

```
tail new_file
```

Exercise 2.3

Execute head -5 new_file. What difference did the -5 do to the head command? What command should you use to view the last 15 lines of the file?

2.1.5 Searching file contents: less, grep and wc

Using less, you can search though a text file for a keyword (pattern). For example, to search through new_file for the word 'the', execute

```
less new_file
```

then, still in less (i.e. don't press [q] to quit), type a forward slash [/] followed by the word you wish to search for:

```
/the
```

less finds and highlights the keyword if it is present, otherwise it will give an error message. Press [n] to search for the next occurrence of the word.

grep (**g**lobally search a **r**egular **e**xpression and **p**rint) is one of many standard UNIX utilities. It searches files for specified words or patterns. First clear the screen, then execute

```
grep the new_file
```

As you can see, grep has printed out each line containing the word 'the'. Or has it? Try executing

```
grep The new_file
```

The grep command is case sensitive; it distinguishes between 'The' and 'the'. To ignore upper/lower case distinctions, use the **-i** option, i.e. execute

```
grep -i the new_file
```

To search for a phrase or pattern, you must enclose it in single quotes (the apostrophe symbol). For example:

```
grep -i 'shapes of molecules' new_file
```

Some of the other options of grep are:

- -v display those lines that do NOT match
- -n precede each maching line with the line number
- -c print only the total count of matched lines

Try some of them and see the different results. You can use more than one option at a time, for example, the number of lines without the words 'the' or 'The' is

```
grep -ivc the new_file
```

A handy little utility is the wc command, short for word count. To do a word count on **new_file**, execute

```
wc -w new_file
```

To find out how many lines the file has, execute

```
wc -l new_file
```

2.1.6 Summary

cp *file1 file2*	make a copy of file1, called file2
mv *file1 file2*	move or rename file1 to file2
rm *file*	remove file
rmdir *directory*	remove (empty) directory
cat *file*	display a file
less *file*	display a file page by page
head *file*	display the first few lines of a file
tail *file*	display the last few lines of a file
grep *keyword*	search a file for keyword
wc *file*	count the number of words/lines/characters in a file

2.2 Linux Tutorial: Redirection

Most processes initiated by UNIX commands write to the standard output (that is, they write to the terminal screen), and many take their input from the standard input (that is, they read it from the keyboard). There is also the standard error, where processes write their error messages, by default, to the terminal screen.

We have already seen one use of the cat command to write the contents of a file to the screen. Now execute

```
cat
```

without specifing a file to read. Then type a few words on the keyboard then hold the [Ctrl] key down and press [d] (written as ^D for short) to end the input. What has happened?

If you run the cat command without specifing a file to read, it reads the standard input (the keyboard), and on receiving the "end of file" parameter (^D), copies it to the standard output (the screen). In UNIX, we can redirect both the input and the output of commands. To continue within cat, press the [Return] key. To exit cat and return to the prompt, press ^D twice (or once if the current line is a blank line).

2.2.1 Redirecting the output

We use the > symbol to redirect the output of a command. For example, to create a file called **list1** containing a list of fruit, execute

```
cat > list1
```

Then type in the names of some fruit. Press [Return] after each one and then ^D ([Ctrl] and [d]) when the list is finished.

```
pear
banana
apple
^D
```

What happens is the cat command reads the standard input (the keyboard) and the > redirects the output, which normally goes to the screen, into a file called **list1**. To read the contents of the file, execute

```
cat list1
```

Exercise 2.4

Using the above method, create another file called **list2** containing the following fruit: orange, plum, mango, grapefruit. Read the contents of **list2**.

To append items to an existing file, use >>. So to add more items to the file **list1**, execute

```
cat >> list1
```

Then type in the names of more fruit:

```
peach
grape
orange
^D
```

Read the contents of **list1** and verify that it now contains six fruit. We can use the cat command to join (concatenate) **list1** and **list2** into a new file called **biglist**. Execute

```
cat list1 list2 > biglist
```

What this is doing is reading the contents of **list1** and **list2** in turn, then outputing the text to the file **biglist**. Read the contents of the new file.

2.2.2 Redirecting the input

We use the < symbol to redirect the input of a command. The command sort alphabetically or numerically sorts a list. Execute

```
sort
```

then type in the names of some vegetables. Press [Return] after each one.

```
carrot
beetroot
artichoke
^D
```

The output will be:

```
artichoke
beetroot
carrot
```

Using < you can redirect the input to come from a file rather than the keyboard. For example, to sort the list of fruit, execute

```
sort < biglist
```

and the sorted list will be output to the screen. To output the sorted list to a file, execute

```
sort < biglist > slist
```

Use cat to read the contents of the file **slist**.

2.2.3 Pipes

To see who is on the system with you, execute

```
who
```

One method to get a sorted list of names is to execute

```
who > names.txt
sort < names.txt
```

This is a bit slow and you have to remember to remove the temporary file when you have finished. What would be better is connect the output of the who command directly to the input of the sort command. This is exactly what pipes do. The symbol for a pipe is the vertical bar (|). For example, executing

```
who | sort
```

will give the same but quicker and cleaner result as above. To find out how many users are logged on, execute

```
who | wc -l
```

2.2.4 Summary

command > *file*	redirect standard output to a file
command>> *file*	append standard output to a file
command < *file*	redirect standard input from a file
command1 | *command2*	pipe the output of command1 to the input of command2
cat *file1 file2* > *file0*	concatenate file1 and file2 to file0
sort	sort data
who	list users currently logged in

2.3 Linux Tutorial: Wildcards, filename conventions, and getting help

2.3.1 Wildcards: * and ?

The character * is called a wildcard, and will match against none or more character(s) in a file (or directory) name. For example, in your **unixstuff** directory, execute

```
ls list*
```

This will list all files in the current directory starting with **list**. Try executing instead

```
ls *list
```

This will list all files in the current directory ending with **list**.

The character ? will match exactly one character. So ls ?ouse will match files like house and mouse, but not grouse. Try executing

```
ls ?list
```

2.3.2 Filename conventions

We should note here that a directory is merely a special type of file. So the rules and conventions for naming files apply also to directories.

In naming files, characters with special meanings such as /, *, &, % should be avoided, as well as spaces. The safest way to name a file is to use only alphanumeric characters, that is, letters and numbers, together with _ (underscore) and . (dot).

File names conventionally start with a lower-case letter, and may end with a dot followed by a group of letters indicating the contents of the file, known as the file extension. For example, all files consisting of C code maybe named with the ending .c, for example, **prog1.c**. Then in order to list all files containing C code in your home directory, you need only type ls *.c in that directory.

Beware: Some applications give the same name to all the output files they generate. For example, some compilers, unless given the appropriate option, produce compiled files named **a.out**. Should you forget to use that option, you are advised to rename the compiled file immediately, otherwise the next such file will overwrite it and itwill be lost.

2.3.3 Getting help

The man command was introduced in the first Linux Tutorial. To recap:

man wc

gives the manual pages of the command wc. Alternatively,

whatis wc

gives a one line description of the command, omitting any information about options etc. The command

wc --help

gives a brief synopsis of the command and a summary of the options and their uses.

If you are not sure of the exact name of a command,

apropos *keyword*

will give you the commands with the keyword in their manual page header.

Exercise 2.5

Try executing apropos copy. You will notice that there are so many results that they go off the screen. How can you scroll through the results? Save the results into a text file called **copy_functions.txt**. How many different functions are there (**Hint:** this is equal to the number of lines in the file)?

2.3.4 Summary

*	match any number of characters
?	match one character
man *command*	read the online manual page for a command
whatis *command*	brief description of a command
apropos *keyword*	match commands with keyword in their man pages

3 * Linux systems and shell scripting

This is an assessed lab. You must write up and submit your answers to this lab for assessment as part of your logbook.

Before you begin this lab, you will need to have completed the following Linux Tutorials: Basic Linux; vi and File permissions.

Learning objectives

The aims of this lab are to:

- Practice the Linux skills and concepts covered in the first few Linux Tutorials
- Introduce shell scripting and user management on Linux

By the end of this lab session, you should be able to:

- Navigate the UNIX file system
- Create and execute UNIX shell scripts
- Create and manage new UNIX users and groups
- Modify file and directory permissions and ownerships

Log in to the Slackware machine as root.

Shell scripting

Exercise 3.1 Creating an executable bash script

1. Using vi (or another terminal text editor such as emacs), create a script called **test.sh** in your home directory with the following contents:

```
#!/bin/bash
clear
echo "Hello World"
```

   Make sure the script is executable, then run it by executing: ./test.sh

2. A student, when completing the above question, executed the following command:

   chmod 777 test.sh (equivalent to executing chmod a+rwx test.sh)

   Explain why giving 777 permissions to a file is a bad idea.

System administration

New users may be created using the following commands:

```
useradd -d /home/bob bob  # creates user bob and sets home directory to /home/bob
mkdir /home/bob           # creates the directory bob in /home
chown bob /home/bob       # changes the owner of /home/bob from root to bob
chgrp bob /home/bob       # changes the group of /home/bob from root to bob
```

Note: chown bob:bob /home/bob can be used to change the owner and group to bob without having to use a separate chgrp command.

Passwords to user accounts can be changed by using: passwd *username*. If *username* is not given, the password for the currently logged in user is changed.

Exercise 3.2 Creating new users

Create two new user accounts:

- username: bob, password: bob
- username: smith, password: smith

Exercise 3.3 Creating a shared executable script

1. Create a publically readable and writeable directory with the path **/home/ncs** with the appropriate directory permissions.
2. Create a bash script in this directory called **hello.sh** which should print a message saying "Hello World".
3. Execute this script.
4. Note down the owner/group ownerships and the file permissions of this script.

Exercise 3.4 Accessing files from different user accounts

a) Press [Ctrl] + [Alt] + [F2] and log in as bob.

1. Navigate to the directory **/home/ncs**. What do you see in this directory?
2. Execute ./hello.sh. Does it succeed? Why (not)?
3. Create a script called **bob.sh**, which should print the message "Hello this is Bob" when executed.
4. Execute ./bob.sh and explain the result you get.

b) Press [Ctrl] + [Alt] + [F3] and log in as smith.

1. Navigate to the directory **/home/ncs**. What do you see in this directory?
2. Execute ./hello.sh and ./bob.sh. Explain the results you get.

Exercise 3.5 Optional exercises

These exercises require you to read up on additional material.

1. Create a group called **sysadmins**, and add bob and smith as members. Change the group owner of **/home/ncs**, **/home/ncs/hello.sh** and **/home/ncs/bob.sh** to this group. Can both bob and smith execute both scripts now?
2. **Disable** smith's user account. **Do not delete smith's account or files.**

If you finish this lab early, you should use the remaining time to complete the Linux Tutorials (if you haven't already done so).

4 * Daemons and processes

This is an assessed lab. You must write up and submit your answers to this lab for assessment as part of your logbook.

Before you begin this lab, you will need to have completed the * Linux systems and shell scripting lab **and have created the bob user account on the Slackware machine.** It is recommended that you also complete the remaining Linux Tutorials before beginning this lab: Further file handling; Redirection and Wildcards, filename conventions and getting help.

Learning objectives

The aims of this lab are to:

- Introduce the concept of processes, daemons and UNIX signals
- Explore displaying and manipulating running processes, and to gain experience in configuring daemon behaviour (in particular inetd)

By the end of this lab session, you should be able to:

- State the differences between daemons and processes
- Describe the purpose of the inetd super-server daemon
- Identify the directory where daemons and processes are located
- Start, stop and restart daemons and processes
- Modify daemon behaviour through editing configuration files
- Find the process ID number for a particular daemon or process
- List all active daemons and processes on the system

Log in to the Slackware machine as root. Execute finger bob to check that the bob user account exists in the system.

The internet service daemon: inetd

inetd (internet service daemon) is also known as the "super-server daemon" on many UNIX/Linux systems. It manages Internet services, runs at boot time and listens for connections on a number of ports. When a connection comes in, inetd decides which service should be started and starts the application required to handle the request. Having passed the request to another application, it can then becomes idle, waiting for the next request.

For example, when an ftp request comes in, it is received by inetd which starts the ftp daemon. The ftp daemon then takes over and handles the ftp session.

In UNIX/Linux, many Internet-related services are handled by inetd. The idea is that instead of starting every daemon that the system could possibly need, inetd listens for certain requests and starts services as appropriate, thus making better use of the system's valuable resources.

Note: Not all Internet services are started by inetd, for example nfsd is run by the system at boot time (if there is anything to export).

Like other daemons, inetd reads its configuration file (usually located at **/etc/inetd.conf**) as it starts up and "remembers" this configuration from then on. Therefore, if you edit the configuration file, you must restart the inetd daemon in order to ensure that the new changes will take effect.

Because there is only one instance of inetd running for a given UNIX/Linux system, we can simply restart the daemon with a HUP (hang up) signal:

```
kill -HUP process_pid
```

where *process_pid* is the PID (process identifier) number of the daemon to be restarted. For example, the command

```
ps ax | grep inetd
```

will return a line containing the process details for inetd:

```
1686   ?      Ss     0:00   /usr/sbin/inetd
```

so the command

```
kill -HUP 1686
```

would restart inetd. There is also a command called pidof that returns the PID for a process, for example

```
pidof inetd
```

which, in the above example, would return 1686.

Exploring processes and jobs

A process is an executable program identified by a unique PID. To see information about your processes, with their associated PID and status, execute

```
ps
```

A process may be in the foreground, in the background, or be suspended. In general the shell does not return the UNIX prompt until the current process has finished executing.

Some processes take a long time to run and hold up the terminal. "Background"ing a long process has the effect that the UNIX prompt is returned immediately, and other tasks can be carried out while the original process continues executing.

Listing all processes

To list all the processes currently running, execute

```
ps aux
```

A command which lists the most CPU-intensive processes is

```
top
```

which is dynamically updated (press [q] to quit).

Exercise 4.1 Exploring currently running processes

Using ps aux and top (and any other commands you feel would be useful), find out the total number of processes that are currently running. Identify the 10 most CPU-intensive processes and give a brief (one sentence) description of what each of them do. **Hint:** you can use ps aux | less to scroll through the list and man *command* to find out more about a command.

Listing suspended and background processes

When a process is running, suspended or in the background, it will be entered onto a list along with a job number. To examine this list, execute

```
jobs
```

An example of a job list could be:

1. Suspended sleep 100
2. Running netscape
3. Running nedit

To restart (or foreground) a suspended processes, execute

```
fg %job_number
```

For example, to restart sleep 100 in the above example, execute

```
fg %1
```

Executing fg with no job number foregrounds the last suspended process. To resume a suspended process in the background, the bg command is used instead.

Looking up processes

It is useful to be able to look up a particular process/daemon. Sometimes a port scan will do the job.

nmap is a network discovery utility useful for managing network services. It uses raw IP packets to determine the network hosts and the services (application name and version) they offer as well as the type of operating systems (and OS versions) that they are running.

Exercise 4.2 Exploring network processes

Execute the command nmap localhost. Write down all the processes returned and explain their purpose.

To look for a particular process, execute

```
ps aux | grep process_name
```

Starting processes

Processes are listed in **/etc/rc.d/** directory. To start a process/daemon simply lookup the name of the process/daemon and execute

```
/etc/rc.d/process_name start
```

Stopping a currently running process

A currently running process can be terminated using the command kill, which sends a signal to the selected process. There are several different types of signal: signal 9 (SIGKILL) will terminate the process immediately whilst signal 1 (SIGHUP) will wait until all other dependent child processes are terminated.

To stop a running process one must need to know the process ID. Once you know the process ID, then

kill -9 *process_id* or kill -SIGKILL *process_id*

will kill the process immediately.

Exercise 4.3 Exploring UNIX signals

Execute the command kill -l. Explain what this command does and use a table to summarise the results (signal number, signal name, short description). You only need to consider signal numbers 1 – 9, and five more of your choice between signal numbers 10 and 31. Please remember to cite any sources that you use to answer this exercise using a recognised academic referencing system (see http://libweb.anglia.ac.uk/referencing/referencing.htm).

Exercise 4.4 Processes and networking

You created the bob user account in Exercise 3.2 on page 22, check that it exists before completing this exercise. When completing this exercise, please remember to provide screenshots in your logbook to demonstrate your results.

1. Edit **/etc/inetd.conf** to enable ftp and telnet. Restart inetd and execute ftp localhost and telnet localhost and log in as bob to see if they are enabled. Make sure you know how to upload, download, delete and rename files. **Hint:** use the vsftpd daemon. Its configuration file is located at: **/etc/vsftpd.conf**.
2. Edit **/etc/inetd.conf** again to disable ftp. Restart inetd and test out your changes.
3. What is sftp and ssh? Why is the use of telnet discouraged in the "real world"?

Exercise 4.5 Optional exercises

1. How could you combine pidof with kill -HUP in order to restart inetd in **one** command? **Hint:** try using quotation marks (such as: 1, ", `).
2. Can you ftp as root? Is it a good idea to be able to ftp as root and why? If so, in what context?
3. From your Windows host machine, ftp and telnet to your Slackware virtual machine.

5 Samba

Before you begin this lab, you will need to have completed the * Daemons and processes lab **and have created the bob user account on the virtual machine** (from the * Linux systems and shell scripting lab).

Learning objectives

The aims of this lab are to:

- Introduce the SMB (Server Message Block) and CIFS (Common Internet Filesystem) protocols
- Explore the use of Samba to share files between Linux and Windows operating systems

By the end of this lab session, you should be able to:

- Start, stop and restart the Samba daemon
- Create Samba users
- Create a Samba share on a Linux system that can be accessed from a Windows system
- (Optional) Create a Windows share that can be mounted on a Linux system

Samba is a free suite of programs that implement the Microsoft SMB (Server Message Block) protocol to enable resources to be shared between Microsoft Windows and UNIX operating systems.

Introduction

Samba is a free suite of programs that implement the Microsoft SMB (Server Message Block) protocol to enable resources to be shared between Microsoft Windows and UNIX operating systems.

In this lab, we will focus on the creating a Linux share that can be accessed from Windows machines. If time permits, we will try the reverse (creating a Windows share that can be accessed from Linux machines). Samba can also be used to share printers between Linux and Windows, however, we will not cover this in this lab.

Samba runs using the daemons smbd and nmbd. The smbd daemon provides the file and print services to SMB clients, such as Windows XP, Windows 7 etc. The nmbd daemon provides NetBIOS name serving and browsing support. The configuration file for both daemons is located at **/etc/samba/smb.conf** in Slackware.

Log in to the Slackware machine as root. Ensure that the Windows host machine can communicate with the Slackware virtual machine: use the ipconfig (Windows) and ifconfig (Linux) commands to find the IP address of the machines and the ping command to test connectivity.

Adding Samba users

Samba uses a different user system to the standard Linux system.

A Samba user can be created by executing

smbpasswd -a *username*

The username specified must be an existing Linux user with a non-empty password.

Exercise 5.1 Creating a Samba user

Create a Samba user with username **bob** and password **bob**.

Setting up a Samba share

On your Windows host machine, right click Computer then click Properties to bring up the System properties window. Note down the "workgroup" setting.

On the Slackware virtual machine, create an active configuration file:

```
cp /etc/samba/smb.conf-sample /etc/samba/smb.conf
```

Open the configuration file in a text editor:

```
vi /etc/samba/smb.conf
```

Make the following changes:

- Change the **workgroup** setting to match the workgroup of the Windows host machine.
- Change the **browseable** attribute under **[homes]** to yes.

Save your changes and exit your text editor. To check that there are no syntax errors, execute

```
testparm
```

If there are any errors identifed, you will need to re-edit the configuration file.

For the purpose of managing concurrent access, Samba requires a lock directory. To create a lock directory, execute

```
mkdir -p /usr/local/samba/var/locks
```

Start the Samba daemons (smbd and nmbd) and verify that they are running.

Exercise 5.2 Samba and port scanning

In Lab 4 Exercise 4.2 on page 25, the concept of a port scan using nmap was introduced. Find out which ports are being used by Samba.

Exercise 5.3 Accessing a Linux share using Windows

Find out the IP address of the Linux virtual machine by executing ifconfig.

On the Windows host machine, open Windows Explorer and navigate to

```
\\virtual_machine_IP_address
```

If everything is working properly, you should get a login prompt. Enter the appropriate username and password (refer back to Exercise 5.1 on the previous page). Create a file called **samba-test.txt** in the default directory.

On the Linux virtual machine, verify that the file **samba-test.txt** has been created.

Exercise 5.4 Accessing a Windows share from Linux (Optional exercise)

On the Windows host machine, navigate to the C:\ drive and create a folder called **samba-on-windows**. Right click this folder, then click Properties. Enable sharing and editing of this folder.

On the Linux virtual machine, execute:

```
mount -t cifs //win_IP /samba-on-windows ~/winmount -o username=win_username
```

replacing *win_IP* with the IP address of the Windows host machine and *win_username* with the username used to log in to the Windows host machine. You will be prompted to enter your Windows password (to access the Windows share).

If successful, you can access **samba-on-windows** on the virtual machine by changing directory to **~/winmount**. Verify that you have editing permissions for **samba-on-windows** (try to create a file in that directory).

To unmount the share, change directory so that you are not in **~/winmount** and then execute

```
umount ~/winmount
```

Troubleshooting

It is not uncommon to find that only one of the Samba daemons starts. If this is the case, check your system messages and log files for clues to help solve the problem. Log files are saved in **/var/log/**. Use the `tail` command to view the last few lines of a file (refer back to the Linux Tutorials). You will also see a **samba** subdirectory in **/var/log/** – be prepared to check the files in there too.

6 * Email under Linux

This is an assessed lab. You must write up and submit your answers to this lab for assessment as part of your logbook.

Before you begin this lab, you will need to have completed the * Daemons and processes lab and all of the Linux Tutorials.

Learning objectives

The aims of this lab are to:

- Consolidate the Linux skills and concepts covered in the last few Linux Tutorials and labs (in particular piping and redirection, user management and daemon configuration)
- Gain an understanding of how email works at the network level through exploring SMTP and POP3 sessions directly
- (Optional) Explore IMAP and/or PGP through independent research

By the end of this lab session, you should be able to:

- Identify the email daemon, and start, stop and restart it
- Compose and read email on the system as different users, and send email to different user accounts
- Locate the directory where emails are stored
- Interact with SMTP and POP3 servers

Introduction

In this session we will take a look at some simple applications for reading and dealing with email, the files and directories that they use and the way email is stored on the system before it is read.

Log in to the Slackware machine as root. **Before you continue check that:**

- The Windows host machine can communicate with the Slackware virtual machine, and vice-versa.
- The sendmail process is running (daemon located at: **/etc/rc.d/rc.sendmail** – you may need to chmod it first so that it is executable. Refer to Linux Tutorial: File permissions on page 11 and Lab 4 on page 23 (* Daemons and processes) if you are stuck).
- You have the hostname of the virtual machine.
- The user bob exists on the Slackware system.

Sending email

The main options of the mail command include:

- -s *subject* – The email subject
- -c *email_address* – Carbon copy (Cc) an email address
- -b *email_address* – Blind carbon copy (Bcc) an email address

Execute

```
mail -s testing-mail bob@localhost
```

Then type, pressing [Return] after each line:

```
hello
this is a test
.
```

The final line (containing only a dot) signals the end of the message. You may be prompted for any recipients you wish to carbon copy (Cc), if so press [Return] to skip this step.

The above sends a message to user bob on the local machine, with subject "testing-mail" and a message body containing two lines.

Alternatively, assuming the computer name is anglia and the domain is bryant, one could also execute:

```
echo "Email body." | mail -s "Subject" bob@anglia.bryant
```

If the message to be sent is contained within a text file, it can be redirected into the mail command using the syntax

```
mail -s subject-name user@domain < message.txt
```

Exercise 6.1 Sending email using mail

1. Create a file called **message.txt** with some text, then redirect it to mail using the syntax above to send it to bob.
2. Explain what the following command does:

   ```
   echo "Welcome to Network Computer Systems" | mail -s "Hello world"
   bob@anglia.bryant -c smith@anglia.bryant -b root@anglia.bryant
   ```

A major drawback of mail is that it does not support attachments. Two popular text-based email programs that do are mutt and alpine. Both of these programs are included with Slackware by default, and you may wish to explore them. mutt in particular is a very powerful email client – if you can manage its steep learning curve, you will be able to organise your email very quickly (similar to how using the command line can be much quicker than using a graphical interface).

Reading email

You may have noticed the message "You have mail" when you log in to the Slackware machine. To read email, simply execute

```
mail
```

If you have any unread email, you will be presented with a summary of the unread messages.

Exercise 6.2 Checking email

Log in as bob (you can execute su bob, then exit when you are finished) and check that all emails sent to bob are present. If they are not, check that sendmail is running. (If sendmail is not running, the emails are saved in **/var/spool/mqueue/** and will be sent once you start the daemon.)

Exercise 6.3 Exploring mail

You should complete this exercise as bob, not root.

1. Describe, using appropriate screenshots, how to do the following in mail: read, reply, send, delete, list and save messages. **Hint**: to display help in mail, type [?].
2. What is contained in **/var/spool/mail/**? What are the security implications of this?
3. From your Windows host machine, telnet to your virtual machine on port 25 (telnet *ip_address* 25) and send a message to bob@localhost from *your_name*@localhost by talking to the SMTP server. The message body text and subject can be anything you like. **Hint:** follow the example SMTP session on page 33.
4. Enable POP3. **Hint:** look in **/etc/inetd.conf**. Also, you should remember that you need to do something after saving changes to a configuration file to make those changes take effect... (refer to Lab 4 on page 23)
5. From your Windows host machine, telnet to your virtual machine on port 110. Explore talking to the POP3 server. **Hint:** follow the example POP3 session on page 33.
6. What ports are used by SMTP and POP3?

Exercise 6.4 Optional exercises

1. What is IMAP? List the differences between POP3 and IMAP.
2. What is PGP encryption? How does it work?

Example SMTP and POP3 sessions

You should refer to the below when completing Exercise 6.3 questions 3 and 5.

Type and execute lines beginning with "S:" (don't type the "S:" itself), then wait for the server's reply. Do not execute lines beginning with "R:" as this is the reply from the server.

SMTP session (S = sender's input; R = mail server's reply)

```
S: helo wonderland
R: 250 wonderland.com Hello ely.esf [192.168.200.2], pleased to meet you
S: MAIL FROM:<bob@anglia.bryant>
R: 250 OK
S: RCPT TO:<bob@localhost>
R: 250 OK
S: DATA
R: 354 Start mail input; end with <CRLF>.<CRLF>
S: From: "Bob" <bob@anglia.bryant>
S: Subject: Have you seen my white rabbit?
S: To: bob@localhost
S: Content-Type: text
S:
S: I'm most concerned.
S: I fear he may have fallen down a hole.
S: .
R: 250 OK
```

POP3 session (S = sender's input; R = mail server's reply)

```
S: <client connects to service port 110>
R: +OK POP3 server ready <1896.697170952@mailgate.dobbs.org>
S: USER bob
R: +OK bob
S: PASS replace_with_bob's_password
R: +OK bob's maildrop has 2 messages (320 octets)
S: STAT
R: +OK 2 320
S: LIST
R: +OK 2 messages (320 octets)
R: 1 120
R: 2 200
R: .
S: RETR 1
R: +OK 120 octets
R: <the POP3 server sends message 1>
R: .
S: DELE 1
R: +OK message 1 deleted
S: RETR 2
R: +OK 200 octets
R: <the POP3 server sends message 2>
R: .
S: DELE 2
R: +OK message 2 deleted
S: QUIT
R: +OK dewey POP3 server signing off (maildrop empty)
S: <client hangs up>
```

7 * Apache HTTP server and PHP

This is an assessed lab. You must write up and submit your answers to this lab for assessment as part of your logbook.

Learning objectives

The aims of this lab are to:

- Gain experience in setting up and configuring an Apache server capable of serving web pages
- Introduce the PHP scripting language and the **hosts** file

By the end of this lab session, you should be able to:

- Locate the Apache daemon, and start, stop and restart it
- Change the behaviour of the Apache daemon
- Create and host HTML and PHP files
- Interact with the web server using a terminal interface
- State the purpose of the **hosts** file, and locate and edit it

Introduction

Apache is the most widely used web server software in the world. It is open source and cross-platform, so you are free to download and install it on your own machine (unlike Microsoft IIS, which is proprietary and limited to machines running Microsoft Windows only). This lab will introduce you to running a basic Apache server. You are advised to follow this lab carefully and ask questions if you do not understand something. Most importantly, *think about what you are doing*!

Log in to the Slackware machine as root.

Installing and configuring Apache

Apache is preinstalled in Slackware 13.37 by default, so there is no need to compile and install it from source. We simply need to configure it and then run it.

The main Apache configuration file is usually located at **/etc/httpd/httpd.conf**.

Open **/etc/httpd/httpd.conf** using a text editor (such as vi) and browse through it. If you are using vi, you can search by using the following syntax in command mode:

```
?search_string
```

If there are several matches, pressing N (i.e. [Shift] + [n]) will give the next match whilst [n] (lower case n) will give the previous match.

Exercise 7.1 Configuring Apache

Whilst **/etc/httpd/httpd.conf** is still open, answer the following questions.

1. Lines that start with a # are comments: what are the purpose of these?
2. Find the default values of **ServerName** and **DocumentRoot** and note them down. (Uncomment the line containing **ServerName**.) What do they do/mean?
3. Uncomment the line Include /etc/httpd/mod_php.conf. What does this do?

Make sure you save the changes made before continuing.

Running Apache

The following commands will start, stop and restart httpd, the Apache HTTP daemon, respectively (you may need to chmod **/etc/rc.d/rc.httpd** first so that it is executable: this also makes httpd run on startup):

```
/etc/rc.d/rc.httpd start
/etc/rc.d/rc.httpd stop
/etc/rc.d/rc.httpd restart
```

If you make any configuration changes to Apache whilst httpd is running (including **httpd.conf**), you must restart httpd for those changes to take effect. (Refer to Lab 4 on page 23 if you are stuck.)

You can check if httpd is running by executing the following command

```
ps aux | grep httpd
```

Exercise 7.2 Running Apache

1. Why do you need to restart httpd if you make changes to the configuration?
2. This question is about the command ps aux | grep httpd.
   a) What does the command ps aux do? What about the command grep httpd? What does "|" mean? Hence, explain what the command ps aux | grep httpd does.
   b) What would you expect to see as the output of the command ps aux | grep httpd if httpd is running? How about if it is not running? Try both cases and note down the results.
3. By executing ps axl | egrep "httpd|PPID" (make sure you copy this command exactly: note the spacing and capital letters), find the process ID of the **parent** httpd process (the PPID).

Creating and viewing HTML files

Earlier in the lab (Exercise 7.1 on the preceding page), you noted down the default value of **DocumentRoot** from the Apache configuration file, which is a directory. Navigate to this directory. A file named **index.html** should already exist in this directory with the following HTML code:

```
<html><body><h1>It works!</h1></body></html>
```

Exercise 7.3 Creating HTML files

1. What is special about **index.htm** and **index.html** files?
2. Find out the permissions and file and group ownership of **index.html**.
3. Create a new file called **test.html** with the following source code:

```
<html>
<head><title>NSE Apache Lab</title></head>
<body>
<h1>This is the NSE Apache Lab test page</h1>
<p>Under construction!</p>
</body>
</html>
```

   Draw out, on paper, what you expect to see if you opened **test.html** in a web browser.

We would normally use a web browser to view HTML files (such as Mozilla Firefox or Google Chrome), however, since we are running Slackware using a CLI rather than a GUI, we will use a command-line web browser called lynx.

Exercise 7.4 Viewing HTML files using a terminal interface

1. Explain the difference between CLI and GUI.
2. What is special about the IP address 127.0.0.1?
3. View the HTML file in lynx by executing

   ```
   lynx 127.0.0.1/test.html
   ```

   If the HTML file does not load, check to see if httpd is running and that the HTML file is saved in the correct location and has appropriate file permissions and file owner. Press the [=] button to see more information about the web page (including the page title and URL). Take a screenshot of this screen. To exit lynx, press [q] and then [y].
4. (Optional) Find out the IP address of the virtual machine, then view the file using your Windows host machine by opening a web browser (such as Mozilla Firefox or Google Chrome) and navigating to http://*replace_with_ip_address* /test.html.

Creating and viewing PHP files

In the same directory as **index.html**, create a new file called **nse.php** with the following code:

```
<?php
phpinfo(); //test if PHP is working
?>
```

Exercise 7.5 Creating and viewing PHP files using a terminal interface

1. What does phpinfo(); do?
2. Load the file in lynx by executing

```
lynx 127.0.0.1/nse.php
```

Take a screenshot of the result. The page should be titled "phpinfo()", the phrases "PHP Logo" and "PHP Version 5.x.x" should appear at the top of the page and you should see a lot of information about the Apache configuration. If you just see the source code of **nse.php**, open the Apache configuration file (**httpd.conf)**, check that PHP is enabled, then restart httpd.

The hosts file

The **hosts** file maps a hostname to an IP address on the local machine. Earlier in the lab (Exercise 7.1), you noted down the default value of **ServerName** from the Apache configuration file. We will host **ServerName** on our local machine by adding an entry to the **hosts** file.

Exercise 7.6 Exploring and adding an entry to the **hosts** file

Using a text editor, open the file **/etc/hosts**. You should see the following:

```
# For loopbacking 127.0.0.1 localhost
# This next entry is technically wrong, but good enough to get TCP/IP apps
# to quit complaining that they can't verify the hostname on a loopback-only
# Linux box
127.0.0.1 darkstar.example.net darkstar

# End of hosts.
```

Add the following line to the **hosts** file on the line just above `# End of hosts.`:

```
127.0.0.1 replace with ServerName
```

Note: In Exercise 7.1, you noted down the default value of **ServerName** – use this without the port number e.g. www.example.com

The hosts file should now look like this:

```
# For loopbacking 127.0.0.1 localhost
# This next entry is technically wrong, but good enough to get TCP/IP apps
# to quit complaining that they can't verify the hostname on a loopback-only
# Linux box
127.0.0.1 darkstar.example.net darkstar
127.0.0.1 www.example.com
# End of hosts.
```

Save the file and exit the text editor. Now execute

```
lynx replace with ServerName
```

Does it work? Press the [=] button and take a screenshot.

You have now completed the main part of the lab. If time permits, you are recommended to go through Exercise 7.7 on the next page.

Exercise 7.7 Optional exercises

1. Create a new file called **index.php** in the **DocumentRoot** directory (same directory as the **test.html** and **nse.php** files you edited earlier in the lab), with the following code:

```
<?php
echo "<head><title>PHP test</title></head>\n";
echo "Current PHP version: " . phpversion();
?>
```

   You now have both an **index.html** file and an **index.php** file in the same directory. Which file will load when you execute the command lynx 127.0.0.1? How can you change this? (**Hint:** Look at the **DirectoryIndex** settings in **httpd.conf**, and remember to restart httpd after making any changes.)

2. Apache provides authentication and authorization capabilities, which can be used to restrict access to an area of a website. Whenever a user tries to view a protected directory or a file, they must log in as a user first (authentication). If that user is allowed to access the file (authorization), they are granted permission, otherwise the request is denied (HTTP error 401). The following steps will enable the **DocumentRoot** directory to be password-protected using username **user** and password **password**.

   a) First, we need to create a password file. Execute:

```
htpasswd -c /var/www/.htpasswd user
```

   When prompted, enter **password** as the password. This will create a file called **.htpasswd** in **/var/www/**. Open and view the file – you should see an entry for user **user** with an encrypted password.

   b) Open **httpd.conf**, find the line: <Directory "/srv/httpd/htdocs"> and add the following green text just above the </Directory> tag (**[…]** indicates other parts of the file which you should not change):

```
[...]
<Directory "/srv/httpd/htdocs">
        [...]
        Order allow,deny
        Allow from all

        AuthType  Basic
        AuthName "Authorisation required"
        AuthUserFile /var/www/.htpasswd
        Require   valid-user
        Order allow, deny
        Allow  from all
</Directory>
[...]
```

   Save the file and restart httpd.

   c) Execute the command lynx 127.0.0.1 (if you have successfully password-protected the **DocumentRoot** directory, it will display Alert! Access without authorization denied -- retrying before you can enter a username and password combination).

8 MySQL (optional – for advanced students)

This lab is optional and can be skipped if you wish – you should only attempt it if you are already experienced with databases and MySQL (or other SQL platforms) or if you are happy to teach it to yourself. If you have already done the Developing Web Applications module you will be able to do this chapter, but if you have not you may want to skip over it for now and come back and attempt it later after you have done the module.

Learning objectives

The aims of this lab are to:

- Practice using MySQL to create simple databases and tables
- (Optional) Link PHP and MySQL together by creating a dynamic web page

By the end of this lab session, you should be able to:

- Locate the MySQL daemon, and start, stop and restart it
- Change the behaviour of the MySQL daemon
- Write and execute basic SQL commands to interact with the MySQL server
- Create and view databases and tables using SQL commands
- Insert data into tables using SQL queries

Introduction

MySQL is the one of the most widely used database management software in the world. It is open source and cross-platform, so you are free to download and install it on your own machine. This lab will introduce you to the basics of running a MySQL server. You are advised to follow this lab carefully and ask questions if you do not understand something. Most importantly, *think about what you are doing*!

Log in to the Slackware machine as root.

Installing and configuring MySQL

MySQL is preinstalled in Slackware 13.37 by default, so there is no need to compile and install it from source. It does, however, require some more initial configuration compared with Apache before it can be run for the first time.

MySQL comes with some default configuration templates. The active configuration file is located at **/etc/my.cnf**. The template that we will be using in this lab is **my-small.cnf** (there are also **my-medium.cnf**, **my-large.cnf** and **my-huge.cnf**, and you may wish to investigate the differences between them). As root, execute the following command

```
cp /etc/my-small.cnf /etc/my.cnf
```

to copy the template to the active configuration file. Use the following command

```
mysql_install_db
```

to set up the MySQL database for first use. Finally, file and group ownership are fixed by executing:

```
chown -R mysql:mysql /var/lib/mysql
```

Running MySQL

The following commands will start, stop and restart mysqld, the MySQL daemon, respectively (you may need to chmod **/etc/rc.d/rc.mysqld** first so that it is executable):

```
/etc/rc.d/rc.mysqld start
/etc/rc.d/rc.mysqld  stop
/etc/rc.d/rc.mysqld  restart
```

If you make any configuration changes to MySQL whilst mysqld is running (including **my.cnf**), you must restart mysqld for those changes to take effect. (Refer to Lab 4 on page 23 if you are stuck.)

You can check if mysqld is running by executing the following command

```
ps -ef | grep mysqld        or        mysqladmin ping
```

Once mysqld is running, we can switch to the MySQL prompt. This is done by executing

```
mysql
```

You should now see the MySQL prompt:

```
mysql >
```

We can now execute SQL commands. All SQL commands must end with a semi-colon (;). To view databases, execute:

```
SHOW DATABASES;
```

and you should see the following output

```
mysql > SHOW DATABASES;

+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
+--------------------+
```

We can create a database using the command

```
CREATE DATABASE database_name;
```

and delete a database using the command

```
DROP DATABASE database_name;
```

WARNING: DON'T DELETE THE mysql DATABASE! If you do this, you won't be able to log in to MySQL. This can be fixed by executing mysql_install_db. **Only delete databases that you have created yourself.**

We can select a database for use using the command

```
USE DATABASE database_name ;
```

Once a database is selected, we can view the tables contained therein using the command

```
SHOW TABLES;
```

Hint: MySQL will not parse a query until a semi-colon is typed in. You can therefore break up a long MySQL query into several lines.

Exercise 8.1

1. By executing SELECT User, Host FROM mysql.user; find out the number of entries in the **user** table in the **mysql** database.
2. What are the commands to create and delete a table, and to select data from a table? What about inserting data into a table? **Hint:** use the MySQL manual (http://dev.mysql.com/doc/refman/5.1/en/tutorial.html)
3. Create a database called **nselab**. Within this database, create a table called **users**, with three fields: **studentid**, **firstname** and **lastname**. Make **studentid** the primary key field. This field cannot be empty for any record and should automatically increase by one each time a new record is added (the first record should have a studentid of 1, the second 2 etc.) The other two fields should be limited to a maximum of 25 characters each. Add two students to this table: Joe Bloggs and Ashley Smith. Devise a plan to test the validation rules (this will involve adding more records) and carry it out.

Exercise 8.2 Optional exercises

1. Create a MySQL user called **adminsec** and grant it full privileges to the **nselab** database only (and no privileges, including viewing, to any other database). You will need to look up the syntax (don't try to do this by editing the **mysql** database directly).
2. Create a web page using PHP that can display the data in the **users** table in the **nselab** database. You should try to show in your logbook that you *understand* what the code does – this could be achieved by annotating the code with comments. (If you are new to PHP, you will need to do some independent research. You may find the following tutorial helpful: http://www.lynda.com/MySQL-tutorials/PHP-MySQL-Essential-Training/119003-2.html. You will need to follow the instructions here to log in: http://web.anglia.ac.uk/it/training/lynda/.)

9 * Network traffic analysis

This is an assessed lab. You must write up and submit your answers to this lab for assessment as part of your logbook.

Learning objectives

The aims of this lab are to:

- Use shell scripting to carry out network traffic analysis, combining regular expressions and redirection
- Explore "spoofing" a MAC address, DNS lookup and the routing table

By the end of this lab session, you should be able to:

- Understand UNIX commands involving regular expressions and redirection
- Temporarily spoof the MAC address of a Linux system
- Use shell scripting to find the IP address of all machines alive on the same network

Introduction

Networking is the act of interconnecting machines to form a network so that the machines can interchange information. The most widely used networking stack is TCP/IP, where each node is assigned a unique IP address for identification. There are many parameters in networking, such as subnet mask, route, ports, host names, and so on which require a basic understanding to follow.

Several applications that make use of a network operate by opening and connecting to something called ports, which denote services such as data transfer, remote shell login, and so on. Several interesting management tasks can be performed on a network consisting of many machines. Shell scripts can be used to configure the nodes in a network, test the availability of machines, automate execution of commands at remote hosts, and so on.

Login to the Slackware machine as root.

Printing the list of network interfaces

Here is a one-line command sequence to print the list of network interfaces available on a system:

```
ifconfig | cut -c-10 | tr -d ' ' | tr -s '\n'

eth0
lo
wlan0
```

The first 10 characters of each line in the ifconfig output is reserved for writing names of network interfaces. Hence, we use:

- cut to extract the first 10 characters of each line
- tr -d ' ' to delete every space character in each line
- tr -s '\n' to squeeze the \n newline character, producing a list of interface names

Depending on how the networking is set up on your Slackware virtual machine, you will normally see either **eth0** or **wlan0**, but not both. **eth0** will be used for the rest of this lab, but if you see **wlan0**, change the commands appropriately.

Displaying IP addresses

The ifconfig command displays details of every active network interface available on the system. However, we can restrict it to a specific interface using:

```
ifconfig interface_name
```

For example:

```
ifconfig eth0

eth0
Link encap:Ethernet
HWaddr 00:1c:bf:87:25:d2        [Hardware (MAC) address]
inet addr:192.168.0.82          [IP address]
Bcast:192.168.3.255             [Broadcast address]
Mask:255.255.252.0              [Subnet mask]
```

In several scripting contexts, we may need to extract any of these addresses from the script for further manipulations. Extracting the IP address is a frequently needed task. In order to extract the IP address from the ifconfig output one could use:

```
ifconfig eth0 | egrep -o "inet addr:[^ ]*" | grep -o "[0-9.]*"

192.168.0.82
```

Here, the first command egrep -o "inet addr:[^]*" will print inet addr:192.168.0.82. The pattern starts with inet addr: and ends with some non-space character sequence (specified by [^]*). In the next pipe, it prints the character combination of digits and "**.**".

Spoofing the hardware address (MAC address)

In certain circumstances where authentication or filtering of computers on a network are based on the hardware address, we can use hardware address spoofing. The hardware address appears in the ifconfig output as HWaddr 00:1c:bf:87:25:d2.

We can spoof the hardware address at the software level as follows:

```
ifconfig eth0 hw ether 00:1c:bf:87:25:d5
```

In the preceding command, 00:1c:bf:87:25:d5 is the new MAC address to be assigned. This can be useful when we need to access the Internet through MAC-authenticated service providers that provide access to the Internet for a single machine. However, note that this only lasts until a machine restarts.

DNS lookup

There are different DNS lookup utilities available from the command line, which will request a DNS server for an IP address resolution. host and nslookup are two of such DNS lookup utilities. When host is executed it will list out all of the IP addresses attached to the domain name. nslookup is another command that is similar to host, which can be used to query details related to DNS and resolving of names. For example:

```
host google.com

google.com has address 64.252.191.242
[...]
google.com has address 64.252.191.217
```

```
google.com has IPv6 address 2a00:1450:4009:80d::200e
google.com mail is handled by 30 alt2.aspmx.l.google.com.
google.com mail is handled by 50 alt4.aspmx.l.google.com.
google.com mail is handled by 40 alt3.aspmx.l.google.com.
google.com mail is handled by 10 aspmx.l.google.com.
google.com  mail  is handled  by  20 alt1.aspmx.l.google.com.
```

We can also list out all the DNS resource records as follows:

```
nslookup google.com

Server:      192.168.118.2
Address:   192.168.118.2#53

Non-authoritative answer:
Name:  google.com
Address: 64.252.191.217
[...]
Name:  google.com
Address:    64.252.191.242
```

The last line in the preceding command-line snippet corresponds to the default name server used for resolution.

Without using the DNS server, it is possible to add a symbolic name to the IP address resolution just by adding entries into the file **/etc/hosts**. In order to add an entry, use the following syntax:

```
echo ip_address symbolic_name >> /etc/hosts
```

For example:

```
echo   192.168.0.9   backupserver   >> /etc/hosts
```

After adding this entry, whenever resolution to **backupserver** occurs, it will resolve to 192.168.0.9.

Showing routing table information

Having more than one network connected with each other is a very common scenario. An example of this is in a college, where different departments may be on separate networks. In this case, when a device on one network wants to communicate with a device on the other network, it needs to go through a device which is common to the two networks. This special device is called a **gateway** and its function is to route packets to and from different networks.

The operating system maintains a table called the **routing table**, which contains the information on how packets are to be forwarded through machines on the network. The routing table can be displayed as follows:

```
route

Kernel  IP  routing  table
Destination     Gateway         Genmask         Flags   Metric  Ref    Use  Iface
192.168.0.1     *               255.255.252.0   U       2       0        0wlan0
link-local      *               255.255.0.0     U       1000    0        0wlan0
default         p4.local        0.0.0.0         UG      0       0        0wlan0
```

Or:

```
route -n

Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.0.1     0.0.0.0         255.255.252.0   U     2      0        0wlan0
169.254.0.0     0.0.0.0         255.255.0.0     U     1000   0        0wlan0
0.0.0.0         192.168.0.4     0.0.0.0         UG    0      0        0wlan0
```

The -n option of the route command causes all entries to display numerical IP addresses rather than symbolic hostnames.

A default gateway is set as follows:

```
route add default gw ip_address interface_name
```

For example:

```
route add default gw 192.168.0.1 wlan0
```

Listing all the machines alive on the network

When we deal with a large local area network, we may need to check the availability of other machines in the network. A machine may not be alive due to one of two conditions: either it is not powered on, or due to a problem in the network. By using shell scripting, we can easily find out and report which machines are alive on the network.

Exercise 9.1

Create a bash script called **ping.sh** with the following code, then run it. Press [Ctrl] + [z] to quit the program.

```
#!/bin/bash
# Change base address 192.168.0 according to your network.
for ip in 192.168.0.{1..255};
do
        ping $ip -c 2 &> /dev/null;
        if [ $? -eq 0 ];
        then
                echo $ip is alive
        fi
done
```

Think about how the script works before reading the next section.

We used the ping command to find out the alive machines on the network. We used a for loop for iterating through a list of IP addresses generated using the expression 192.168.0.{1..255}. The {start..end} notation will expand and will generate a list of IP addresses, such as 192.168.0.1, 192.168.0.2, 192.168.0.3 up to 192.168.0.255.

ping $ip -c 2 &> /dev/null will run a ping command to the corresponding IP address in each execution of the loop. The -c option is used to restrict the number of echo packets to be sent to a specified number. &> /dev/null is used to redirect both stderr and stdout to **/dev/null** so that it won't be printed on the terminal. Using $? we evaluate the exit status. If it is successful, the exit status is 0, else it is non-zero. Hence, the IP addresses which replied to our ping are printed.

In this script, each ping command for the IP address is executed one after the other. Even though all the IP addresses are independent of each other, the ping command is executed as a sequential program, it takes a delay of sending two echo packets and receiving them or the time-out for a reply for executing the next ping command.

10 * Further UNIX tools

This is an assessed lab. You must write up and submit your answers to this lab for assessment as part of your logbook.

Learning objectives

The aims of this lab are to:

- Gain experience in using "classic" UNIX user and system management tools
- Explore symlinks and further file management commands
- Introduce file compression and backup utilities built into UNIX

By the end of this lab session, you should be able to:

- Use tools such as finger to find out more information about a UNIX user
- Create symlinks and hard links, and explain the difference between the two
- Compress files into archives, and then extract them

Log in to the Slackware machine as root.

User and system information

users and who show the list of users logged into a machine. who also shows the terminal they are using and the date they logged in on.

```
users
```

```
bob
```

```
who
```

```
bob          tty1         2014-12-05  19:41
```

w gives more detailed information than who.

```
w
```

```
USER     TTY      FROM             LOGIN@  IDLE   JCPU   PCPU  WHAT
bob      tty1                      08:45    1.00s  0.06s  0.00s w
```

finger also gives detailed information about the list of users logged into a machine, but it can also give detailed information about a specific user (whether or not they are currently logged in).

```
finger ashley
```

```
Login name:          ashley
Registered name:     Dr A. Smith          Directory:    /home/ashley
Personal name:       Ashley Smith         Shell:        /bin/bash
Affiliation(s):      Department of Computer Science
Last login Tue Aug 6 18:24 2013 (BST) on /dev/pts/0 from localhost
Mail last read Tue Aug  6 18:36 2013 (BST)
Plan:
To think of a plan.
```

In order to get information about previous boot and user login sessions, use the last command.

```
last
```

```
smith    tty2                          Fri Feb  5 10:31   still logged in
bob      tty1                          Fri Feb  5 08:45   still logged in
reboot   system boot  3.19.0-25-generi Fri Feb  5 08:45 - 10:48  (02:02)
bob      tty1                          Thu Feb  4 20:07 - down   (01:48)
reboot   system boot  3.19.0-25-generi Thu Feb  4 20:07 - 21:55  (01:48)

wtmp begins Thu Feb  4 20:07:31 2016
```

last can also be used to obtain information about login sessions for a single user or just the reboot sessions.

```
last bob
```

```
bob      tty1                          Fri Feb  5 08:45   still logged in
bob      tty1                          Thu Feb  4 20:07 - down   (01:48)

wtmp begins Thu Feb  4 20:07:31 2016
```

```
last  reboot
```

```
reboot   system boot  3.19.0-25-generi Fri Feb  5 08:45 - 10:48  (02:02)
reboot   system boot  3.19.0-25-generi Thu Feb  4 20:07 - 21:55  (01:48)

wtmp begins Thu Feb  4 20:07:31 2016
```

Information about failed user logins (if any) can be displayed by executing lastb.

```
lastb
```

```
bob      tty1                          Fri Feb  5 10:54 - 10:54  (00:00)
UNKNOWN  tty1                          Fri Feb  5 10:53 - 10:53  (00:00)
root     tty1                          Fri Feb  5 10:52 - 10:52  (00:00)

btmp begins Fri Feb  5 10:57:16 2016
```

In order to see how long the system has been powered on, use the uptime command.

```
uptime
```

```
12:33  up 12 days, 20:57, 2 users, load averages: 1.79, 1.84, 1.78
```

Exercise 10.1 User and system information

Before attempting the questions below, you may wish to deliberately reboot the machine and create some failed login attempts so that you have some data to work with.

1. How many login attempts (successful and failed) occurred in the past 48 hours?
2. How many system reboots occurred in the past 48 hours?

Symbolic and hard links

In the * Email under Linux lab, you were introduced to the mail command. In Slackware 13.37, the program is actually called mailx. Some (older) Linux distributions use nail instead.

So how can we tell whether to use mail, mailx or nail? The command

```
whereis program
```

tells you the path to the binaries in the system, so the command

```
whereis mailx
```

will tell you that mailx is in **/usr/bin/mailx** (and also return a number of other answers). If you do the same for mail, you should find that it returns **/bin/mail**. If you execute

```
ls -l /bin/mail
```

you will see that **/bin/mail** is a symlink (symbolic link) to **/usr/bin/mailx**. Therefore it does not matter if we execute mail or mailx: they are exactly the same. You may wish to try the above for nail as well.

Symlinks are similar to shortcuts in Microsoft Windows and aliases in Mac OS X. Symlinks may be created using the syntax below:

```
ln -s /path/to/original/file /path/to/symlink
```

Hard links may be created as above, but without the -s option.

Exercise 10.2 Symbolic and hard links

1. Create a file **~/unixstuff/extra_file** and a symlink **~/unixstuff/links/extra_file_link** which links to **extra_file** (you may need to create the **links** directory). Use `ls -l` whilst in **~/unixstuff/links/** to check that the symlink has been created.
2. Edit **extra_file** and add some text to it. Now open **extra_file_link** by executing the following command:

```
cat ~/unixstuff/links/extra_file_link
```

   Do you see the changes you made?

3. Move **extra_file** to the **backups** directory (so its location is now **~/unixstuff/backups/extra_file**).
   a) What happens to **extra_file_link** (if anything)? **Hint:** try opening the symlink using cat, what is the result? Execute `ls -l` whilst in **~/unixstuff/links/**, do you notice anything different?
   b) Move **extra_file** back to the **unixstuff** directory – predict what happens to **extra_file_link** then test your prediction.
4. Delete **extra_file_link**. What happens to **extra_file** (if anything – try opening it using cat)?
5. Recreate the **extra_file_link** symlink and delete **extra_file**. What happens to **extra_file_link** (if anything)? See the hint to question 3 (a) if you are stuck.
6. Delete **extra_file_link** and redo questions 1 – 5 above, but this time use hard links instead. Hence explain the differences between symbolic and hard links. You might also wish to do some research to explain why you see these differences.

File management

Some useful file management commands are:

- df (disk free)

  The df command outputs a report on the disk space available.

- diff

  diff allows you to compare the content of two text files and outputs the differences. The syntax is as follows:

  diff *file1 file2*

  Lines in **file1** are prefixed with < whilst lines in **file2** are prefixed with >.

- find

  find searches through the file system for files matching specified attributes (such as file name, file contents, size). Execute man find to see what options are available.

- touch

  touch updates the access and modification date and time of a file to the current time. The syntax is as follows:

  touch *file*

  If **file** does not already exist, it is created with zero contents.

File compression and backup

UNIX systems usually support a number of utilities for backing up and compressing files. The most useful are:

- tar (tape archiver)

  tar backs up entire directories and files onto a tape device or (more commonly) into a single disk file known as an archive. An archive is a file that contains other files plus information about them, such as their filename, owner, timestamps, and access permissions. tar does not perform any compression by default.

  To create a disk file tar archive, use

  tar -cvf *archivename filename*

  where *archivename* will usually have a **.tar** extension. Here the -c option means create, -v means verbose (output filenames as they are archived), and -f means file. To list the contents of a tar archive, use

  tar -tvf *archivename*

  To restore files from a tar archive, use

  tar -xvf *archivename*

- cpio

  cpio is another facility for creating and reading archives. Unlike tar, cpio doesn't automatically archive the contents of directories, so it's common to combine cpio with find when creating an archive:

  find . -print -depth | cpio -ov -H tar > *archivename*

  This will take all the files in the current directory and the directories below and place them in an archive called *archivename* . The -depth option controls the order in which the filenames are produced and is recommended to prevent problems with directory permissions when doing a restore. The -o option creates the archive, the -v option prints the names of the files archived as they are added and the -H option specifies

an archive format type (in this case it creates a tar archive). Another common archive type is crc, a portable format with a checksum for error control.

To list the contents of a cpio archive, use

cpio -tv < *archivename*

To restore files, use:

cpio -idv < *archivename*

Here the -d option will create directories as necessary. To force cpio to extract files on top of files of the same name that already exist (and have the same or later modification time), use the -u option.

- compress and gzip

compress and gzip are utilities for compressing and decompressing individual files (which may be or may not be archive files). To compress files, use:

compress *filename* or gzip *filename*

In each case, *filename* will be deleted and replaced by a compressed file called **filename.Z** or **filename.gz**. To reverse the compression process, use:

compress -d *filename* or gzip -d *filename*

zcat can be used to read gzipped text files without uncompressing them first. The output can be piped to less if the text file is very long.

Exercise 10.3 File compression and backup

1. Archive the contents of your home directory (including any subdirectories) using tar and cpio.
2. Compress the tar archive with compress, and the cpio archive with gzip.
3. Now extract their contents.
