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.
