

Intro Lab Session 3

Pi power

Contents

3.1	The LXDE graphical environment	60
3.2	Configure mutt again using scp and ssh	61
3.2.1	Emailing your tutor	64
3.3	X Windows again	64
3.4	A Simple Web Server	65
3.4.1	A slightly more interesting web page	68
3.5	Headless Pi	69
3.5.1	Setting up Apache, a proper web server	70
3.5.2	Permissions, users and groups	71
3.5.3	Back to that webserver	73
3.6	Shutting down gracefully	75
3.7	Finishing up	75

These notes are available online at
studentnet.cs.manchester.ac.uk/ugt/COMP10120/labscripts/intro3.pdf
You may find it useful to use the online version so that you can follow web links.

So far you've used your Pi and the lab PCs separately, and hopefully you've come to understand that they are both essentially the same kind of machine; they both run variations of the same operating system, and the principles you learn using one for the most part apply to the other. In the case of the Pi, you have complete control of the device as a superuser, so can do absolutely what you like to it; in the case of the lab machines you have access to much more powerful hardware, but more restricted access to the filestore and operating system for reasons of security and privacy.

Today, we'll be getting your Pi and a desktop PC to communicate with one another, to reinforce the similarities between these setups, and to expose you to some of the principles of networking and remote access.

3.1 The LXDE graphical environment

Connect your Pi up to the monitor, keyboard, mouse, network and power supply as before, and log in (remembering this time to use whatever password you set on the Pi rather than

Breakout 3.1: LXDE isn't a window manager either



Rather like GNOME, LXDE is technically a collection of desktop tools rather than a window manager as such. LXDE uses the **openbox**[™] window manager, and **GTK+**[™], which was developed as part of the **GIMP**[™] project to draw the buttons, sliders and other graphical controls. LXDE is the default graphical environment for several 'lean' or 'low power' Linux distributions such as **Knoppix**[™], **Lubuntu**[™] and of course the Pi's operating system, **Raspbian**[™].

your University password, and the username 'pi'). At the console, type:

```
$ startx
```

to **start up X Windows** and the Pi's default window manager, **LXDE**, which appear moments later looking like the screenshot in Figure 3.1.

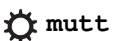
LXDE (the 'Lightweight X11 Desktop Environment') is **designed to be a lean, fast desktop environment, which makes it ideally suited for the Pi's modest CPU**. Although rather less rich in features and 'eyecandy' than GNOME, LXDE is a fully functioning environment that has all the features you will need for operating and programming the Pi.

Up at the top of the screen you will see **controls** that give you **access to various applications, tools and system preferences**. On the **panel's right** are a **CPU meter and a clock**. On the **desktop** itself is a shortcut to the **Wastebasket**, which is similar to the Recycle Bin on Windows and the Trash on OS X.

Spend a few minutes exploring the GUI. You may find that you're double-clicking on things that only need a single click, and vice versa, and may find that things aren't quite where you expect them to be—but rather than dismissing LXDE as being crude, embrace the differences; you may well find that you come to prefer its 'no frills' approach to window management over that of richer, heavier-weight environments such as GNOME. **LXDE** and other slimmed-down graphical environments **consume considerably fewer CPU cycles and hence less power than their richer counterparts**, and while this isn't an issue when you're running on a mains-powered desktop machine with a reasonably beefy graphics card such as the lab PC, this can be a serious issue for devices running off batteries (**and it does mean of course that LXDE is rather more environmentally friendly!**)

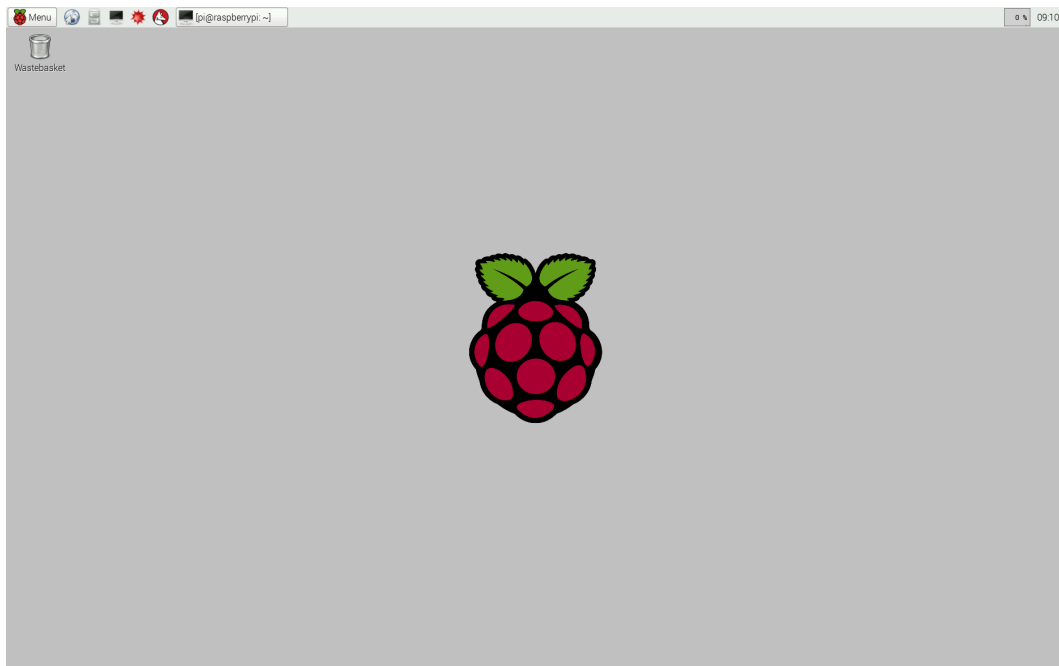
3.2 Configure mutt again using scp and ssh

Once you've found your way around LXDE, fire up a terminal so that you can configure your Pi to read your University email using Mutt. Unlike the lab machines, the `mutt` command isn't installed by default on the Pi, so you'll need to do that yourself using:



```
$ sudo apt-get install mutt
```

Once the install has completed, you'll need to **adjust Mutt's settings so that they once again point at your University email account**. You could do this by following the instructions in the previous session's script again, but there's a much easier way: **let's just copy the configuration file you created on the desktop PC last time, over to the Pi**.

**Figure 3.1**

The Pi's default window manager is called LXDE.

Breakout 3.2: Unix hostnames



A machine's hostname identifies it on a network in a way that's usually a bit more human-readable than its IP address. There are various ways of finding out a machine's hostname:

- If you haven't logged into the PC and can see its screen, then it should be showing the hostname such as `E-C07KILF3101`: before the login prompt.
- If you're already logged in to Unix, but have not started a window manager, your console prompt will be something like `[mrnoodle@E-C07KILF3101 ~]$` (where `mrnoodle` would be replaced by your own username, of course).
- If you are logged in to the PC and inside a window manager, open a terminal window. It should show you the command line prompt, as above. You could also type `echo $HOSTNAME`, or use the `hostname` command.



hostname


First you need to check that you still have the `.muttrc` configuration file in your home filestore of your Computer Science account. You *could* unplug all the cables from the Pi, connect them back into the desktop PC and check that way, but that's no fun (especially because we'd then have to reverse it all to do the next bit of the exercise). But you don't need to do that: you can check remotely.

Find the hostname of the lab PC on your desk: there should be a sticker on both the PC itself and the monitor with a name such as `E-C07KILF3101`. If for some reason there isn't a sticker, Breakout 3.2 describes several other ways to find a Unix machine's hostname.

Once you've got hold of the PC's hostname, you're going to use a Secure Shell[™] to connect to from the Pi to the the desktop, allowing you to type commands on the Pi that will be executed

remotely via a shell that's running on the desktop PC. Open a terminal window on the Pi and issue this command:

```
$ ssh [USERNAME]@[HOSTNAME].cs.man.ac.uk
```

 **ssh**

replacing `[USERNAME]` with your University username, and `[HOSTNAME]` with the name of the PC in front of you. You'll need to enter your University password, and will most likely be presented with text something similar to this:

```
The authenticity of host 'E-C07KILF3101 (130.88.197.112)' can't be
established.
RSA key fingerprint is 20:6d:2d:90:5e:8f:9f:19:39:70:ce:48:a6:93:ec:4c.
Are you sure you want to continue connecting (yes/no)?
```

Type `yes` (and hit enter) in response to the question. You should then be given a command prompt. You'll learn more about what this message actually means later in your programme, but for now just treat it as something that happens the first time you try to connect to a particular machine. Notice that this command prompt no longer says `pi@raspberrypi`, but rather has the name of the machine you've just remotely logged into. If your prompt still includes `pi@raspberrypi`, then something hasn't worked: you probably got the PC hostname wrong, or mistyped your University username or password, so just try again.

Type `ls -a` to confirm that your `.muttrc` file is still where you expect it to be, and if all is well then press `<ctrl>d` to log out of the remote shell you just started, and you will return to the shell running locally on your Pi (you'll see the command prompt change back to being `pi@raspberrypi` again). Now we know that the file we want is there, we need to copy it from your Computer Science filestore onto your Pi's local filestore.

 **ls**

To do this, we're going to use the `scp` (Secure Copy) command, which in some ways behaves like `cp`, but allows us to copy files *between machines*.

 **scp**
 **cp**

Like `cp`, the `scp` command in its basic form takes two arguments, the first is the name of the source file (the one you want to copy), and the second is the name of the destination file (the one you want to create). The difference with `scp` is that either (or less commonly, both) of these files can be on a remote machine, which means that you need to provide the command with enough information about the location of the remote file in terms of the hostname and file system, and any login details necessary to get at it. The syntax for providing this information is:

```
[USERNAME]@[HOSTNAME]:[FILEPATH]
```

So for example, if you wanted to retrieve a file called `cheese.jpg` from the home directory of a user called `mrnoodle` that was stored on a machine with the hostname `mypc.example.com`, and you wanted the local copy of the file to be called `mycheese.jpg` the command would be:

```
$ scp mrnoodle@mypc.example.com:cheese.jpg mycheese.jpg
```

Then, supposing you had edited the file `mycheese.jpg` on your local machine and wanted to put the file back into the home directory of the `mrnoodle` account on the remote machine—but under a different name so as not to over-write the original—you would use the command:

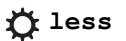
```
$ scp mycheese.jpg mrnoodle@mypc.example.com:newcheese.jpg
```

We're not exactly sure why MrNoodle and cheese feature quite so prominently in this lab either, just go with the flow. To summarise the behaviour of `scp`: if you prefix either or both of the source or destination files with the pattern `[USERNAME]@[HOSTNAME]`: then you are referring to a file on a remote machine, otherwise you are referring to a local file in the same way that you would with a regular `cp` command.

Now use your new-found knowledge of `scp` to copy the `.muttrc` file from your Computer Science account onto your Pi. You could use the hostname of the PC you're sitting at as the remote host, or alternatively you could use a server called `kilburn.cs.manchester.ac.uk` which is also set up to be able to see your Computer Science filestore. In fact, you could use the hostname of *any* of the PCs in this lab, because your Computer Science filestore is accessible to them all—but it's probably antisocial to use a machine that someone else is sitting at, so for now stick to using the one in front of you, or `kilburn.cs.manchester.ac.uk`.

Conveniently, there's nothing in the `.muttrc` file that is specific to the Computer Science account setup, so you can use it as-is for Mutt on the Pi (as usual, if you get stuck ask for help.)

Check that the file has copied over successfully using `less`, and then start up `mutt` from a terminal. If everything has gone to plan, you should now be able to read and compose emails on your Pi. You can of course install other mail clients if you want to; there is a slimmed-down version of Thunderbird for the Pi called Icedove (yes, yet another play on words), or a much leaner graphical client called Claws Mail (which, if you want to, can be installed using `sudo apt-get install claws-mail`). Remember, though, that the memory-card we've given you for your Pi is fairly small, so you probably don't want to clutter it up with unnecessary packages, and you should find that Mutt is perfectly okay for sending and reading the occasional email.



3.2.1 Emailing your tutor

To test that you've successfully set `mutt` up on the Pi, use it to send a friendly email to your personal tutor to tell him or her that you've reached this point of the lab (your personal tutor will be the person you met last week for lunch with your tutor group; don't email whoever it is that's running this lab unless they also happen to be your personal tutor). If you've not figured out your tutor's email address yet, you can find it using Epiphany at

<http://www.cs.manchester.ac.uk/about-us/staff/>

3.3 X Windows again

As we mentioned before, the X Windows system is a powerful system, and although it was designed a long time ago (around 1984), it was in many ways way ahead of its time (rather like the design of Unix itself.)

Remember that the GUI you're now using on the Pi consists of two systems working together: X Windows (which amongst other things gives software access to the display hardware), and the Window Manager / Desktop environment (in this case, LXDE) that provides the WIMP-style features such as movable windows and clickable controls. The X Windows system operates as a Client/Server architecture, where the server part does the drawing of stuff onto the screen, and clients request that things be drawn. One of the really nice features of X Windows is that it doesn't care too much about where the requests to draw things come from. Typically they come from processes that are running on the same computer as the X Server, but this need not be the case as you're about to demonstrate.

Log back in to the lab PC using the `ssh` command, but this time include a `-X` switch before your username, like:

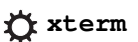
```
$ ssh -X [USERNAME]@[HOSTNAME]
```

The `-X` switch (note that it's an uppercase X) tells the `ssh` program to send any graphics instructions generated on the remote system back through the network to the X Server running on the local machine.

Confirm that you are indeed logged into your Computer Science account by checking the command prompt and using `ls` to make sure the files in your home directory are the ones you'd expect (or use one of the techniques described in Breakout 3.2 to check the hostname), and then type:

```
$ xeyes
```

Googly eyes™ that follow the mouse! What's not to like? Well, okay, perhaps not hugely exciting in itself, but what's actually happening here is rather interesting and quite sophisticated. The `xeyes` program is running on the remote machine (the desktop PC); but the instructions to draw its graphical output are being forwarded over the secure shell connection you've made from the Pi to the remote machine, so that the Pi's X Windows system receives them. Make sure that the terminal window you used to start `xeyes` is the active window, then press `<ctrl>c` to quit `xeyes`, and instead try running `xterm`. You should see a new terminal window appear on your Pi's screen (which probably looks slightly different to the terminal you launched on the Pi a moment ago; it does essentially the same thing though). This X Terminal, rather like `xeyes`, is actually running on the desktop machine—only its graphical representation is appearing on your Pi (if you use `ls` in that terminal, you'll see that it's your Computer Science account that's visible, rather than your Pi's filestore and you can confirm this another way using the `hostname` command). Look at Figures 3.2 and 3.3 for a diagrammatic representation of what's just happened. Now exit this X Terminal by typing `<ctrl>d` in it.



3.4 A Simple Web Server

This next exercise involves setting up a simple web server on the Pi, but before we can do that you'll need to create some basic web pages to display.

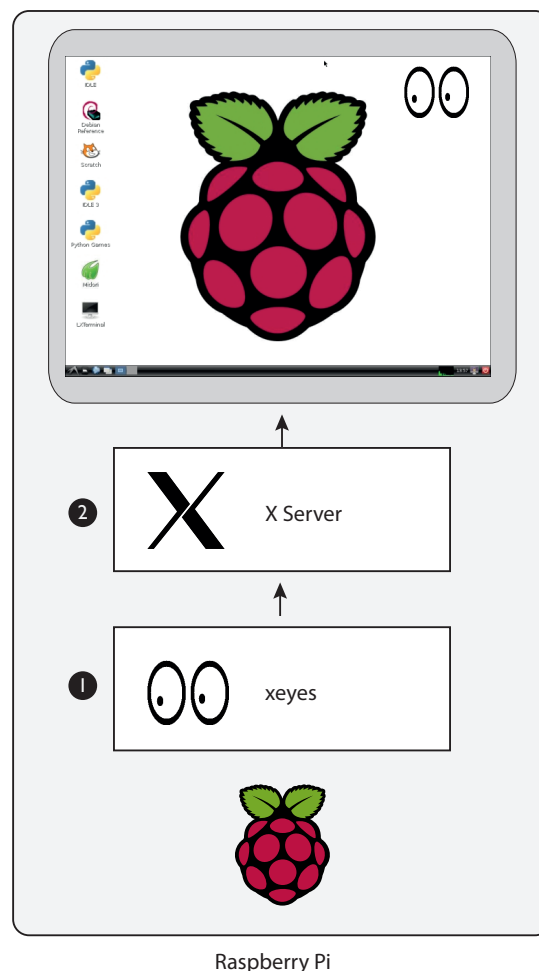
Start up a terminal on your Pi, and use the `mkdir` command to create a directory called `htmlexample1` in your home directory, and in that directory, use `nano` to create a file called `index.html` with the following content:

```
<html>
  <body>
    Hello World!
  </body>
</html>
```

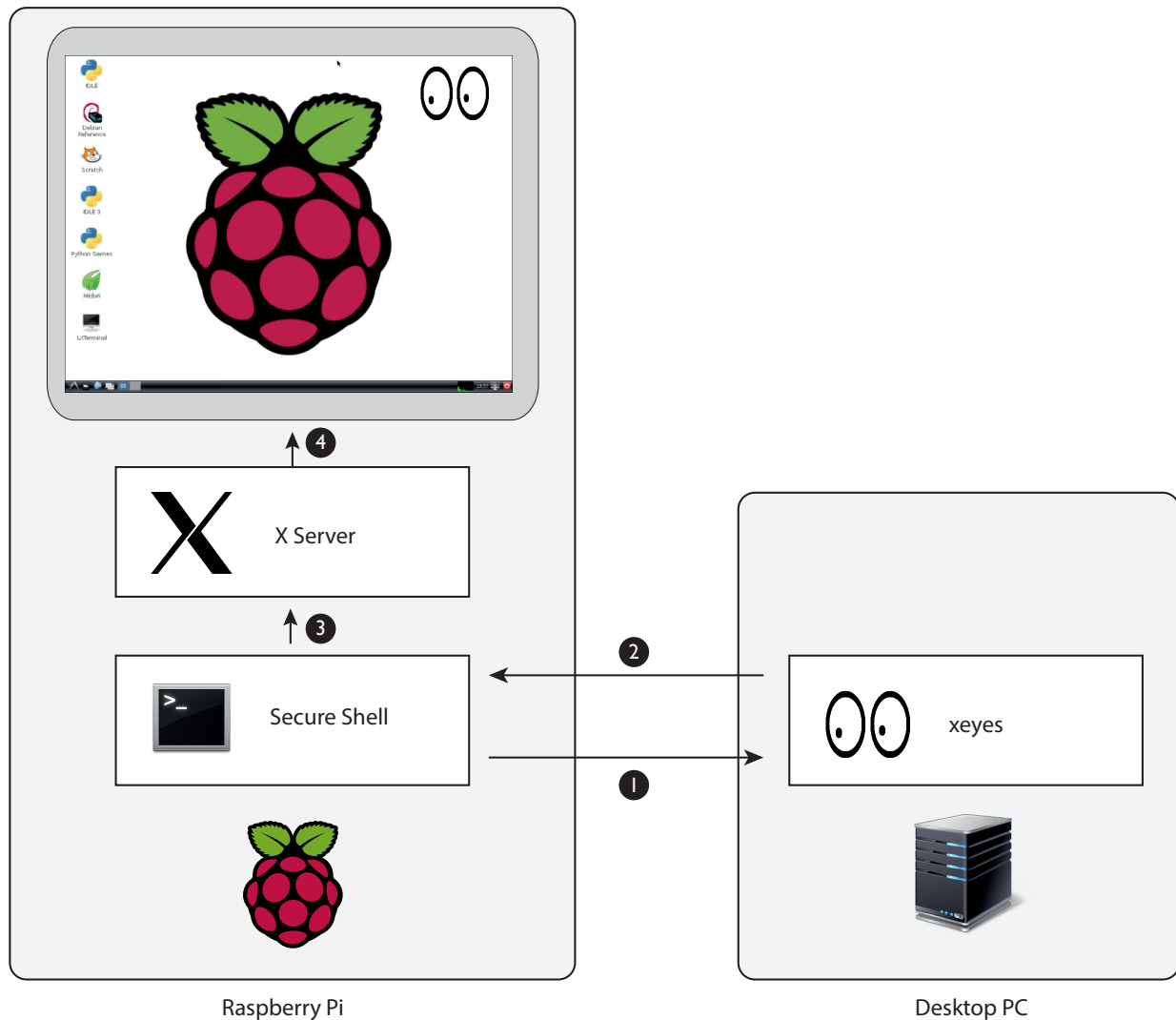
From within that directory, run the command:

```
$ python -m SimpleHTTPServer
```



**Figure 3.2**

In Unix, when you run a program such as `xeyes` on your local machine, it doesn't draw directly to the screen, but instead communicates with an X Server that by default is running on the same machine. In our example, the `xeyes` program ❶ connects to its local X Server ❷, and asks it to draw a window containing two ellipses to represent eyeballs, and then does some calculations to work out where to draw the two smaller ellipses to represent the pupils of the googly eyes so that they follow the mouse pointer. The X Server then communicates with the graphics hardware of the local machine in order to make things appear on the display that you see as a user.

**Figure 3.3**

One of the powerful features of the X Windows system is that the program generating graphical output, and the display on which the graphical output appears need not be on the same physical machine. In this example, we've used a Secure Shell connection from the Pi **1** to make a connection to the desktop PC, and have used the `-x` option to ask that any graphical instructions for the X Windows system get routed back through that connection. We then start the `xeyes` program on the desktop PC. Now although the calculations to draw the googly eyes are being done on the remote PC, the instructions to draw ellipses are sent back via the `ssh` connection to the Pi **2**, and are **3** routed to the X Server that's running on the Pi. As before, the X Server **4** interprets these instructions (not caring that they originated on another machine), and draws the eyes on the display that's physically connected to the Pi.


which starts a very basic **Web Server** and then use **Epiphany** to browse to the following URL:

```
http://localhost:8000
```

If your browser displays a page saying ‘Hello World’, pat yourself on the back—you’ve just created a web page and set up a simple web server to host it. The URL you used to view this page may look a bit odd compared with others you have seen. The `http://` part you’ll no-doubt be familiar with from other web-addresses that you’ve seen; the `localhost` part is a convention that means ‘this machine’. The section of the URL that follows `localhost` may be even less familiar: this is the *port* (for now just think of this as a kind of communication channel) on which the simple web-server that you’ve set up is serving web pages; by default web browsers expect servers to use port 80, but the `python -m SimpleHTTPServer` command you used here defaults to port 8000, so we had to add that to the URL. We’ll leave the issue of ports there for now, and revisit that in more detail in the second semester in COMP18112.


3.4.1 A slightly more interesting web page

Next you’re going to create a slightly more interesting web page that contains a paragraph describing who you are, which programme you’re on (e.g. Computer Science, Software Engineering, etc.), and which contains a picture of yourself as well as the image you created in the previous session using Inkscape.


Use **cd** to change to your home directory, and then **curl** to fetch a gzipped tar of the example Mr Noodle web page that we’ve created for you from:  **cd**

```
http://studentnet.cs.manchester.ac.uk/ugt/COMP10120/files/mrnoodle.tar.gz
```

In your home directory, use `tar` to ‘untar’ this bundle of files using the command:

 **tar**

```
tar xvzf mrnoodle.tar.gz
```

Notice that this time we’ve added an extra `z` switch to the `tar` command’s argument which tells `tar` to both ‘unzip’ and then ‘untar’ the file in one go (if you remember when you downloaded the Quake bundle in the first Pi session, you did this in two stages, first using `gunzip`, and then `tar`).  **gunzip**

You should end up with a `htmlexample2` directory containing an `index.html` file and two image files, one in `.png` and the other in `.jpg` format.

Use a text editor to change the content of `index.html` so that it says something about you. Don’t worry about finely crafted words here—this is really just a way of creating some files that we can get you to edit in various ways a little later on. A few sentences will do, and you can always change it later. Replace the default images we’ve provided with something that’s relevant to you. Perhaps a photograph of yourself, and whatever it was you drew in Inkscape would work here (though note, you’ll have to reopen the picture in Inkscape and export it to a bitmap format such as `.jpg` or `.png` to be able to include it in your webpage).

You’ve already learned several ways of getting files onto your Pi, but here’s a reminder:

- You could mail the photo to yourself, and use Mutt to save the attachment onto the Pi (for help on how to do this see Mutt’s online FAQ at <http://dev.mutt.org/trac/wiki/MuttFaq>)

- If the picture is already on the web somewhere, you could use Epiphany to find and save it.
- Alternatively for images on the web, you could use `curl` to fetch it directly from a URL to a file.
- You could use `scp` to copy it from some other machine directly to your Pi.
- Or if all else fails, you could use a USB device to copy it from one place to another.

If you need more guidance on how to write the web page itself there are plenty of tutorials on the web (search for something like ‘basic HTML tutorial’).

To finish this section, start up the SimpleHTTPServer that we used earlier *but this time in your `htmlexample2` directory*, and use Epiphany to check that your web page is displaying correctly.

3.5 Headless Pi

Don't Panic!

The rest of this lab session exposes you a fair number of quite advanced concepts pretty quickly, and we don't expect all of them to sink in this first time round. Stay calm. Don't panic. Just follow our instructions and everything will be fine; if you get lost or things aren't going to plan, just ask a member of lab staff to help you get back on track, and we'll revisit these ideas in later labs where there will be more opportunities to put them into practice and make sure everything makes sense.

The Pi can be used as a respectable desktop machine, but it really comes into its own as a server or controller for other pieces of hardware. In this section we'll use the Pi in what is called ‘headless’ mode—that is without its own screen and keyboard—to create a proper web server to host your pages.

In a terminal, type the command `ifconfig`, short for ‘interface configuration’, which will give you details about the network configuration on the Pi. This will return something along the lines of

```
eth0      Link encap:Ethernet  HWaddr b8:27:eb:a5:d5:82
          inet addr:10.2.233.1  Bcast:10.2.239.255  Mask:255.255.248.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:39778 errors:0 dropped:0 overruns:0 frame:0
          TX packets:4338 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:18251497 (17.4 MiB)  TX bytes:651537 (636.2 KiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:105 errors:0 dropped:0 overruns:0 frame:0
          TX packets:105 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:8550 (8.3 KiB)  TX bytes:8550 (8.3 KiB)
```

inet addr: 10.2.238.14

This tells you that the Pi has two network devices currently active: one called 'eth0', which represents the physical ethernet socket into which you plugged the blue network cable, and which allows the Pi to communicate with other computers on the network (and in this case, on the internet); and a second one called 'lo' which is a 'virtual connection' or local loopback connection that allows the Pi to route network traffic back to itself (this is how the 'localhost' trick you used earlier to look at web pages on the Pi worked). You can learn a lot more about network configuration in the second year Computer Networks course (COMP28411).

The thing we're interested in right now is the IP Address that's been allocated to your Pi. Look for the line in the eth0 block of text that says inet addr: and note down the number that follows this in your logbook (in the case of our example that is 10.2.233.1 but in your case it will probably be something else). Don't worry too much about what this number means or where it came from for now—we'll return to this in COMP18112 in the second semester. For now, just treat this as being a unique number that identifies your Pi on the Computer Science network.

Quit the graphical environment, and log out of your Pi by typing logout at the console's command prompt (or you can press <ctrl>d to achieve the same effect). Leave the network connection and power supply plugged in, but disconnect the mouse/keyboard lead, and reconnect them to the desktop PC. Switch the monitor over to display the desktop PC's screen, and log in to that with your University credentials.



Once you're in the graphical window environment on the desktop PC, start up a terminal, and use ssh to log into your Pi:

```
$ ssh pi@[IPADDRESSOFPI]
```

replacing [IPADDRESSOFPI] with the IP Address you noted down a moment ago. Since this is the first time you're logging in from your CS account to your Pi, expect to see the 'The authenticity of host' warning again; just say 'yes' to the prompt, and enter your Pi user's password.

Change directory to the htмлexample1 directory you created for your web-page earlier, and re-start the simple Python web server.

Next, start up Firefox on your desktop using the keyboard shortcut you created in the previous lab session, and enter the URL:

```
http://[IPADDRESSOFPI]:8000
```

and you should see your web page appear, served off your Pi to the desktop machine just like a real web server. Get the person next to you to see if they can see your web page from their machine by using your Pi's address; and return the favour by checking that theirs is also working (it's worth noting that the IP addresses of the Pis are only visible within the School of Computer Science, so pages served off your Pi will not be visible on the wider Internet).

3.5.1 Setting up Apache, a proper web server

The simple Python-based web server that you've been running so far is doing the bare minimum necessary to allow HTML pages to be fetched by a browser. Although it was a handy way of getting you going with web server technology, it's a long way off being the kind of fully-featured web server you would need to run a modern website. Fortunately, the Apache HTTP Server—the software that is used to run over half the world's websites—is Open Source

total 76204						
-rw-r--r--	1	pi	pi	138240	May 13 16:47	Advent.z5
drwxr-xr-x	2	pi	pi	4096	May 14 14:28	build
drwxr-xr-x	2	pi	pi	4096	Oct 28 2012	Desktop
drwx-----	2	pi	pi	4096	May 11 10:50	Mail
-rw-r--r--	1	pi	pi	1726	May 11 12:06	muttrc
drwxr-xr-x	2	pi	pi	4096	Aug 7 13:23	mysite
drwxrwxr-x	2	pi	pi	4096	Jul 20 2012	python_games
drwxr-xr-x	8	pi	pi	4096	May 13 17:26	quake3

Figure 3.4

An example of the 'long format' output from ls.

and runs quite happily on a Raspberry Pi. You wouldn't want to be powering the next eBay or Facebook from a Pi, but to illustrate the principles, it'll do the job nicely.

Before we install Apache on your Pi, there's a bit of housekeeping to do that will conveniently expose you to a few more Unix concepts that we've rather skated over so far.

3.5.2 Permissions, users and groups

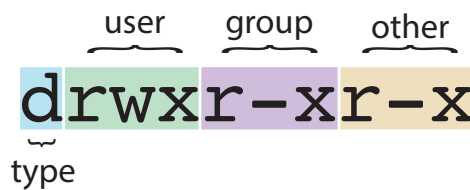
Right at the beginning of these sessions you logged in to the Pi as a particular user called 'pi', and on the desktop machines you've been logging in using your University username and password. It's fairly obvious what the general principles are here—logging in with a username and password is a way of protecting 'your stuff' from being seen or messed around with by other users. Roughly speaking this means two things: first, and most obviously, files created by you should be in some sense 'owned' by you so that you can control who can see/modify them; second, and perhaps less obviously, processes that you start—whether from the command line or the graphical environment—are also 'yours' and have certain privileges/restrictions that are associated with your user.

For this to work, it means that both the Unix file system and its way of dealing with processes need to be aware of the notion of a user. Back in the terminal that's connected by ssh to your Pi, use cd to change to your home directory and type ls -l to list the files there in what's called 'long format' (the -l switch means 'give me extra information about each file'). You'll see something like Figure 3.4 (but without the coloured background, which we've added to help distinguish the different columns in the figure).

Working from right to left in Figure 3.4, column ⑦ contains the filename, column ⑥ gives the date that the file was last modified (the exact format of the date will vary so as to always be 'useful'; older files will have a year instead of an exact time, for example) and column ⑤ shows the size of the file in bytes. Columns ④ and ③ give the group and user to which the file belongs; we'll come back to this in a moment. Column ② shows the number of links associated with the file (ignore this for now), and finally column ① gives a summary of the file's permissions, which we'll look at again shortly.

In our example, the file's user (Column ③) is not surprisingly 'pi', which is the username under which you logged in. Run:

```
$ ls -l /
```

**Figure 3.5**

Unix file permissions as shown with `ls -l`. In this example shows the permissions for a directory that can be read from, written to and listed by its user (owner), but only read and accessed by group members or anyone else.

(i.e. ‘long format list of the root directory’), and you’ll see that **the files at the top** of the filesystem are **owned by a user called root**.

As well as being owned by a particular user, **each file is associated with a group** (shown in column ④, which in this case is a group called ‘pi’ too; although both the user and the group are called pi here, they are different things). Every user is a member of one or more groups. The idea of a Unix group is quite straightforward: **it’s a mechanism to allow collections of users to share resources with one another in a controlled way, while stopping other unwanted users from being able to access those resources**. For example, you might want to say “Only I as this file’s creator am allowed to modify or delete this file, but other members of my Tutorial Group can read the file’s content, and the file is inaccessible to everybody else”. In Unix each file can have three different types of access permission: read, write and execute (run). These permissions can be set for three different categories of user: user (‘owner’), group (a specific set of users) and ‘other’ (which means ‘everyone else’).

The file permissions shown in column ① of Figure 3.4 consist of 10 ‘slots’ as shown in more detail in Figure 3.5. The first slot indicates the type of the file, and appears as a – for a regular file or a d for a directory. The next three slots represent the **user’s** permissions and can consist of any combination of r for ‘readable’, w for writable and x for executable. The next three slots show the read/write/execute permissions for members of the file’s **group**, and the last three slots show the same set of available permissions for **other** (i.e. anyone else with access to the system). Execute permissions when applied to a directory mean that particular set of users can access the directory’s contents.

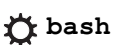
We’ll come back to file permissions in more detail shortly. In the meantime, let’s look at the ownership model for processes. Type:

```
$ ps waux
```

to **list any processes that are currently running** (you’ll probably need to widen the terminal window to see the output properly). The **ps** **command is unusual in that if used with no arguments**, its output is largely uninformative. It’s also **unusual in that explaining what even the most basic arguments do and how they interact** (i.e. the `waux` options in this case), **is quite complex**—so for now just treat this as a special recipe that does something useful. Note that there’s no hyphen in front of the options in this case.

**ps**

Look at the **rightmost column of the output (which has the heading ‘COMMAND’)** and you should **see a few familiar process names such as bash, startx, and the ps process itself** in amongst many commands that will be unfamiliar to you at the moment. The **leftmost column** tells you **who owns the processes**, and you’ll see that some of them are owned by you (or, rather by the user ‘pi’), and others are owned by the root user. Most of the processes that you’re seeing have been started as part of the window manager system, or are in one way or

**bash**

another associated with X Windows; the rest are ‘housekeeping’ processes the details of which we don’t care about for now.

If you think back to what you learned about the hierarchical relationship between processes in the first Pi lab, it should be starting to become clear how the ‘family tree’ of processes when combined with the notion of users and resource ownership knit together nicely to give a secure multi-user system. When you first log in to a Unix machine, a single process is started on your user’s behalf (in most cases this is a command shell). From that shell you can start other processes, which could be individual commands like `tar` or `ls`, or could be a whole window manager system such as GNOME or LXDE; in either case though, these inherit the properties of the parent shell in terms of being owned by the same user. Then, any processes that are started by the window manager also inherit the same user properties, and so forth.

3.5.3 Back to that webserver

Anyway, let’s get back to setting up the Apache web server. When you ran the Python web server, that was a process owned by you, and because we didn’t do anything special to protect it, if you closed the shell/terminal that was used to start it, it too would die. Generally speaking, you’d want a web server to continue running whether anyone was logged into the machine or not, so this isn’t ideal. The other issue with starting a web server in this way is that because the process is owned by you, it has access to all your files—so if a malicious hacker was able to take control of the web server process, he or she would be able to read or even delete your work, which clearly isn’t a good thing.

So to set up a web server properly we need to:

- make sure that it can continue to function after you’ve logged out, and
- somehow make sure that it can only access a specific set of resources that represent the website, and not run riot around your filestore doing bad things.

Let’s first tackle the issue of a shared directory; this is quite easy to set up using Unix’s group mechanism.

By default, the Apache package that we’re going to install shortly is set up to serve web pages from a directory called `/var/www` and although you can change it in Apache’s configuration files to be elsewhere, that’s as good a place as any for what we need.

Let’s go ahead and install Apache using the command:

```
$ sudo apt-get install apache2
```

It’s a reasonably large package so may take a little while to install; just be patient. Once that’s completed, take a look in `/var` using `ls -l` and you should see a directory in there called `www` which is owned by `root`.

Before we start serving any web pages, there are a few security issues that we need to take into account. In its simplest form, where you’re just serving static web pages (i.e. HTML files), Apache is secure enough, but as soon as you start creating web sites where users can add and modify content on your server, you potentially open yourself up to malicious damage, so it’s best to take some initial precautions.

Take a look at the permissions for `/var/www`, which was created when you installed the Apache package, and you will see that they are set to `drwxr-xr-x` which means ‘this is a directory that

the owner can read, write and access the contents of, and which both group and other users can only read and access'. Notice too that there's a single file in that directory called `index.html`, which is an 'it works!' declaration that you can use to test that Apache is doing the right thing (and which we'll do shortly). The important thing here is that anybody or any process that is not running as the root user has only got **read** and **not write** access to the contents of the `/var/www` directory structure—and this is exactly what we want. They can look, but not touch, and so can't break anything.

But how do we get content into the `/var/www` directory so that it can be served by Apache? We could simply use `sudo` to give ourselves root privilege every time we need to modify a file, but that's a rather clumsy solution: we may not want other users on our machine to have root access at all, and in any case you really do want to keep `sudo` commands to a minimum so that you don't accidentally do something bad to your system. So let's create a new group that will represent users on the Pi that are allowed to put content into that directory. Use the command `addgroup` to create a new group called `www-contrib` (for 'web contributors'). The syntax is straightforward:

**addgroup**

```
$ sudo addgroup www-contrib
```

And you should see that a **new group has been created with a Group Identifier (GID) probably set to 1004**.

Change the **group ownership of `/var/www` to be `www-contrib`** using the command

```
$ sudo chgrp -R www-contrib /var/www
```

and check that this has changed using `ls -l`. Note that we're using the `-R` option to `chgrp`, which means 'apply this change not only to `/var/www` itself, but to all files and directories contained within it' (`R` here stands for **recursive**^w). Next we need to add the `pi` user to that group. Type

**chgrp**

```
$ groups
```

to see what groups your user is already a member of (you'll see things like `pi`, `adm`, `dialout`, `cdrom`, `sudo` and several others). Use the command:

```
$ sudo usermod -a -G www-contrib pi
```

to add the `pi` user to the `www-contrib` group. The `usermod` command can be used to modify lots of things about a user; here we are using with the combination of `-a` and `-G` to mean 'append this group to the user's list of groups'. You'll need to log out and back in again to see this change take effect, so do that now and then run the `groups` command to confirm that `pi` is now indeed a member of group `www-contrib` as well as the other original groups.

**usermod**

We now need to make the `/var/www` folder **writable** by members of our new group, so do that using the command:

```
$ sudo chmod -R g+w /var/www
```

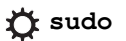
which reads as 'modify the permissions of the directory `/var/www` and any files/directories in it to add write access for members of its group'.

Now if all has gone to plan, the user `pi` should be able to read and write to the directory `/var/www`. Try this out by editing the content of the `index.html` file that's in that directory (and which was created when you installed Apache) to say something different from the default text.

We're finally ready to start Apache. Use the following command:

```
$ sudo service apache2 start
```

to initialise the server (you can use a similar command with `stop` to shut it down again), and then use `ps waux` to confirm that you can see several processes with `apache` in the name now running. Note that although you started the server with `sudo` which runs things as root, that the owner of the Apache processes is `www-data`. The `service apache2 start` command has restricted Apache to running as a regular, non-root user, and you should by now be able to work out why having a web server running as root would be a Bad Thing from a security point of view.

**sudo**

On the desktop machine use a web browser to see what content the Pi is now serving. The URL will be:

```
http://[IPADDRESSOFPI]
```

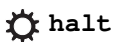
and since we're now using a standard web server on a standard port (port 80), there's no need to add the `:8000` port number as we did in Section 3.5.

You should see the 'It works!' page appear, with whatever modification you made to it a moment ago.

Nearly finished. Copy the content from your `htmlexample2` directory so that its served via Apache, and check using a web browser that the 'It works!' page has been replaced by your own masterpiece.

3.6 Shutting down gracefully

That's all the exercises done for this lab session; but before you finish there's one last important thing to do. Make sure you use the `halt` command to shut your Pi down cleanly, otherwise you risk losing some of your work by corrupting its file system. You can do this over a `ssh` connection quite safely, so you don't need to reconnect the keyboard/mouse/monitor to the Pi, just type `sudo halt` in the secure shell that's running on the Pi. Of course, you shouldn't be surprised when secure shell gets disconnected as the Pi begins to shut itself down.

**halt**

3.7 Finishing up

Don't worry if not everything you've done in this session has sunk in yet—you've encountered a lot of new concepts in this session, and we'll be revisiting everything in one form or another in future labs. This is the last lab that uses your Pi for a while now, and in the next session we'll be introducing you to the virtual learning environment that will be used for several of the course units this year, and giving you an opportunity to practise some of the skills and techniques that we've covered already. But this isn't the last you'll see of the Pi—you'll be using it as part of your group project in **COMP10120**, so please don't lose it. In the meantime,

remember that even if you do something wrong on the Pi, it's very easy to restore it back to a clean state, and we encourage you to explore Unix and have fun with this fantastic bit of hardware.