The keyboard is mightier than the mouse!

A Linux command line taster workshop
(including an introduction to LaTeX)

Dr. John Latham
School of Computer Science
University of Manchester

jtl@cs.man.ac.uk

November 28th, 2012 (Final draft)

Contents

Chapter 1

Logging in and basic commands

In this first chapter you will use Linux, perhaps for the first time, and be exposed to the idea of using a **command shell**, meeting some Unix (Linux) commands along the way. You will also have the opportunity to observe the difference between the **operating system** and a **window manager**, simply by us at first using Linux without a window manager!

By the way, in case you have not used Linux before, it should be noted that Linux can be used in much the same way as most people use Windows – i.e. with a file browser and much use of the mouse. Indeed, this is how most people who use Linux use it – the default environments for Linux have been deliberately designed to make them easy to use by those who know how to use Windows. However, this workshop isn't so much about Linux as the Linux command line – and so today you will be using Linux in a different and more clever way compared with that default!

1.1 Logging in

You are about to **login** to the School Unix (Linux) network, for which purpose you will need a **user name** and **password**.

Your **user name** for today is the word `lw` (that's an el, not a one) followed by a three digit number, followed by `g`. It is most important that everyone chooses a different user name, so please use the three digits we give you by hand. Your **password** is `Np4lw10`. (That's an el after the 4, and a one after the `w`, finishing with a zero.)

The machine should be displaying the **login prompt**, which will look something like the following.

```
ff029 login:
```

(**ff029** is just the name of the machine displaying the prompt, yours will probably be different).

Note: If you use the numeric keypad to the right hand side of the keyboard during the following sections, you must ensure 'Num Lock' is on to input numbers.

Now type your **user name**, followed by the return key ⏎ . E.g.

```
lw0XXg ⏎
```

except you will use the three digits we gave you personally instead of `0XX`.

Whether you typed your **user name** correctly or not, the machine will now ask for your **password**.

```
Password:
```

You now have to type the **password**. Note that when you type your **password**, *nothing* will appear on the screen, not even a * for each character. This is to prevent anyone looking over your shoulder seeing your **password**, or even how long it is.

```
Np4lw10 ⏎
```

(That's an el after the `4`, and a one after the `w`, finishing with a zero.)
If you make a mistake either in the **user name** or the **password** then you must type *both* of them again. If you cannot **login** after several tries, ask for help.

1.1.1 Choice of window manager

When you have logged in, the machine responds with assorted information, and then prompts you with the following.

```
          Manchester University, School of Computer Science
                    Linux Desktop Switcher

 Your desktop is currently set for:  AnotherLevelUp Fvwm95


 Please choose your desktop from the list below:

 <RETURN>     Continue to use AnotherLevelUp Fvwm95
 1            Use AnotherLevelUp Fvwm95
 2            Use GNOME
 3            Use KDE
 6            Use AnotherLevelUp Lesstif (Mwm)
 8            Use FVWM (vanilla)
 9            Use twm

 C            Console login
 I            Start X Interactively (pick from available screens or layouts)
 R            Start X With window manager on a Remote host
Enter choice:
```

This is the list of **window manager**s you can choose to use under **X-windows**. For the moment, we do not want to go into X-windows, so type

```
c ←┘
```

to choose the simple **console** interface (an upper or lower case `c` will do here).

The machine now gives you a prompt, which will be something like

```
lw0XXg-)
```

to show that it is waiting for you to type a command.

1.2 Some simple Unix (Linux) commands

In this section you will learn some basic Unix (Linux) commands. You will also set up your personal data ready for the first problem in the next chapter.

You are interacting with Unix (Linux) through a single **command shell** as your main **login process**. This allows you to type commands and have them executed by the operating system. The specific **command shell** you are using is called `bash`. This is an extremely powerful

program that obtains commands from the user, executes them, and displays the results. Unix (Linux) offers a variety of different **command shell** programs, but `bash` is the most popular and probably the most powerful.

Your first command is called `echo` and all it does is print its arguments on the **standard output**, which will appear on the screen. Type

```
echo John
```

except you should use *your* first name, instead of mine (`John`)! Please do not include any spaces or punctuation in your name – just a *capital letter* followed by some lower case letters.

A useful thing to remember is that typing the **up arrow key** scrolls backward through your **command history**. Use it right now to step back up to the previous command, use the **left arrow key** and the **right arrow key** to edit any errors in your name, and then press ⏎ to run it again.

Now run the following command, using your name instead of mine. (You do not need to retype the `echo...` – use the **up arrow key**!)

```
echo John > me2811.txt
```

There should be no output this time. Instead the `>` has **redirect**ed the **standard output** to the file `me2811.txt`.[1] By the way, unlike Windows, Unix (Linux) has **case sensitive file names** – so if you called it `Me2811.txt` or `ME2811.txt`, etc., please do it again – you will see why later.

How do you know that the previous command created a file called `me2811.txt`? Run the following. (Note: the first character and the one after the hyphen is an el, not a one.)

```
ls -l me2811.txt
```

The command `ls` shows the names of file(s) matching its argument(s) and with the `-l` option it shows details about those files. In this case it will show you information about the file you just created, including **file permissions** (who can read or write the file), the **file owner** (your **user name**) the **file size** (a small number of bytes – the length of your first name plus one) and the **file modification time** as a date and time.

How do you know what the file contains? Run the following.

```
cat me2811.txt
```

The command `cat` simply lists the contents of the file(s) given as its argument(s) to its **standard output**. In this case it should show your first name on the screen. If your name is not shown as a *capital letter* followed by lower case letters, please correct this (and read more carefully

[1] The string 2811 is being used to make the exact filename be dependent on this run of the workshop.

from now on). If you get this wrong, you may spoil the next chapter for everyone, depending on what your name is. . . .

Unix (Linux) is a **multi-user operating system**, and here in the School all user files are served across the network from a **file server**. This means you could sit at any machine in the School and access your files. For security reasons, only you can read (or write) your files by default, but for the purposes of this exercise, you need to make it so that all other users can read your file. Run the following two commands.

```
chmod a+r me2811.txt
ls -l me2811.txt
```

The command `chmod` changes the mode (**file permissions**) of the file(s) listed as its argument(s), depending on what options it is also given. In the above case it adds (`+`) readability (`r`) for all users (`a`). You can see from the output of the `ls` command that the permissions now have more **r**s listed than before.

Next you must make sure that other users can access your home directory, otherwise they still will not be able to read your file. Run the following (don't miss out the dots).

```
chmod a+rx .
ls -ld .
```

A single dot means the **current directory**. The first command adds (`+`) readability (`r`) and executability (`x`) for all users (`a`). In the second command, the `-d` option tells `ls` to show file permission details of the named directory itself, rather than the contents of it.

STOP AND CHECK IT!	Please get a teaching assistant to check you have set up your file and home directory properly, including correct capitalization of your name.

Now **logout**, by typing **Control-D** (hold the `Control` key and type `D`). This caused the **command shell** you were interacting with to exit, and as that was your main **login process**, you were logged off.

1.3 Using X-Windows

Log on again, and this time choose option 1 **AnotherLevelUp Fvwm95**. Simply press `1` ⏎ to select that choice, and then ⏎ again to go ahead with it. Note: if you have used Linux before, and you recognise your favourite desktop environment in the list then *please do **not** use it*! The whole point of this workshop is to expose some new material to you, and it is highly unlikely you have used the option 1 environment as it is developed and maintained by this School.

After a short delay, a number of windows will appear on the screen. You are now running **X-Windows** with a particular **window manager**.[2]

Now delete all the windows on the screen by clicking their **close button**s.

Move the pointer to anywhere on the blue background but preferably not near to the right hand edge of the screen. The background area is known as the **root window**. Now click the *left* mouse button, and you should see the **main menu**. You can also get this by pressing the **main button** on the task bar, but if you have clear space on the root window then it is much easier to get it from there. You will see a number of choices, some of which have a little rightwards pointing arrow next to them: these indicate **sub-menu**s. Find the item Preferences and follow it through to its **sub-menu** Stored preferences/Themes and on to Load shared theme and then on to the menus of particular desktop themes. Choose one with a name (and tiny icon) that suggests you might like it, by left clicking on it.

After a short while your desktop will settle down with your chosen theme. If you do not like it then choose another, but avoid getting distracted and spending too long on this!

Now obtain the **main menu** again (without using the **main button** on the task bar) and this time select a New Shell. A new window should appear. This is an **xterm** window, running an interactive **command shell** just like the one you used in the previous section in **console** mode. Find the button on the title bar of this window, fourth from the right, and use it to **tall maximise** the window. You do *not* want to **full screen maximise** the window – Unix (Linux) users typically want to do several things at the the same time in separate windows, and thus do not want to make any of these windows bigger than necessary.

You are now ready for the first problem in the next chapter.

[2] You may be interested to know that, contrary to popular myth, the notion of having multiple windows on the screen was not invented by **Microsoft**. They started using the idea after **Apple** did. But, neither was it invented by Apple, as a second myth would have you believe. It was in fact used as early as 1973 in the **Xerox Alto** computers. Inspired by the Alto, **X-Windows** was developed at **MIT** in 1984.

Chapter 2

Sending an email to each other participant

In this chapter you will be exposed to some more Unix (Linux) commands, and in particular how we can combine them together in various ways at the command prompt, to get the computer to do highly sophisticated and specialised tasks without the need to get a programmer to implement our requirements for us! At the end of the chapter you will send a separate email saying hello to all the workshop participants, using just one complex command made out of simple parts.

2.1 Collecting the names of all the users

Run the following command.

```
echo $HOME
```

`$HOME` is a **shell variable** which contains the **path name** of your **home directory** – the place where your files live. All of the computer accounts being used in this exercise have home directories next to each other, with the **user name** as the last part of the path name. Note that Unix (Linux) uses slash (`/`) to separate the names of the directories (folders) in the path name.[1]

Run the following command.

```
ls -l ../*/me2811.txt
```

`..` means **go up one level** and `*` means **match everything**. So, the output of this command was a list of all files called `me2811.txt` in the directories next to yours. You should see your file in there too.

Now we shall look at the contents of these files! Use the `cat` program to list the contents of all of them.

[1] **Microsoft** Windows uses backslash because DOS did, and DOS did because slash was used for command switches before it was imagined that DOS would have folders added to it.

```
cat ../*/me2811.txt
```

You should see the list of first names, one per line, in **user name** order. If somebody has forgotten to make their file readable then you will see an error message for their file! *Please tell a teaching assistant so we can get that sorted.*

It would be interesting to sort these names into first name order. We can do this by using a **pipe** (`|`) to put the **standard output** through the Unix (Linux) sorting program called `sort`.

```
cat ../*/me2811.txt | sort
```

Piping means we connect the **standard output** of the first command into the **standard input** of the second. The `sort` program takes its input, sorts it, and then outputs it.

Better still, let us remove any duplicated first names, and count instances of each name.

```
cat ../*/me2811.txt | sort | uniq -c
```

Here we have **pipe**d the **standard output** from `sort` into a program called `uniq` which deletes adjacent identical lines from its **standard input** to form its **standard output**. With the `-c` option, it prepends a count of the number of occurrences of each adjacent unique line.

A further **pipe** through another `sort` will help us find the most popular first names!

```
cat ../*/me2811.txt | sort | uniq -c | sort -n
```

(You didn't re-type all of the above line, did you?) The `-n` option tells `sort` to treat the data as numeric, so it will output `9` before `10`, rather than the other way round. Try the following two commands.

```
(echo 9; echo 10) | sort
```

```
(echo 9; echo 10) | sort -n
```

The round brackets `()` above mean "run the commands which are in the brackets in a **sub-shell**, thus grouping the output of them together". The semi-colon (`;`) is used to separate **two commands on the same line**.

Finally, we can count how many unique first names there are, using the `-n` option to `cat`! This option puts a number, counting upward, on each line of its **standard output**.

```
cat ../*/me2811.txt | sort | uniq -c | sort -n | cat -n
```

2.2 Pairing user and first names

Now we are going to make a list of pairs of **user name** and first name. To do this we shall use a **for loop** command. Run the following.

```
for file in ../*/me2811.txt
do
  echo $file
  cat $file
done
```

This loops through all the files that match the given pattern, and for each one, assigns the name to the **shell variable** file and then, inside the loop, uses `echo` to print that file name and uses `cat` to show its contents. This is nearly what we want – the information mapping each **user name** onto the person's first name.

But it's not quite right yet – we need to cut out the distracting bits of the **home directory path name**s, for which we need to introduce some more Unix (Linux) commands.

Run the following commands.

```
file=$HOME/me2811.txt
echo $file
```

Does that make sense? That is one of the values that $file has when the **for loop** is running – the one that corresponds to your **user name**.

Now use the `dirname` command to output the **path name** of the **directory** (folder) containing $file.

```
dirname $file
```

That should be the same value as your $HOME – of course. But later we shall use that for everyone's file, not just yours. We want to put that value into another **shell variable**, rather than have it appear on the output. Run the following – note that the quotes here *must* be **back-quotes** – *the one on the left of the keyboard*.

```
homedir=‘dirname $file‘
echo $homedir
```

A pair of **back-quotes** means "run the command in the quotes, but treat its **output as though it were typed on the command line**". So the output from running `dirname $file` gets assigned into the shell variable $homedir. This will, of course, have the same value as $HOME – but remember that we shall shortly use this command for everyone's details, not just your own.

Having got the **path name** of the **home directory**, we can extract the last part of it – which will be the **user name** – using the command `basename`.

```
username=‘basename $homedir‘
echo $username
```

This should show your **user name**, of course.

By a similar technique, we shall obtain the first name by getting the contents of the file and storing it in yet another **shell variable**.

```
firstname=‘cat $file‘
echo $firstname
```

Did that make sense?

Finally, to output the **user name** and the first name together on one line, with a tab character between them, run the following.

```
echo -e "$username\t$firstname"
```

The `-e` option tells `echo` to enable certain feature extensions, which include recognising `\t` as a short hand for a **tab character**.

Now, let’s put all that together to obtain the information for everyone!

```
for file in ../*/me2811.txt
do
  homedir=‘dirname $file‘
  username=‘basename $homedir‘
  firstname=‘cat $file‘
  echo -e "$username\t$firstname"
done
```

STOP AND CHECK IT!	Please get a teaching assistant to check that you understand the above code – feel free to ask for as much explanation as you need.

Now run that whole command again, but this time make its output go into the file all-by-user.txt using `>` to **redirect** the output. (You do not need to retype the command. . . .)

If you are wise, you are probably not *convinced* that you have got this data right, and would wish to check it against everyone else’s result. How can you do this? Well, an easy way would be to run the following.

```
sum all-by-user.txt
```

The `sum` command outputs two numbers – a **check sum** and a **block count**. Everybody ought to get the same two numbers, so check with each other and/or a teaching assistant to see if you have the right ones.

2.3 Swapping the order of the information

From the work in the previous section you have a file which maps **user name** to first name. In this section we shall create a file containing the reverse pairing. (Note: this may not be a **true function**, of course, because more than one of you may have the same first name. Whereas, the mapping you already have is a **true function** as each **user name** is unique.)

To build the second file with the pairs reversed we shall use a **while loop** command. Run the following.

```
while read username firstname
do
  echo -e "$firstname\t$username"
done < all-by-user.txt > all-by-name.txt
```

Can you figure out what `<` does? What `read` does? Look at the file you have just produced using `cat`.

STOP AND CHECK IT!	Please get a teaching assistant to check and explain this to you if you are unsure about any of it.

The information in all-by-name.txt would be more useful if it was sorted by first name rather than by **user name** – you could then more easily find a person you might be looking for. Run the **while loop** again, but this time **pipe** the output through `sort`, by inserting `|sort` just before the `>` symbol. (*Please* do not retype the whole command. . . !).

Check you have got this file correct in the same way as you did the previous one by comparing the result of the following command with that obtained by others.

```
sum all-by-name.txt
```

2.4 Finally, sending email to everyone!

Warning: you are about to use a very powerful facility of Unix (Linux) – to send email to people from within a loop. You must ensure you have got this right before pressing the final

↵ , otherwise you might possibly end up sending unsolicited email to users outside this room! This would be a bad thing.

There are only two new bits in this next command. The first is the **shell variable** $USER which contains your **user name**. The second new bit is the `mail` program which can be used to send email. Its **standard input** is the body message, its `-s` option means the following argument is the message subject, its `-c` option means the following argument is an email address to send a carbon copy to, and all remaining argument(s) are the email addresses of recipients. On a properly set up Unix (Linux) system, one can send email to a user in the same domain simply using their **user name** as the address.

Study the following code, and try to figure out what it would do.

```
while read username firstname
do
  echo Processing $username $firstname
  (echo Hello $firstname
   echo Best wishes from ‘cat me2811.txt‘
  ) | mail -s Hello -c $USER $username
done < all-by-user.txt
```

Now type it in carefully, and check it carefully before pressing the final ↵ . If it looks like it is misbehaving, just press **Control-C** to kill it.

You will want to look at your email. You should have received a copy of the messages you have just sent, and eventually, you will get an email from everyone else![2]

```
mail
```

You can read a message by typing its number. When you are ready, you can exit `mail` by typing `x` ↵ . If messages arrive while you are reading your mail, you will see them the next time you run the `mail` program.

If you prefer, and to further convince you that this is real email, you can start the `thunderbird` **email client** from a command prompt and read your email through that.

```
thunderbird &
```

(The `&` tells `bash` to run this command as a **background process**, so you will get your prompt back straightaway.)

You can check your mail again later, especially if you are one of the first to complete this chapter, and so have not had any or many emails from others. Because you ran it as a background process, there's no need to exit thunderbird before carrying on.

[2] Due to loading on the School mail servers, thanks to SPAM, it may take a while for your messages to arrive.

Chapter 3

Counting the different words in an essay

In this chapter you will meet some more Unix (Linux) commands and combine them together to make a useful tool. You will then store these commands in a **shell script** so that you could use that tool in the future.

Most word processors enable you to easily determine how many words are contained in a document, which is especially useful when one is writing an essay with a target word count. However, what can be more interesting is the number of *different* words, perhaps each with their frequency. This might be used to indicate something about the author's vocabulary and writing skills.

3.1 Obtaining and viewing an essay

Let us start by obtaining a plain text copy of an essay. The `cp` command allows us to copy a file from one place to another. Type the following, including the dot at the end, and observe the spaces separating the two **command argument**s from the `cp` command.

```
cp /opt/info/courses/LinuxWorkshop/essay.txt .
```

This means copy the file /opt/info/courses/LinuxWorkshop/essay.txt into the **current directory** (`.`) which will be your **home directory** (assuming you have not changed it).

You can view this wonderful piece of work using the `less` command.[1]

```
less essay.txt
```

You can use the arrow keys to scroll up and down. You can search for something, say `Lovelace`, by typing `/Lovelace` and the next occurrence of it by typing `n`, or backward with `N`. When you've finished being impressed with this literary masterpiece, exit `less` by typing `q`.

[1] If you're interested, ask a teaching assistant to explain why this program is called less!

You can find out how many lines, words and characters it has using the `wc` (**word count**) command.

```
wc essay.txt
```

3.2 Translating characters

But what about the number of different words? To find these we shall start by changing the line structure so that every word appears on one line. The `tr` command is good for this – it simply copies its **standard input** to its **standard output**, but makes a **character translation** based on a mapping supplied as **command argument**s. This mapping is specified by the user giving two strings: the first are characters that `tr` might find in its input, and the second are the corresponding characters they should be changed into. Try the following.

```
tr aeiou eioua < essay.txt | less -p Ade
```

This means `tr` takes its **standard input** from essay.txt (by **redirect**ion, `<`), and translates `a` into `e`, `e` into `i`, and so on. The resulting **standard output** is **pipe**d (`|`) into the **standard input** of `less`. This command, when given no files as **command argument**s, takes its data from **standard input** instead. Also, its `-p` options tells it to search for and highlight the string given in the next argument.

Exit `less` when you have finished viewing this quasi-Swedish version of the work.

For the purposes of solving the problem in this chapter, we are going to use `tr` to change every **space character** into a **new line character** using the following command.

```
tr " " "\012" < essay.txt | less
```

See how the first **command argument**, which is just a space, is enclosed in **double quotes** (`"`). The second **command argument** contains an **octal escape sequence**, and the octal number 012 corresponds to decimal 10, which is the code for the **new line character** in Unix (Linux). Together these tell `tr` to change every **space character** into a **new line character**.

All we need to do now is sort the words using `sort` and remove adjacent duplicates using `uniq`.

```
tr " " "\012" < essay.txt | sort | uniq | less
```

(You didn’t retype most of the above line, did you?)

You can see that there is also a blank line in this output – that corresponds to the lines between paragraphs in the original text. We can remove these using the `grep` command.

```
tr " " "\012" < essay.txt | grep -v "^ *$" | sort | uniq | less
```

`grep` is a program that copies its **standard input** to its **standard output**, but only includes lines that match a given pattern. With the `-v` option, it *excludes* lines that match the pattern, and includes all the others. The pattern above matches: the start of a line ("^") followed by any number of spaces (" *") and then the end of the line ("$"). So, all blank lines, including those containing only spaces, are removed by the `grep`.

What about punctuation etc.? We can remove this using `tr`, replacing all of them with new line too.

```
tr " [:punct:]" "\012" < essay.txt | grep -v "^ *$" | sort | uniq | less
```

The string `[:punct:]` is recognised by `tr` as meaning all punctuations characters. Notice the two map strings are now not the same length – `tr` simply repeats the last character of the second string if it is shorter than the first.

Another thing we have overlooked so far is the difference between upper and lower case letters. Let us change them all to upper case. The backslash at the end of the first line is the **line continuation** marker enabling us to split a line of commands into more than one physical line. (Press ⏎ after the backslash.)

```
tr "a-z [:punct:]" "A-Z\012" < essay.txt | grep -v "^ *$" | sort \
| uniq | less
```

This means the characters between `a` and `z`, inclusive, are translated to upper case, then space and punctuation are translated to new line.

Now all we need to do is to count the number of lines in this result.

```
tr "a-z [:punct:]" "A-Z\012" < essay.txt | grep -v "^ *$" | sort \
| uniq | wc -l
```

The `-l` of the `wc` command tells it to output only the number of lines. Also, like `less`, if no files are given as **command argument**s then `wc` takes its data from **standard input** instead.

3.3 Finally, word frequency count

To obtain a frequency count of the different words we can use the `-c` option of the `uniq` program, which causes it to prepend a count of the number of occurrences of each adjacent unique line.

```
tr "a-z [:punct:]" "A-Z\012" < essay.txt | grep -v "^ *$" | sort \
| uniq -c | less
```

A further **pipe** through another `sort` will help us find the most used words!

```
tr "a-z [:punct:]" "A-Z\012" < essay.txt | grep -v "^ *$" | sort \
| uniq -c | sort -n | less
```

The `-n` option tells `sort` to treat the data as numeric, so it will output `9` before `10`, rather than the other way round.

Perhaps you are interested in knowing just the most frequent 5 words, expressed in descending frequency?

```
tr "a-z [:punct:]" "A-Z\012" < essay.txt | grep -v "^ *$" | sort \
| uniq -c | sort -rn | head -5
```

The `-r` option of `sort` makes it reverse its order. The `head` command shows just the first 10 lines of its **standard input**, or the given number of lines if an optional number argument is supplied.

3.4 Making our own word count command

If we were asked to write lots of essays, then we might be likely to want to use our word counting technique on a regular basis. We can store commands in a **shell script** which we can later invoke as a command.

You are going to create a **directory** (folder) in which you will store your own commands. On the standard Manchester University Computer Science Linux set up, if a user creates a **directory** called `bin` just below his or her **home directory**, then any **executable file**s stored there can be treated as commands.

To create a **directory** we use the `mkdir` command.

```
mkdir bin
```

Next you will create a file called `dwc` in that directory, containing the command that shows the different word count of its **standard input**. You will use `cat` to collect **in-line standard input** from the keyboard and **redirect** it to the file. The syntax « `"."` means **treat the following lines as input**, until a line containing just a dot. There are two such lines of input. The first, `#!/bin/bash` is there to tell the Unix (Linux) **kernel** that the file is to be interpreted by the program `bash`. The second line is your command to create the result. You can use the up arrow key to find this from your **command history**, and edit out the `< essay.txt` part of it.

```
cat << "." > bin/dwc
#!/bin/bash
tr "a-z [:punct:]" "A-Z\012" | grep -v "^ *$" | sort | uniq | wc -l
.
```

Don’t miss the final line containing only a dot – that marks the end of the input.

Next you will want to make this file **executable** by changing its **file permissions** using the `+x` options of `chmod`.

```
chmod +x bin/dwc
```

Because you have only just created your bin **directory**, you will need to **logout** and **login** again before it is picked up by the system as a place where commands live. At the bottom of the **main menu** there is an Exit Fvwm option. Select it.

After logging back in, move the mouse back into the **xterm** window and type the following.

```
echo $PATH
```

The **shell variable** `$PATH` contains a list of places where the **kernel** will search for commands. Your new **directory** should appear at the end of that list.

So now, run your new command!

```
dwc < essay.txt
```

Chapter 4

How many dads from across the pond?

In this chapter you will meet and use the idea of accessing information from the worldwide web, without using a standard web browser.

4.1 Accessing the web non-interactively

The web is a huge and versatile source of information, but most people use it in a rather narrow way – with an interactive web browser. `wget` is a very useful command, especially when you know the URL of the file you want.

Try this!

```
wget http://www.manchesterdda.net/webcams/albert-square/photo.jpg
ls -l photo*
```

You should see the file photo.jpg which has just been downloaded from that web site. You can view it using the `display` program.

```
display photo.jpg
```

If all is working well, this should pop up a picture, taken within the last 20 seconds minutes, of Albert Square taken from the top of Manchester Town Hall. Close this window.

4.2 Updating our root window image, from the web

Why don't we put that image onto our desktop? One way of doing this is to use the `xloadimage` command, as follows.

```
xloadimage -onroot photo.jpg
```

We can put those two commands together, and avoid the need to save the image in a file. (Don't forget the **line continuation** marker at the end of the first line.)

```
wget -O - http://www.manchesterdda.net/webcams/albert-square/photo.jpg \
| xloadimage -onroot stdin
```

The `-O` (big oh) option tells `wget` to output the downloaded data to the file whose name is the next **command argument**, except that a single hyphen (-) is intepreted as meaning **standard output**. And `xloadimage` interprets the filename `stdin` as meaning **standard input**.[1]

How about if we put that into a loop? The command `sleep` can be used to have a pause for a number of seconds, given as a **command argument**. The ampersand (`&`) after the `done` below means run the command as a **background process**. That is, the **command shell** prompt comes back immediately, while the loop continues to run. Please notice the `-q` option to `wget` and the `-quiet` option to `xloadimage`: these make their respective commands run quietly, so their output does not clutter your xterm window. The `2>/dev/null` makes any actual error messages from `xloadimage` (e.g. caused by a corrupt image file) to be diverted to nowhere – so they won't appear on your screen.

```
while true
do
  wget -O - -q http://www.manchesterdda.net/webcams/albert-square/photo.jpg \
  | xloadimage -onroot -quiet stdin 2>/dev/null
  sleep 20
done &
```

If you get the above wrong, or in any case want to end it, then you can `kill` that **background job**.

```
kill %1
```

But if it is working then you don't have to – maybe it's nice to have your desktop updated?

4.3 Tonight's programmes on BBC3

Next we want to have a simple way of searching for programmes on TV that we might be interested in watching. Let us start by looking at the programme guide for tonight's BBC3.

[1] It would be nice if there was more consistency for this sort of thing, but it's not actually as bad as this pair of commands would have you believe.

```
wget -O beeb.html http://www.bbc.co.uk/bbcthree/programmes/schedules/ataglance
less beeb.html
```

If that worked, then you're looking at the **html** code for a page that shows the TV programmes. We're interested in extracting out the text describing those programmes, and what time they are on. We could write commands that parse the **html**, but we'll do something simpler than that. (Type `q` to exit `less`.)

There is a web browser called `lynx` which displays its pages in text format, so it can be run inside an `xterm` or in **console** mode. Let's try it.

```
lynx beeb.html
```

If you typed that correctly, then you should be looking at the **html** rendered as text. Press space to go down a page. Type `q` then `y` to quit.

The feature of `lynx` that we're most interested in here, is its ability to send the rendered text to its **standard output**, when we give the `-dump` option.

```
lynx -dump -nolist beeb.html | less
```

4.4 What is on most over the next week on BBC3?

Take a look at the following script. (Don't type it.) It runs `lynx` on the above URL, for each of seven dates, starting with today, in a **for loop**. (We'll see more about manipulating dates with the `date` command, and also the `seq` command in a later chapter.)

```
#!/bin/bash

for day in ‘seq 0 6‘
do
  date=‘date --date="+$day day" +%Y/%m/%d‘
  wget -O beeb.html -q \
       http://www.bbc.co.uk/bbcthree/programmes/schedules/$date/ataglance
  lynx -dump -nolist -width=1000 beeb.html \
```

The rendered output for each day is put through a `grep` with a pattern that will keep only lines starting with a letter or a digit, which all the line we are interested in, plus a few more.

```
| grep ’^[A-Za-z0-9]’ \
```

Now we remove a couple of lines we do not want.

```
  | grep -v '^Late$' \
  | grep -v '^BBC links$' \
```

And then we use the `tr` command to change newlines into spaces, so that it all becomes one line.

```
    | tr "\012" " " \
```

The command `sed` (short for **stream editor**) is then used to place a # at in front of the time for each programme, and also at the every end of the text, and these are then replaced by a newline.

```
  | sed 's, [0-9][0-9]:[0-9][0-9] - [0-9][0-9]:[0-9][0-9],#\1,g' \
  | sed 's, $,#,g' \
  | tr '#' '\012' \
```

Then we remove lines that do not start with a digit.

```
  | grep -v '^[^0-9]' \
```

Finally for each web access, the date is prepended on the front of each line (which already consists of a start time, end time and programme name).

```
    | sed "s,^,$date ,"
  done \
```

The whole output of the **for loop** is then **pipe**d through `sort`, with the `-k` option telling it to sort from field five (the first word of the programme name onwards).

```
  | sort -k5 | less
```

Well, I hope you didn't type it, because I already have. The tilde character ~ is a shorthand for your **home directory**.

```
  cp -ai /opt/info/courses/LinuxWorkshop/bbc3 ~/bin
```

Now run it to see what gets shown (and re-shown) over the next week. How many American dads are there?

Chapter 5

An introduction to LaTeX

In this chapter we shall take a look at LaTeX, an extremely powerful text formatting system. It is not **WYSIWYG** (what you see is what you get), like **Microsoft** Word is. Instead the author of a document concentrates on the *contents* of it, and leaves the layout to the typesetting experts that designed LaTeX. Thus, while writing the contents, the author does not get continually distracted by what it will look like, and in the end its looks better than he or she could typically manage anyway.

5.1 Obtaining the source of this document

Use the `cp` command to copy the source of *this* document as follows. The `-a` asks `cp` to make an **archive copy** of the given source, which means it will copy the directory and its contents as well. Don't forget the tilde (~) to signify that you want to copy into your **home directory**.

```
cp -a /opt/info/courses/LinuxWorkshop/NOTES ~
```

Now change the **current directory** to be your new `NOTES` **directory**, using the `cd` command.

```
cd ~/NOTES
```

You can confirm what the **current directory** is with the `pwd` command (**print working directory**).

```
pwd
```

Now see what you've got in here using the `ls` command. Take a look at the file `notes.tex`.

```
less notes.tex
```

This is the top level file of the document, and contains some definitions and among, other things, input references to other files, each of which is a chapter of the document.

Now look at latex-introduction.tex.

```
less latex-introduction.tex
```

You should recognise this as being the source of the chapter you are currently reading. The first line of this file tells LaTeX to start a new chapter, and what the name of it is. The author doesn't have to worry about what number the chapter has, nor that it starts on a new double page, nor that the document will have the chapter name on the page headings etc.. Then follows the text of the first paragraph, with some LaTeX commands embedded in it. These include ones that were written by the author especially for this document, such as `\concept` and `\company` which place their arguments in particular fonts, and also create index entries for them.

Next you will see a command that starts the first section of this chapter, and so on.

As the files are just text files, they are created using any simple **text editor**.

5.2 Producing the document

To process this text into a type-set document we use the `latex` command and, as it has index entries, the `makeindex` command. (Notice the need to run `latex` twice – this is because it is a one-pass system, and the second execution uses information obtained in the first, plus the output of `makeindex`).

```
latex notes
makeindex notes
latex notes
```

This has produced a **device independent** version of the document.

```
ls -l notes.dvi
```

To view this result, we use the command `xdvi`.

```
xdvi notes.dvi
```

Maybe we would like to have a **pdf** version of the document? First we turn the dvi version into **postscript** (from which it could be printed). Then we turn that into **pdf**.

```
dvips -f notes.dvi > notes.ps
ps2pdf -sPAPERSIZE=a4 notes.ps notes.pdf
```

You can view the **pdf** using `acroread`.

```
acroread notes.pdf &
```

5.3 Making a website for the document

A common thing we want to do nowadays with documents is put them on the web. We could place the **pdf** there, but perhaps it is nicer for our readers to have a **html** version.

```
latex2html notes
```

This should have built a website in the subdirectory notes. You can browse it to see what it is like.

```
firefox ‘pwd‘/notes/index.html &
```

Chapter 6

Generating documents

In this chapter you will meet the idea of using a combination of Unix (Linux) commands to generate a document, first one in plain text and then a LaTeX document.

One of the joys of the student experience is sharing a house with fellow students, and perhaps the very pinnacle of that person-enriching time is the process of deciding who should have cleaned the kitchen. And the bathroom, and the lounge. This situation clearly needs a chores rota – a table of weeks by date, with columns for the rooms that need cleaning, in which house mates can (be forced to) write their initials. At least we do not need to have the chore of creating all those dates – as you are about to see.

6.1 The date command

The command `date` can be used to show today's date.

```
date
```

We can control the format of the output with a range of options, such as the following.

```
date +"%a %d %b %y"
```

We can even tell it which date to use instead of today's.

```
date --date="Sep 1" +"%a %d %b %y"
```

Even more usefully, we can specify a date which is relative to another, by adding a number of weeks to it.

```
date --date="Sep 1 1 week" +"%a %d %b %y"
date --date="Sep 1 2 week" +"%a %d %b %y"
```

And days too.

```
date --date="Sep 1 1 week 6 day" +"%a %d %b %y"
date --date="Sep 1 2 week 6 day" +"%a %d %b %y"
```

One of the things we want to do is run that sort of `date` command in a loop, a fixed number of times. For counting we can use the `seq` command, which generates numbers between two given **command argument**s.

```
seq 0 9
```

We simply want to put those numbers as the values of a **for loop**, using **back-quotes**.

```
for w in ‘seq 0 9‘
do
  date --date="Sep 1 $w week" +"%a %d %b %y"
  date --date="Sep 1 $w week 6 day" +"%a %d %b %y"
done
```

6.2 Generating a cleaning rota

The flexibility of `date`, combined with the usual **command shell** power gives us all we need to create a text format cleaning rota.

Study the following script (don’t type it in).

```
#!/bin/bash

FirstDate=‘date --date="$1" +"%m/%d/%y"‘
test "$FirstDate" = "" && echo "Please specify first date" && exit

echo Cleaning Rota
echo
echo Starting ‘/bin/date --date="$FirstDate" +"%a %d %b %Y"‘
echo

echo "Date Kitchen Bathrm Lounge"
echo
echo "-----------------------------------------------------"
for count in ‘seq 0 25‘
do
  Date1=‘/bin/date --date="$FirstDate $count week" +"%a %d %b %y"‘
  Date2=‘/bin/date --date="$FirstDate $count week 6 day" +"%a %d %b %y"‘
```

```
  echo "$Date1 - $Date2 | | | |"
  echo "-----------------------------------------------------"
done
```

Now obtain it and run it.

```
  cp /opt/info/courses/LinuxWorkshop/cleaning-rota-text ~/bin
  cleaning-rota-text "Sep 1"
```

6.3 Generating a cleaning rota using LATEX

Here is a more elaborate version of our cleaning rota that generates a LATEX document, so that we get a nicely formatted document. (Don't type it in.)

```
#!/bin/bash

FirstDate=‘date --date="$1" +"%m/%d/%y"‘
test "$FirstDate" = "" && echo "Please specify first date" && exit

(cat << "."
\documentclass[12pt]{article}
\usepackage{a4-mancs}
\begin{document}
\begin{center}
\Huge
\textbf{Cleaning Rota} \\[.3em]
\small
.

echo Starting ‘/bin/date --date="$FirstDate" +"%a %d %b %Y"‘
cat << "."
\end{center}
\large
.

cat << "."
\begin{tabular}{|l|p{0.8in}|p{0.8in}|p{0.8in}|} \hline
Date & Kitchen & Bathroom & Lounge \\ \hline
.

for count in ‘seq 0 33‘
do
  Date1=‘/bin/date --date="$FirstDate $count week" +"%a %d %b %y"‘
  Date2=‘/bin/date --date="$FirstDate $count week 6 day" +"%a %d %b %y"‘
```

```
  echo "$Date1 - $Date2 & & & \\\\ \hline"
done

cat << "."
\end{tabular}
\end{document}
.
) > rota.tex

latex rota.tex
xdvi rota
```

Now obtain this version and run it.

```
  cp /opt/info/courses/LinuxWorkshop/cleaning-rota ~/bin
  cleaning-rota "Sep 1"
```

You might be interested to see what the generated LaTeX looks like.

```
  less rota.tex
```

Chapter 7

Making a photo website

In this final chapter you see more uses of Unix (Linux) commands combined to manipulate a large number of files in a systematic way – a job that would take ages by hand with a file browser. In the end you will have generated a website comprising pictures taken by different photographers at an 18th birthday party!

7.1 Obtaining the photos

Let us say you've been to an event at which lots of people took lots of photos, and you wish to build a website showing all these in chronological order. You asked the photographers in advance to make sure the clocks on their cameras are accurate, and you plan to trust that the **file modification time** of each file is the moment when the photograph was taken.

Use the `cp` command to copy a sample characterisation of this problem as follows. The `-a` asks `cp` to make an **archive copy** of the given source, which means it will copy subdirectories as well, and preserve **file modification time**s. Don't forget the tilde (~) to signify that you want to copy into your **home directory**.

```
cp -a /opt/info/courses/LinuxWorkshop/PHOTOS ~
```

Now change the **current directory** to be your new `PHOTOS` **directory**, using the `cd` command.

```
cd ~/PHOTOS
```

You can confirm what the **current directory** is with the `pwd` command (**print working directory**).

```
pwd
```

See what you've got in here using the `ls` command. The `-R` option asks `ls` to **list files recursively** down the file structure.

```
ls -lR | less
```

So, you have a **directory** of photos from each photographer, with the directory being named after the photographer.

7.2 Renaming the photo files

Next you want to copy all the photos from the sub-directories to here, but because there are obviously duplicated file names, you shall prepend the name of the photographer onto each one. Type the following (very carefully).

```
for dir in *
do
  for file in ‘cd $dir; ls *.jpg‘
  do
    cp -ai $dir/$file $dir-$file
  done
done
```

Take a look at the result using `ls`.

```
ls -l *.jpg | less
```

You should see all the photos listed here, each called by the photographer and then the original filename, separated by a hyphen.

STOP AND CHECK IT!	Please get a teaching assistant to check this, and if it is not right he or she will help figure out what you did wrong and fix it.

The files were shown sorted by photographer, rather than by time. We can ask `ls` to **list files by time** in **file modification time** order using the `-t` option. That actually lists them in *reverse* chronological order, so we also use the `-r` option asking it to **list files in reverse order**, i.e. in chronological order!

```
ls -rt *.jpg | less
```

How many photos are there?

```
ls *.jpg | wc -l
```

Let's set a **shell variable** to hold that value.

```
count=‘ls *.jpg | wc -l‘
echo $count
```

We wish to prepend a sequence number to each file name, with the oldest having `001` and so on. The command `seq` can be used to generate numbers between two given **command argument**s. The `-w` option tells `seq` to make the numbers all have the same width, by placing leading zeroes as required.

```
seq -w 1 $count
```

Finally we are ready to rename all the files to prepend the sequence number. In Unix (Linux) one renames a file using the `mv` command.[1] The `-i` tells `mv` to confirm with the user if a file of the new name already exists, and would thus be lost. Type the following carefully. The backslash at the end of the first line is the **line continuation** marker.

```
ls -tr *.jpg \
| for n in ‘seq -w 1 $count‘
  do
    read file
    mv -i $file $n-$file
  done
```

Take a look at the result using `ls`.

```
ls *.jpg | less
```

You should see all the photos, each prepended with a sequence number. This now has the effect of the name order and time order being the same.

```
ls -tr *.jpg | less
```

You can check the two listings are the same using the `sum` command as follows.

```
ls *.jpg | sum
ls -tr *.jpg | sum
```

STOP AND CHECK IT!	Please get a teaching assistant to check what you did wrong, if the two sums are not the same.

[1] `mv` is short for move. Renaming a file is a special case of moving it.

7.3 Building the website

You will want to have a **directory** in which the thumbnails will be stored. Call this THUMBS.

```
mkdir THUMBS
```

To do the actual building, you will use an already prepared **shell script**. Take a *brief* look at it. (You can study it in more detail later if you wish.)

```
less /opt/info/courses/LinuxWorkshop/make-thumbs
```

Now take a copy of it to your bin **directory**. The tilde character ~ is a shorthand for your **home directory**.

```
cp -ai /opt/info/courses/LinuxWorkshop/make-thumbs ~/bin
```

Now you're ready to run the script.

```
make-thumbs
```

This may take a while. It is creating a thumbnail for each photo, and **html** pages which link them all together, and an index.html page for the top level.

When it is finished you can take a look at what it has done.

```
ls | grep -v .jpg
ls THUMBS
```

Finally, you can browse the result.

```
firefox ‘pwd‘/index.html &
```

7.4 Life is never quite that simple. . .

When I asked my daughter to arrange for me to obtain three sets of photographs of her friend's 18th birthday party, I was not really expecting the resulting camera times to be exactly synchronised. So I wasn't surprised that I needed to correct the file dates afterwards. It was fairly easy to find the inaccuracies by comparing the times of photos of the same moment taken on different cameras.

Fixing the problem did not involve a lot of work. For example, this is how I set the file dates of all the photos in one directory to be 54 minutes and 7 seconds later than the time recorded in the corresponding photograph (don't type it).

```
for i in *.jpg
do
  time=‘jhead $i | grep Date/Time|cut -f2- -d: | sed "s,:06:,/06/,g"‘
  time=‘date -d "$time 54 min 7 sec"‘; touch -d "$time" $i;
done
```

Index