UNIX AND X

by

Frank H. King

An Introduction to Unix and X Windows using Linux

UNIVERSITY OF CAMBRIDGE
COMPUTER LABORATORY

JAVA EDITION
JULY 2007

An Introduction to Unix and X — Part I

At first mention, any technical term in this document is written in italics. The term *Public Workstation Facility* (PWF) is used in Cambridge to refer loosely to a Personal Computer (PC) together with its connection to a *network*. This document explains how to use a variant of the Unix *operating system* called *Linux* which may be implemented on a PC and which has, in particular, been implemented on the PWFs in Cambridge. These PWFs also support the Windows XP operating system.

1.1 Assumptions

- It is assumed that you have a PWF account and know your user name and password. Computing Service Reception can reset forgotten passwords.

1.2 Power

- It is local policy that the computers are *kept permanently switched on*. This saves time when you first sit down and also means that the state of the machines can be monitored remotely. An automatic power-save feature ensures that very little electricity is used when the computer is left unattended.

1.3 Two Operating Systems

- When you first sit at a free workstation the power-save feature may mean the screen is completely blank. If this is the case, move the mouse slightly. After 8 to 10 seconds this should result in one of three possibilities:
  1. A PWF Login *window* dominated by the word Novell.
  2. A large *message window* which refers to the Computer Misuse Act of 1990. If you see this, click OK and the screen will change to the third possibility...
  3. A PWF Linux login window dominated by the word Username.
- The PWFs support both the Microsoft Windows operating system (when you see the first possibility) and the Linux operating system (when you see the second or third). If you can see a PWF Login window, your PC is running Microsoft Windows and you will have to work through the next section to install PWF Linux. If you can see the PWF Linux login window, you should skip to section 1.5 now.

1.4 Installing PWF Linux

- To install PWF Linux it is first necessary to shut the PC down and start it afresh via the Shutdown facility. The PWF Login window incorporates the invitation to press:

  Ctrl+Alt+Delete to Login or Shutdown
- Key in Ctrl+Alt+Delete and the result on the screen should be the Microsoft Windows variant of the message window which refers to the Computer Misuse Act 1990.
- Click OK and the screen should soon display a new window with the *title* Novell Client at the top. You must not login! Instead, click Shutdown, one of three buttons at the bottom of the window.

- A new window offers various options including Shutdown and Restart. Click Shutdown and Restart and then click OK.
- Be patient! Shutting down may take the best part of a minute and you must watch carefully for the request:

Please select the operating system to start:

- Press the ↓ key to *highlight* the second choice, PWF Linux, and then press RETURN.
- Be even more patient! It can take *nearly four minutes* to install PWF Linux. During this time you may see a message which begins ATTENTION NO SIGNAL. Take no action! Eventually, you are presented with the Linux variant of the message window which refers to the Computer Misuse Act 1990.

1.5 Logging into PWF Linux

- The Linux variant of the message window which refers to the Computer Misuse Act 1990 and other more recent legislation fills most of the screen.
- This form of the message window is the principal sign that the PC is running Linux. The information is important and should be read carefully. The window has an OK button in the bottom right-hand corner. Click the OK button.
- The PWF Linux login window should now appear. In it there is an obvious rectangular box labelled Username.
- Key in your user name and press RETURN. This should result in another box labelled Password.
- Key in your password and press RETURN again.
- Various items appear in turn (and some change their appearance) but eventually the screen settles down to showing a number of *icons* on the left, a *task bar* at the bottom and an obvious rectangular window in the middle. All these features are displayed in the *root window*, the Unix equivalent of the *desktop* (in Microsoft terminology).
- You have successfully logged into PWF Linux and have started an *X-session*. The central window has PWF Message of the Day in its *title bar* and is an example of an *X-window*. You normally don't need to take too much notice of the messages.
- The task bar at the bottom of the root window includes a list of all the windows in the root window. There is only one at the moment, the PWF Message of the Day window.
- Move the mouse around and watch the appearance of the mouse *pointer*. . .
- For the most part, the pointer has the form of a roughly upwardly-pointing arrow. When it is on the very edge of the PWF Message of the Day window it changes its appearance to one of the many other forms that it can take.

1.6 A Terminal Window

- With the icons, task bar and window, anyone familiar with Microsoft Windows might be tempted to feel at home. Linux can indeed be used rather like Microsoft Windows but one purpose of this course is to gain familiarity with low-level features of Unix so the method of working will generally be rather different.

- Move the pointer anywhere in the root window (the desktop) and press *and hold down* the *right* mouse button.
- This brings up a *menu* which contains various *entries* including Open Terminal (which may be the fourth entry).
- Move the pointer to the entry Open Terminal and release the right button.
- A new window appears. Its title bar contains a string in two parts, one on each side of an @-sign. [If your identifier is c201 and you are sitting at a workstation whose identifier is pccl504, the string will be c201@pccl504].
- Notice that the workstation identifier also appears as a new entry in the task bar at the bottom of the screen. The new window is the first example of a Unix *shell*.
- The shell is characterised by a Unix *prompt* ending in a >-sign. This prompt may also consist of your identifier, an @-sign and then the identifier of your workstation. Shell windows are used for giving Unix *commands*.

1.7 Focusing

- The *text cursor* after the prompt is in the form of a solid black flashing block and the title bar is highlighted. These are indications that the window is *in focus*. It is the active window and can be used for giving commands.
- Click anywhere in the root window. The text cursor now turns into the form of a hollow rectangle and the title bar loses its highlighting. These are indications that the Terminal window is *out of focus*. Try typing a word. Nothing happens.
- Repeatedly click in the Terminal window and in the root window in turn so that the Terminal window goes in and out of focus. Notice how the entry in the task bar changes appearance, another indication of whether or not the associated window is in focus.

1.8 Two Unix commands

- After making sure that the Terminal window is in focus (the text cursor a solid block), key in the following *Unix command* and press RETURN:

  ```
  date
  ```

- Unix is case sensitive, so it is important to type `date` and not `DATE` or `Date`.
- Any typing mistake which is noticed immediately can be corrected using BACKSPACE. It is almost always necessary to press RETURN after keying in a command. In the present case, the date and time are given.
- Try another command:

  ```
  who
  ```

- The result is a list of people who are using your PC at the moment. You are likely to be the only user!

1.9 The passwd command

- Should you wish to change your PWF password, you can use the `passwd` command:

  ```
  passwd
  ```

- The system invites you to take part in three lines of dialogue, quoting your existing password, your new password and your new password again.

1.10 The Main Menu

- An important feature is the *Main menu* which is used in much the same way as the Start menu in Microsoft Windows.
- Open the Main menu by clicking Computer at the left-hand end of the task bar at the bottom of the screen.
- Notice three buttons Applications, Documents and Places. Click Applications to ensure that a list of Favorite Applications is presented. This list will include Firefox which is the recommended *Web Browser* when using Linux.
- Click somewhere in the root window and the Main window will disappear.

1.11 Finishing

- The approved way of finishing is to use the Logout entry in the Main menu. Click Computer to open the Main menu again and then click Logout on the right.
- You will be asked Are you sure you want to log out? Click OK. A warning message will proabably appear, in which case click Close in the message window.
- The screen goes blank for a while and then reverts to showing the message about the Computer Misuse Act 1990. It is now in order to walk away from the PC if you feel like taking a break.

1.12 Booting back into Windows XP

- If you wish to use Windows XP you will have to reboot the workstation. This is *not* necessary now and you can skip to the next page. Should you, or another user, wish to use this PC for a Windows XP session then you should proceed as follows…
- Click OK at the bottom right-hand corner of the message window and then click Restart in the bar at the bottom of the root window.
- This brings up a qusetion Are you sure you want to reboot the computer?. Click Restart. There is a prolonged shut-down sequence after which the system will automatically boot up into Windows.

An Introduction to Unix and X — Part II

Only three Unix commands were described in Part I. Here in Part II more of the numerous facilities available in PWF Linux will be introduced.

2.1 Logging in

- If you are continuing immediately from Part I your PC should still be showing the Linux variant of the message window which refers to the Computer Misuse Act 1990. Click OK and then log in as before.
- If you have been away some time, or have decided to use a different PC, you may be presented with the Ctrl+Alt+Delete invitation. You will have to install PWF Linux as described in Part I and then log in as before.

2.2 Dragging to Move a Window and Change its Shape

- It is easy to reposition a window and change its shape, at least to the extent of changing its height or width.
- Move the pointer to the middle of the title bar of the Message of the Day window and then *drag* (by holding down the left mouse button) the window to a new position.
- Move the pointer to the right-hand edge of the window. Check that the pointer takes the form of an arrow pointing at a vertical bar. Careful positioning is essential.
- Drag to the right. The window will get wider. Drag to the left. The window will get narrower.
- Experiment with other edges of the window and also try dragging a corner when two sides will move position.

2.3 Two Buttons

- Notice the buttons in the title bar of the Message of the Day window. Point at (but don't click) ☐ on the left. After a second or so, a *tip* explains that this is the Window Menu button. Point at (but don't click) ☒ on the right. This time the tip explains that this is the Close Window button.

2.4 Closing a window

- The Message of the Day window is not going to be of any further use so it can be closed. Click ☐ to bring up the Window Menu and *notice but don't use* the Close entry. Clicking this would be one way to close the window but there is a quicker way…
- Click the Close Window button ☒ and, without using any menu, the window vanishes. It cannot be brought back again. The associated entry in the task bar disappears too. Strictly speaking, the only window left now is the root window.

2.5 A shell

- Using the Open Terminal entry in the menu which pops up when you right-click in the root window, create a new Terminal window or shell. Note the familiar string in the title bar and the associated entry in the task bar.
- Ensure that the Terminal window is in focus and give the `date` command:

```
date
```

2.6 Iconifying and deiconifying

- The Terminal window has three buttons at the right-hand end of its title bar. By pointing at each in turn you can see that they are the Minimise Window button, the Maximise Window button and the already-used Close Window button.
- Click the Minimise Window button and the window, with its title bar, disappears. It hasn't been closed though. The associated entry in the task bar does *not* disappear but it is in square brackets. This action is sometimes called *iconifying* a window.
- Click the associated entry in the task bar and the window reappears. The square brackets disappear. This action is sometimes called *deiconifying* a window.
- Another way to iconify a window is to click the associated entry in the task bar. Click this entry now. The window is iconified and the entry in the task bar acquires square brackets. Click the square-bracketed entry. The window is deiconified.

2.7 A second shell

- You can have several windows open at once. It is easy to create a second shell, a second Terminal window, via a right mouse click in the root window but it is instructive to use another approach...
- Being very careful to note the & give the following command from the Terminal window:

```
xterm &
```

- A new window, called an `xterm` window, appears and it has the familiar string in the title bar and an associated entry in the task bar. There is the usual Unix prompt ending in a >-sign. This is clearly another shell. Ensure that the second window is clear of the first. Move the new window and resize the first if necessary.
- You should notice in the Terminal window that, following the `xterm &` command, there are two numbers. The second (which probably has four or five digits) is a Unix *process number*. It is usually unnecessary to take much notice of it but it is important to check that such a number has been quoted.
- Focus on the two windows in turn and note how the highlighting of the title bars and edges changes and the highlighting of the associated entries in the task bar change too. Click in the root window and note that both shell windows go out of focus.
- The xterm window can, of course, be used for giving Unix commands. Ensure that it is in focus and give the `date` command again:

```
date
```

2.8 Raising and lowering windows

- It doesn't take long for there to be so many windows that there isn't enough space for all of them and they start to overlap. Move the newer `xterm` window (by dragging its title bar) so that it overlaps about half the older Terminal window.
- Focus on the two windows in turn. Whichever window is in focus will be on top. It is said to be *raised*. The other window is *lowered*. Only one window can be in focus at a time and a command can be keyed in only if a shell is in focus.

2.9 Removing the new xterm window

- Two ways of closing a window have already been noted. You can click ☐ and choose the Close entry or you can click ☒ instead. It is instructive to see a third approach...
- Focus on the xterm window and key in Ctrl-d. The xterm window disappears and its associated entry in the task bar disappears too.
- Focus on the Terminal window and press RETURN. A somewhat cryptic message which includes the word `Done` appears. This is simply to note that you have abandoned the xterm window.

2.10 More about the & — background and foreground tasks

- It is instructive to see what happens if you forget the `&` when giving the command `xterm &` to create an xterm window. Focus on the Terminal window and, this time, deliberately omit the `&` when giving the command:

  ```
  xterm
  ```

- A new xterm window is created but no Unix process number or prompt appears in the Terminal window. Move the new xterm window clear if necessary.
- Ensure that the xterm window is in focus and give the `date` command:

  ```
  date
  ```

- There should be no problems. Now focus on the Terminal window and give the `date` command again there:

  ```
  date
  ```

- This time nothing happens. When `&` is used to create a new xterm window, the system arranges for this new window to be run as a *background task*. This means one can focus on the old window and give Unix commands *even though the new window continues in being*. Without the `&`, the new window is a *foreground task* and no useful work can be carried out in the old window.
- Now remove the xterm window by focusing on it and keying in Ctrl-d. When the window goes, the results of the `date` command at last appear in the old window.

2.11 Bigger writing in xterm windows

- The writing in the xterm window has been the same size as that in the Terminal window. To make the writing bigger, first focus on the Terminal window and then incorporate the items `-fn 9x15bold` between the `xterm` and the `&` when giving the `xterm` command (there is a space after `-fn` and *don't forget the* `&`):

```
xterm -fn 9x15bold &
```

- Unix keywords are usually introduced with a '`-`'. Here `-fn` stands for *fount* (English spelling!) and `9x15bold` is the name of one of many founts which are available. The writing in the new xterm window is somewhat easier to read.
- Give the `date` command in the new window:

```
date
```

2.12 Getting going — iconifying the original xterm window

- Different users have different ideas about how best to set up windows at the beginning of a session. After some experience everyone develops an individual style. For the moment, let's assume that you prefer the fount in the xterm window and wish to work from that.
- The Terminal window is unlikely to be used again. You could remove it but it is better practice to iconify it. Iconify the Terminal window now.
- Move the xterm window to the top left-hand corner of the root window.

2.13 The ls command

- The `ls` command is used to List and Sort one's *files*. After making sure that the xterm window is in focus, give this command:

```
ls
```

- At present, you have probably not set up any Unix files explicitly but any entries relating to Microsoft Windows will be listed. Leave these entries alone.

2.14 Setting up a text file — Emacs

- The next task is to set up a Unix file whose file name is `jobletter798` and whose contents are to be a rather silly letter!
- A standard way of setting up a file is to use the Emacs text editor. One way to invoke this editor is to give the command `emacs &` (*don't forget the* `&`):

```
emacs &
```

- A new window appears which has emacs in its title bar and there is an appropriate new entry in the task bar.
- Move the emacs window to the top right-hand corner of the root window.

2.15 Emacs commands

- The main part of the emacs window incorporates some getting-started information.
- Click anywhere in the main part of the window and some instructions will appear which explain how to create a new file.

- In all Emacs documentation the abbreviation `C-` is used for Ctrl- and it is easy to see that the cryptic `C-x C-f` stands for Ctrl-x followed by Ctrl-f. When Emacs is under consideration, the abbreviation `C-` will be used for Ctrl- from now on.
- Near the bottom of the emacs window is a highlighted *mode line* which indicates that the editor is in `Lisp Interaction` mode. This may one day be of significance but ignore it for the moment.
- The mode line begins `-u:-- *scratch*` which implies that you haven't yet specified a file name. Temporary arrangements have been made in consequence.
- The word `All` simply indicates that the entire text fits in the window. The displayed text isn't very long.
- Emacs can be menu driven or command driven. Enthusiasts prefer the latter and this document reflects that enthusiasm!
- The keystrokes `C-x C-f` refer to the Emacs `find-file` command. The `x` refers to an extended list of Emacs commands and the `f` stands for `find-file`. The idea now is to *attempt* to find a file called `jobletter798` and hope to fail!
- Note that if you succeed in finding the file it would be an *old* file but you want to create a *new* file. You would therefore want to choose a different name. By adopting the procedure just described you are unlikely to overwrite an important file by mistake.
- With the emacs window still in focus, key in `C-x` (remember this means Ctrl-x) and notice that after a couple of seconds `C-x-` appears in the *echo line*, the line below the mode line. This is a reminder of what you have just keyed in.
- It is useful to know what to do if you key in `C-x` (or any other Emacs command) accidentally. Simply key in `C-g`. Try that now. The `C-x-` disappears. Now key in:

  ```
  C-x C-f
  ```
- A message which begins `Find file` appears in the echo line. The cursor jumps to the end of that line where you should key in:

  ```
  jobletter798
  ```
- Press RETURN. The `*scratch*` in the mode line changes to `jobletter798` and `Lisp Interaction` changes to `Fundamental`, indicating that Emacs is expecting plain text to be keyed in. The echo line confirms that this is a `(New file)` indicating that the file name has not already been used. The main part of the window clears and the cursor jumps to the top left-hand corner.
- Key in the following, *including the two spelling mistakes*, and *press* RETURN *at the end of every line*:

```
Dear sir,

I am sorry I did not turn up for the job interview last
week, but there was a good horror film on the television
and I did not want to miss it.
```

```
I am willing to come for a new interview on Wednesday
afternoon next week, though I must leave by 4:30 to
go to the football game.

I am an expurt in UNIX, particularly the game playing
programs, so I would be a grate asset to your company.

    yours sincerely,
```

- Immediately you begin typing, the `-u:--` at the beginning of the mode line changes to `-u:**` where the asterisks indicate that the text has changed.
- One imagines that this is a job application letter written by someone who wants to work as a Unix programmer. There are two deliberate mistakes and you may have made some yourself!
- As usual, any typing mistake which is noticed immediately can be corrected by using BACKSPACE. To correct a typing mistake made on a previous line, use the *text cursor keys* (those marked with ↑, ←, ↓ and →) to move the cursor to just *after* the position of the error and delete from there before retyping.
- Try some experiments with the cursor keys. See what happens when the ← or → key is used to move the cursor off the left- or right-hand end of a line. Notice that if one of these keys (or indeed *any* key) is held down for more than a fraction of a second its effect is repeated, a feature known as *auto-repeat*. Double-check the text after finishing the experiments. *Leave in the two deliberate mistakes.*
- The next step is to save this letter as the contents of the file `jobletter798` (this applicant has had numerous previous unsuccessful attempts).
- Saving requires another extended Emacs command, the `save-buffer` command. The associated keystrokes are `C-x C-s` so key in:

  ```
  C-x C-s
  ```
- The echo line displays a message which begins `Wrote` and which ends with the file name `jobletter798` to indicate that the file has been saved.
- The `-u:**` in the mode line changes back to `-u:--` to signify that the displayed text is the same as that in the file.

2.16 The ls command again

- Focus on the `xterm` window. List and Sort your files again:

  ```
  ls
  ```
- This time the file name `jobletter798` should be listed.

2.17 Aside — bigger writing in Emacs

- The default fount in the `emacs` window suits most users but, if you prefer larger writing, you can always set up an `emacs` window from an `xterm` window and supply the fount information explicitly. From the `xterm` window, and noting the *essential* `&`, give the command:

```
emacs -fn 9x15bold &
```

- A new `emacs` window appears with the larger fount.

2.18 Leaving Emacs — I

- There are several ways of leaving Emacs. One way is to click ☒ but this is *not* generally a good idea because the exit from Emacs is precipitate and you are not given a chance to save any files that have inadvertently been left open.
- The new `emacs` window has not yet been used so there is no harm in clicking ☒ this time. Do that now. The new `emacs` window disappears.

2.19 Leaving Emacs — II

- The approved way of leaving Emacs is to use an appropriate Emacs command. First focus on the (original) `emacs` window.
- The appropriate Emacs command is `save-buffers-kill-emacs` and the associated keystrokes are `C-x C-c` and it is probably easiest just to think of the `c` as meaning cancel Emacs. Key in:

```
C-x C-c
```

- A message `(No files need saving)` appears momentarily in the echo line and the window disappears.

2.20 The more command

- From the `xterm` window, give the command:

```
more jobletter798
```

- The `more` command is used for inspecting the contents of a file. Here, the contents of the `jobletter798` file should appear.

2.21 The spell command

- Carefully noting the `<` character, give the command:

```
spell <jobletter798
```

- The `spell` command gives a list of spelling mistakes in the file heralded by `<` which may be pronounced ‘from’. The list should include `expurt` but not `grate`.

2.22 Correcting the mistakes

- The obvious way to correct mistakes is to use Emacs. Give the command `emacs &` to enter Emacs again.
- Move the `emacs` window to the top right-hand corner of the root window.
- To open an existing file, use the Emacs `find-file` command exactly as it was used previously. The difference is that you are not proposing to set up a new file this time so you *want* to find the file.

- Key in `C-x C-f` again.
- The message which begins `Find file` appears in the echo line and the cursor jumps to the end of that line where you should again key in:

  `jobletter798`

- Press RETURN. The letter reappears in the main part of the emacs window and the `*scratch*` again changes to `jobletter798` but there is no mention of `New file` in the echo line.
- Other things to notice are that the cursor jumps to the beginning of the letter and there are no asterisks in the mode line. Again `Lisp Interaction` changes to `Fundamental` indicating that the file contains ordinary text.
- Correct the spelling mistakes. The asterisks reappear. You should next save the revised version in exactly the way you saved the earlier version by using the Emacs save-buffer command, so key in:

  `C-x C-s`

- The echo line confirms that Emacs duly `Wrote` the text to the file.
- Leaving Emacs before giving the `spell` command and then immediately re-entering Emacs was fairly pointless! The reason for doing so here is to show how to open an existing file when using Emacs.
- It is much more common to set up an emacs window at the beginning of a session and keep it until the end. If you think you are unlikely to be doing any editing for a while, you should follow normal practice and simply iconify the emacs window. It will then be instantly available should you wish to use it later. Click ▁ to iconify Emacs now.

2.23 Some Emacs commands

- The following summary lists the Emacs commands that have been mentioned so far:

`C-g`	`keyboard-quit`	Go away, I didn't mean it
`C-x C-f`	`find-file`	Find (open) an existing file
`C-x C-s`	`save-buffer`	Save (update) an existing file
`C-x C-c`	`save-buffers-kill-emacs`	Cancel (leave) Emacs

  [Remember that in Emacs documentation the abbreviation `C-` is used for Ctrl-]

- When used in a simple way, Emacs can be menu driven. With the exception of `C-g`, the commands listed in the summary can be found as entries in the `Files` menu. Against each entry, the associated keystrokes are shown.
- Whether to use menus or keystrokes is largely a matter of taste but note that by no means all Emacs commands can be found as entries in menus.

2.24 The ls command once more

- Focus on the xterm window and give the `ls` command again:

  `ls`

- A new file name is listed. In addition to `jobletter798` there is `jobletter798~`. This new file contains a *back-up copy* of the previous version of the file. Emacs automatically backs up files in this way to protect you against possible accidents!

2.25 The ls command — dot files

- You might think that, at the moment, you have two files concerned with the job application and no other Unix files at all. Actually there are a good many more configuration files which have been set up automatically. You may eventually learn how to modify these files but don't do this until you really understand what you are doing!

- For the moment it is of interest merely to see where these files are and to discover why the `ls` command hasn't revealed them. These files all begin with a dot and ordinarily the `ls` command doesn't list such files. If `ls` is followed by `-a` (`a` is for `a`ll) then the dot files *will* be listed. Give this modified command now:

```
ls -a
```

- A fair number of dot files are now listed in addition to the `jobletter` files you already know about. One dot file which doesn't exist just yet but which you may find a need to set up before long is `.emacs` which is a file for configuring Emacs to your own requirements.

2.26 The cp command

- It is unnecessary to know much about the dot files but there is just one of them which is missing at the moment but will eventually be needed if you are following this course. This is the file `.bash_profile` which should be copied into your file space now.

- It is safest to copy a definitive version from a special *file directory* `$CLTEACH/fhk1` (in which `$CLTEACH` is a Unix *environment variable* which provides a short-cut to the special directory) and you should use the following `cp` command (`cp` is for copy):

```
cp $CLTEACH/fhk1/.bash_profile .
```

- It is important to notice the dot and the *underscore* character in `.bash_profile` and that the command ends with a space followed by a lone dot. This lone dot refers to your file directory (in fact to your *home directory*) and the command as a whole is copying the `.bash_profile` file from the special directory to your home directory.

- If there is any sign of an error message, give the command again.

2.27 Final Check

- It is always a good idea to check your files before finishing. On this occasion you ought to check that that the file `.bash_profile` really has been copied, so give the modified `ls` command again:

```
ls -a
```

2.28 Finishing

- It is essential to finish in the approved manner...
- Choose Logout in the Main menu and click OK and, if a warning message appears, click Close. This will clear away all your windows including any which are iconified. When the Linux variant of the message window which refers to the Computer Misuse Act 1990 appears, it is safe to walk away from the PC. Do not switch off the PC or the monitor.

An Introduction to Unix and X — Part III

Here in Part III, further features of PWF Linux will be presented and a couple of very simple Java applications will be introduced.

3.1 Getting going

- Assuming a cold start, begin this and subsequent sessions by undertaking the following steps, which may be freely adapted as you develop your own style of working:
  1. Check that PWF Linux is running. If not, Shutdown and Restart, selecting PWF Linux at the appropriate moment.
  2. Log in to PWF Linux.
  3. Close the PWF Message of the Day window.
  7. Bring up a Terminal window and move it to the top left-hand corner of the root window if it does not appear there by default.
  8. Bring up an emacs window and move it to the top right-hand corner of the root window.
- If there is no immediate intention to use Emacs, the emacs window should be iconified. This is the present case, so iconify the emacs window.
- The only open window of consequence should be the Terminal window. Check that this and the iconified emacs window are noted in the task bar.

3.2 The cat command, standard input, standard output

- The `cat` command (short for 'concatenate') has many uses. In particular, there is a way of using it to copy the contents of one file to another file. Ensure that the Terminal window is in focus and then, *carefully noting the* `<` *character and the* `>` *character*, give the command:

  ```
  cat <jobletter798 >jobletter799
  ```

- Like many commands, the `cat` command requires *input* which comes *from* somewhere and produces *output* which goes *to* somewhere else. The characters `<` and `>` may be pronounced 'from' and 'to' respectively. In the present `cat` command:

  input is taken *from* the file `jobletter798` and
  output is sent *to* the file `jobletter799`

- Both input and output have been specified explicitly. If no explicit output is specified for a command which produces output then results are sent to the *standard output* which simply means they appear in the shell immediately after the command. Give the command:

  ```
  cat <jobletter799
  ```

- Here, input is from the file `jobletter799` but output is to the standard output. The contents of the new file `jobletter799` appear in the Terminal window and are a direct copy of the contents of `jobletter798`.

- If no explicit input is specified for a command which requires input then input is taken from the *standard input*, meaning input from the keyboard. Give the command:

```
cat >jobletter800
```

- This time the system does not respond with a prompt; it is waiting for you to type the contents of the file. Key in:

```
Dear sir,
Blah!  Blah!  Blah!
```

- Using the `cat` command in this way is an alternative to using Emacs for setting up a text file. It is normally better to use Emacs because mistakes can be corrected more easily. With the `cat` command, the cursor keys cannot be used.
- Key in `Ctrl-d` to terminate standard input. [You are not in Emacs so the abbreviation `C-d` is inappropriate but you may sometimes see `^d` used for `Ctrl-d`.] The `Ctrl-d` must be at the beginning of a line (just following RETURN). The system should reply with a prompt.
- It is here that a minor embarrassment could occur. If `Ctrl-d` is accidentally keyed in a second time, the window disappears. It can readily be recreated so no real harm is done but, on some versions of Unix, closing the principal shell precipitately logs you out. Accordingly, you should learn to take great care when keying in `Ctrl-d` and make sure that you key it in just once!
- Having noted that, by default, standard input is used for input and standard output is used for output, what happens if neither input nor output is specified? Give the `cat` command without any *arguments* (items following the command name):

```
cat
```

- The system is again expecting standard input (from the keyboard) so type some text:

```
The quick brown fox
```

- When RETURN is pressed, the system sends the line to the standard output (the window):

```
The quick brown fox
```

- The system doesn't give a prompt; it is waiting for more standard input. Type another line:

```
jumps over the lazy dog.
```

- The system sends the line to the standard output as before:

```
jumps over the lazy dog.
```

- Although it illustrates standard input and standard output, this exercise is not very useful! Key in `Ctrl-d` to quit from the `cat` command.
- So far, `<` has been used to indicate input *from* a file. This `<` can often be left out. Give the command:

```
cat jobletter799
```

- This works just as well as `cat <jobletter799` did earlier.

3.3 Local line editing

- If after a prompt you press the ↑ key then, instead of the cursor moving up a line, the result is that the most recent command is reproduced. Try pressing the ↑ key now; the result is:

  ```
  cat jobletter799
  ```

- The cursor hovers at the end of the line. Use BACKSPACE to delete just the `799` and then key in `800` as a substitute. Now press RETURN, thereby giving the command:

  ```
  cat jobletter800
  ```

- It is easy to make a more substantial change to the most recent command. First, press the ↑ key again, resulting in:

  ```
  cat jobletter800
  ```

- Now use the ← key and BACKSPACE (these are two different keys) to delete `cat` and then key in `spell` and `<` to give the command:

  ```
  spell <jobletter800
  ```

- Using the cursor keys and BACKSPACE to make changes to the line being typed is sometimes called *local line editing*.

3.4 History

- It is not just the most recent command which can be retrieved for editing. Try pressing the ↑ key six times *slowly*, pausing after each press to see the result. The six most recently given commands appear. Any of these could be edited if desired.
- Now press the ↓ key six times *slowly*. This brings you back to the present in stages. Using the ↑ and ↓ keys in this way is said to reveal *history*.

3.5 Geometry

- Give the following variant of the `xterm` command:

  ```
  xterm -fn 9x15bold -geometry 80x28-0-0 &
  ```

- The `-geometry` item specifies four aspects of the new window:

  `80` is the width of the window measured in characters.
  `28` is the height of the window in lines.
  `-0` places the window on the right-hand side of the root window.
  `-0` places the window at the bottom of the root window.

- In summary, the `80x28` specifies the width and height of the window and the `-0-0` causes the window to be placed in the bottom right-hand corner of the root window.
- Note that `+0+0` would place the window in the top left-hand corner; `-0+0` would place the window in the top right-hand corner and `+0-0` would place the window in the bottom left-hand corner.
- There are other variations on the `xterm` command which are worth exploring so this first attempt will be abandoned. Ensure that the latest window is in focus and key in `Ctrl-d` to remove it.

3.6 Colours

- From the Terminal window use local line editing to give the command:

```
xterm -fn 9x15bold -geometry 80x28-0-0 -fg white -bg blue &
```

- The `-fg` item specifies a foreground colour and the `-bg` item specifies a background colour. The result will soon be clear. Most of the common colours are available. Try some further experiments. Conclude by removing all xterm windows but keep the Terminal window in the top left-hand corner.

3.7 Java Programs — Introduction

- Java is an important programming language which is commonly used in Computer Science projects. Java is designed to be used across the Internet and is also designed for industrial-scale software projects.
- Java programs come in two forms, *applications* and *applets*. The latter are accessed via Web browsers or the `appletviewer` command and will not be discussed here. The two programs introduced in this document are both simple examples of Java applications.

3.8 A Java application — the source code

- The file name for any Java *source code* must be in two parts separated by a dot. In the first illustration, the name `ComeIn.java` will be used. Unix and Java are both *case sensitive* so it will be important to key in `ComeIn.java` and not `comein.java` or `Comein.Java` and so on.
- The part of the file name before the dot can be fairly freely chosen by the programmer but the remainder must be `.java` which is known as a file name *extension*.
- Before going any further, deiconify the emacs window. Ensure that the emacs window is in focus and embark on a new file by keying in:

```
C-x C-f
```

- The cursor jumps to the end of the echo line where (paying attention to the upper and lower cases of the letters) you should key in:

```
ComeIn.java
```

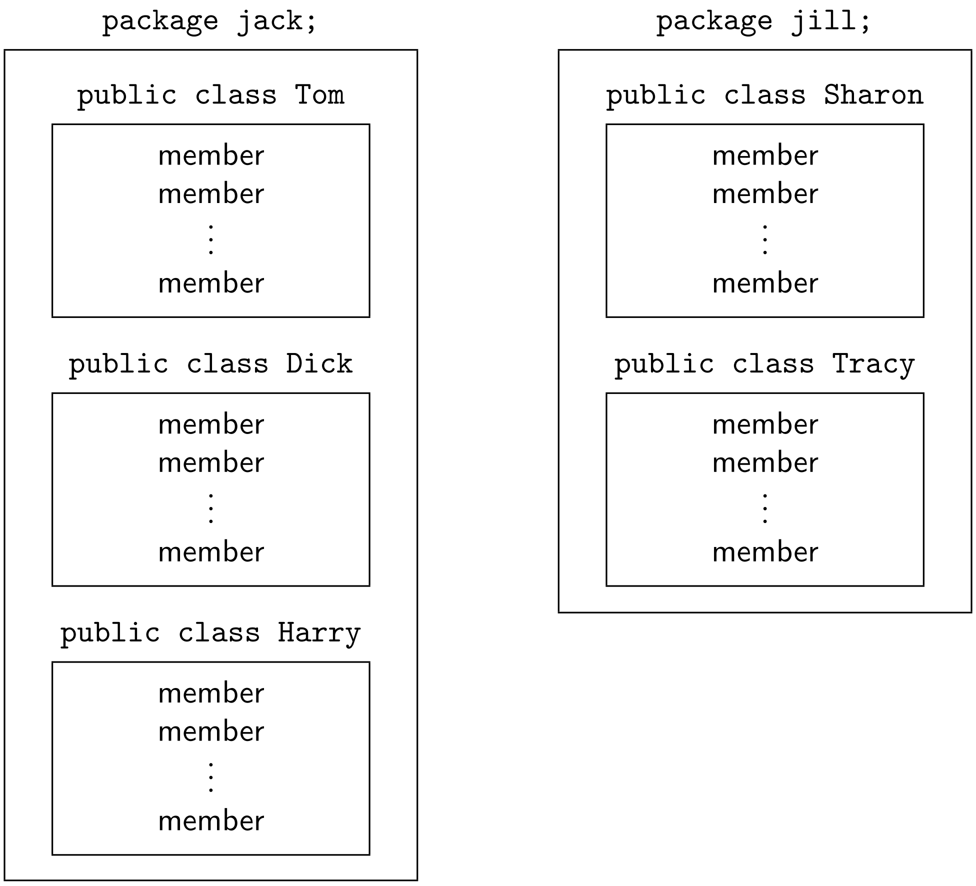
- Press RETURN. Check that the mode line confirms `ComeIn.java` and, also in the mode line, note that `Lisp Interaction` changes to `Java Abbrev`.
- Now key in the following source code of a frivolous Java application:

```
public class ComeIn
 { public static void main(String[] args)
    { System.out.printf("Come in number 57 please%n");
    }
 }
```

- Be careful to write `public` and not `PUBLIC` or `Public` which won't work. Again, write `ComeIn` and not `comein` or `Comein` and, most important, write `String` and not `string`. Notice the semi-colon, the pair of square brackets, the two pairs of round brackets and the two pairs of vertically aligned curly brackets.

3.9 Java Programs — Outline syntax

- Java programs can be thought of as being arranged in various levels of wrapping. A big program will be divided into several *packages* and in each package there will be several *classes* and each class will consist of several *members*.
- To get some idea of the shape or *syntax* of a big Java program consider a diagrammatic representation of a program which is divided into two packages, one of which contains three classes and the other two classes:



- The diagram follows a Java convention that package names are normally written in lower-case letters but class names begin with an upper-case letter. The diagram loosely corresponds to the associated file structure. Every class is in a separate file and the class files in any particular package are in the same directory.
- A class may contain several different kinds of member and two of particular importance are *data fields* and *methods*. The introductory application is very short and consists of only a single class, `ComeIn`, which itself contains just one member, the method `main` which will be discussed shortly. The second application will include a data field.
- It is of minor note that the introductory application is *not* headed by a `package` statement. In consequence, the application will be regarded as belonging to some anonymous package. (In the present context, the anonymous package is, in effect, your home directory and the collection of class files that will accumulate in this directory.)

3.10 Java Programs — Syntax of a class definition

- A Java class definition consists of a *class heading* (conventionally written on a line by itself) followed by a *class body* which is enclosed in curly brackets.
- In a simple case the outline syntax of a class definition is:

```
public class class-name
 { member
   member
     ⋮
 }
```

- The keyword `class` in the class heading is preceded by the `public` *modifier* and this particular modifier ensures that the class is *visible* from other packages.

3.11 Java Programs — Syntax of a method definition

- A method in a Java class consists of a *method heading* (also conventionally written on a line by itself) followed by a *method body* which is enclosed in curly brackets:

```
modifiers method-type method-name(arguments)
 { statement
   statement
       ⋮
 }
```

- The word method stems from the terminology of *object-oriented* programming, Java being an object-oriented programming language. Close parallels of methods are found in almost all programming languages where equivalent entities may be referred to as functions, procedures or subroutines.
- There is a requirement that exactly one class in any Java application includes the method `main` and, moreover, the `main` method must always be heralded by the two modifiers `public` and `static`.
- A full understanding of these two modifiers will come with experience but, roughly speaking, the `public` modifier makes the method visible from outside the class (in particular by the *run-time system* which supervises the running of the program) and the `static` modifier ensures that the method is available for immediate execution.
- A typical method concludes execution by delivering some value as a *result*, perhaps an integer number. The *type* of the result determines the type of the method itself. Some methods do not deliver a result and in such cases the type of the method is deemed `void`. This is the case with the `main` method.
- The *arguments* of a method are enclosed in round brackets. The method `main` has a single argument whose type is a `String` array (written `String[]`). The name of the argument is `args`. When a Java application is run, the method `main` is handed any items such as file names which may appear on the command line. Each such item has type `String` (a sequence of characters) and each will be assigned to an element of the `args` array.

3.12 Printing a String

- In general, a method body incorporates a sequence of *statements* enclosed in curly brackets. In the introductory application, there is only one statement in the body and this prints out the `String` constant `"Come in number 57 please%n"`.
- Printing is achieved via the somewhat ponderous syntax:

  `System.out.printf(`string-to-be-printed`);`

- Once again, a full understanding of this will come with experience and only a brief explanation is worthwhile now. There are numerous built-in packages in Java which contain facilities for standard operations like input–output. One such package is `java.lang` which includes the `System` class. This class has a data field `out` which, in turn, has access to the method `printf`.
- The three items `System`, `out` and `printf` are written as a continuous sequence with dots as separators. A `String` constant is enclosed between double-quotes and is handed to the `printf` method as an argument (which is why it is in round brackets). The statement as a whole must be terminated with a semi-colon.
- Note the `%n` at the end of the `String` constant. When the application is run, this `%n` generates an end-of-line character. You can try leaving it out later and you will see why it is needed.

3.13 Saving the Java application

- The rules for the file name the source code can now be clarified. The first part of the file name must exactly match the class-name and `.java` must be used as the file name extension. Clearly `ComeIn.java` is the appropriate file name here.
- The quickest way to give the Emacs `save-buffer` command is to use the keystrokes `C-x C-s` but it is instructive to give an Emacs command the slow way…
- In Emacs documentation `C-`something stands for Ctrl-something. Additionally there is `M-`something which stands for Meta-something. Few keyboards have a key which is actually marked Meta and on a PWF you should use Esc. [On some other workstations Alt may be the appropriate key.]
- Where Esc is used to key in `M-x`, you must press *and release* Esc before pressing `x`; thus Esc and `x` are pressed in turn and not together. [At terminals where Alt is appropriate, use Alt rather like Ctrl: hold the key down and then tap `x`.]
- Key in `M-x` by whichever means is appropriate. Note that `M-x` appears in the echo line. Strictly, this is the Emacs `execute-extended-command` command. It means that you can now key in the full name of *any* Emacs command. Key in:

  `save-buffer`

- Emacs is another system which is case sensitive so be careful what you key in.
- From the Terminal window give the command:

  `ls`

- You should see the file `ComeIn.java` listed alongside the `jobletter` files.

3.14 Compiling the Java application

- To be useful, a Java application must be *compiled* and then *run*.
- From the Terminal window give the command:

```
javac ComeIn.java
```

- Notice that the `.java` file name extension must be quoted.
- The command `javac` invokes version 1.5 of the Java *compiler*. If you are confronted by a complaint about an `invalid source release` you probably do not have a correct `.bash_profile` file. See page 13.
- If you made any mistakes when keying in the application the compiler will complain! You must correct the source code and try again.
- The compiled code of any Java class goes to a file whose name consists of the class name followed by a `.class` extension. Give the command:

```
ls
```

- Check that the file `ComeIn.class` is listed.

3.15 Running the Java application

- The Java compiler does not generate code which is ready for immediate linking, loading and running. Instead, the compiler generates *bytecode* for the Java Virtual Machine (JVM). The JVM can be regarded as an *interpreter* with a *run-time system.*
- It is the `java` command which invokes the Java interpreter. Key in:

```
java ComeIn
```

- Notice that the `.class` file name extension must *not* be quoted.
- The bytecode is duly interpreted and this will involve, at run-time, the dynamic loading of input–output facilities (and, in general, other facilities besides). Ordinary linking does not occur in Java.
- The output should be `Come in number 57 please` and this should be followed by a Unix prompt.

3.16 A second application

- The second illustration will be sufficiently like the first that the easiest way to proceed is to edit the source code of `ComeIn.java` in Emacs.
- Focus on the emacs window and modify the source code to the following:

```
public class ComeAgain
 { private static int n;

   public static void main(String[] args)
    { n = 57;
      System.out.printf("Come in number %d please%n", n);
    }
 }
```

- The application again consists of a single class, this time called `ComeAgain` so the first line will need changing. The new class incorporates a data field as well as a method.
- The `private static int n;` *declares* a data field whose name is `n` and whose type is `int` (a whole number). The `private` modifier is at the other extreme from `public` and restricts the visibility of the data field to within the class only. This is fine given that the only reference to `n` is from the method `main`. The `static` modifier is essential and ensures that the data field is accessible. The semi-colon terminator is mandatory.
- The `main` method includes the assignment statement `n = 57;` which gives a value to the data field. An assignment statement must also be terminated with a semi-colon.
- The `printf` method now has *two* arguments. The first is a string which incorporates `%d` where `57` was before. The string again concludes with `%n` and is enclosed in double-quotes but is now followed by a comma and the second argument, the data field `n`. Note that `%d` means that the value is interpreted as base-10 (`d` stands for `d`ecimal).
- The `%d` code in the `String` is associated with the `n` presented as a second argument. A second `%d` in the `String` would refer to a third argument and so on. Each `%d` is replaced by the value of the associated argument following the `String`.
- You should *not* use the Emacs `save-buffer` command to save this source code because that will overwrite the `ComeIn.java` file. You should use the Emacs `write-file` command instead. Its associated keystrokes are `C-x C-w` so key in:

  ```
  C-x C-w
  ```
- The message in the echo line begins `Write file` and, at the end of this line, key in:

  ```
  ComeAgain.java
  ```
- Focus on the Terminal window and compile the new application:

  ```
  javac ComeAgain.java
  ```
- Assuming there are no complaints, run the compiled code:

  ```
  java ComeAgain
  ```
- The output should be `Come in number 57 please` as before.
- This second application is somewhat contrived. Its principal purpose is to present a class which contains a data field and a method, the two most important kinds of class member. Typically, classes contain several data fields and several methods.

3.17 Layout

- Readers who have studied Java before, however cursorily, will have noticed that the layout used in the two examples is somewhat idiosyncratic.
- The layout employed in most text books follows a style adopted in the early days of C when only primitive editors were available, a poor precedent! Source code is much easier to understand if matching curly brackets are vertically aligned.
- In the illustrations above, each level of indentation involves shifting the code an extra three spaces to the right. Every curly bracket is centred with respect to a three-space gap. Whatever style you choose, note that good layout greatly improves readability.

3.18 Emacs buffers

- Focus again on the emacs window.
- At the moment Emacs is displaying the new source code. The source code of the first application has not been lost because it was saved in a different file. To retrieve the earlier code, key in:

  ```
  C-x C-f
  ```

- Then key in the file name `ComeIn.java` and press RETURN.
- The previous source code should appear.
- Although Emacs is now displaying just the old version, both files are open. The two lots of source code are in separate Emacs *buffers*. To see what buffers exist, give the `list-buffers` command. Key in:

  ```
  C-x C-b
  ```

- The emacs window splits into two half-size windows. The old source code is in the upper and a list of buffers is in the lower. The list includes `ComeIn.java` and `ComeAgain.java` amongst other entries.
- Note that the cursor is in the upper of the two windows. To get rid of the lower window, give the `delete-other-windows` command. This uses the digit-`1` key *without* Ctrl. Key in:

  ```
  C-x 1
  ```

- To bring the `ComeAgain.java` buffer back as the displayed buffer, give the `switch-to-buffer` command. This uses the `b` key, again *without* Ctrl. Key in:

  ```
  C-x b
  ```

- The message in the echo line should indicate that Emacs will, by default, switch to the `ComeAgain.java` buffer. If this *isn't* the default, you can key in `ComeAgain.java` explicitly. Press RETURN.
- The new source code should reappear in the emacs window and `ComeAgain.java` should appear in the mode line.

3.19 The cal command

- No use will be made of Emacs for a while so iconify the emacs window.
- Almost all commands that have appeared so far have produced results which have been sent to the standard output. The `java` command is no exception. Here is another example. From the Terminal window give the command:

  ```
  cal 1900
  ```

- The calendar for all 12 months of 1900 is displayed but there is a snag. The Terminal window is probably too small to accommodate the whole of the standard output and the first part of the calendar runs off the top of the window. There are various ways of dealing with this problem . . .

3.20 The scroll bar

- Along the right-hand edge of the Terminal window, there is a narrow column, the *scroll bar*, the lower part of which is highlighted by a vertical stripe called the *scroll box.* The height of the scroll box relative to the height of the scroll bar indicates the proportion of the text which is in the window compared with the amount which has run off the top but is still accessible.
- Move the mouse pointer to somewhere inside the scroll box and slowly drag the scroll box upwards. The text in the window moves downwards; this action is called *scrolling* the window. All the earlier months of 1900 can be brought back. When the mouse button is released, the text stays where it was scrolled to.
- Pressing RETURN results in an abrupt jump to the end of the text. Try this. Apart from a new prompt, the window reverts to the way it was before scrolling.

3.21 Sending output to a file

- From the Terminal window press RETURN and then give the following version of the `cal` command:

  ```
  cal 1900 >then
  ```

- The calendar for 1900 goes to the file `then`.
- This may not immediately seem to be helpful since if you inspect the contents of the file `then` using the `cat` command the calendar still won't fit:

  ```
  cat then
  ```

- Resorting to the scroll bar is unnecessary because the `more` command (this was first used in section 2.20) can be exploited. Using the `more` command, a file can be looked at one windowful at a time. Try:

  ```
  more then
  ```

- Only as many lines appear as will fit and there is a note of what percentage of the text this is. Press RETURN repeatedly to advance through the text one line at a time. To advance through the text one windowful at a time, repeatedly press the SPACE-BAR.

3.22 Pipes

- A *pipe* is a sequence of commands which are strung together so that the output of one command becomes the input of the next. The commands are separated by vertical bars (note that the vertical bar key is next to the Z key). An example of a pipe is:

  ```
  cal 1900 | more
  ```

- The output from the command to the left of the `|` becomes the input of the command to the right of the `|`. This example of a pipe is quicker than typing `cal 1900 >then` followed by `more then` which not only involves keying in two lines but introduces the file `then` which isn't needed when a pipe is used.
- The `more` command again causes the display to pause after a windowful but notice that no percentage is quoted because `more` doesn't know how much more there is to come. Press the SPACE-BAR to see the rest of the text.

- A second example of a pipe is:

```
cal 1900 | cat >then
```

- The output of the `cal` command is piped to the input of the `cat` command. The output of the `cat` command goes a new file `then`. Note that this is a silly example because it would have been better to use `cal 1900 >then` alone!

3.23 Highlighting and copying text

- Suppose you wish to set up a file containing just the first three months of 1900. Clearly you could use Emacs to edit the file `then` but it is quicker to use the *highlighting* feature of X.
- Give the `more` command again:

```
more then
```

- Just the first part of the file appears in the window. Now quit from the `more` command *without* inspecting the whole file: simply press `q` (for `quit`).
- Next use the `cat` command to embark on keying in a new file `jfm` (for January, February and March):

```
cat >jfm
```

- Press RETURN. Notice that the cursor is in position ready for you to key in the first line of the required text but don't key in anything. Instead proceed as follows...
- Move the mouse pointer to the beginning of the line on which `January` is written and slowly drag downwards as far as the blank line which separates the first three months from the second three. Dragging this way does not cause anything to move; instead, the *region* of text dragged across becomes reverse-highlighted (normally white characters on a black background).
- When the mouse button is released, the reverse-highlighting persists. Regions of text highlighted in this way can be copied to the place marked by the cursor and, just now, the cursor is at the bottom of the window immediately below `cat >jfm` waiting for standard input.
- To copy the highlighted text, simply press the *middle* button. The copy appears at the bottom of the window as standard input for the `cat` command. Key in Ctrl-`d` to terminate the input.
- To turn the highlighting off, click the mouse anywhere in the Terminal window.
- To check the effect of what you have just done, clear the window and inspect the contents of the new file:

```
clear
cat jfm
```

- Note that highlighted regions do not have to be large chunks of text. Single sentences or even single words can be highlighted and copied. Note, further, that text which is highlighted in one window can be copied to another window.

3.24 Highlighting rules

- To highlight a region of text:

  Drag across the region to be highlighted

- To copy a highlighted region of text to the position marked by the cursor:

  Press the Middle mouse button

3.25 Completion in Emacs

- Emacs has many hundreds of features and it is worth trying out just a few more in this introductory document.
- First, deiconify Emacs. The Java source code is still in the window. Bring back the `jobletter798` file instead. Begin with:

  `C-x C-f`

- This invokes the `find-file` command. The cursor jumps to the end of the echo line where you could key in `jobletter798` but it is unnecessary to key this in in full. *Without pressing* RETURN just key in the first letter:

  `j`

- Now press the SPACE-BAR. The window splits and all files which begin with `j` are listed in the bottom half; these are the possible *completions*. As well as `jfm` there are four file names which begin `jobletter`. Now just key in the single letter `o` and press the SPACE-BAR again:

  `o` SPACE-BAR

- Emacs now knows enough to complete as far as `jobletter` but it still doesn't know exactly what you want. You have to key in:

  `798`

- Press RETURN and the file's contents duly appear.

3.26 Clicking the mouse in Emacs

- Click the first `y` of `yours sincerely`. The cursor jumps to the position of the mouse pointer. This is an alternative to using the cursor keys.
- If you press BACKSPACE you will, of course, delete the character to the left of the `y`, probably a space. To delete the `y` without first moving the cursor, key in:

  `C-d`

- This is the `delete-char` command and deletes the character the cursor is on rather than the one to its left. Now key in an upper-case `Y` instead:

  `Y`

- In a similar way change the `s` of `sir` to an upper-case `S` (or make it lower case if it already is upper case!).

3.27 The Emacs undo command

- A mistake can be undone using the Emacs `undo` command. Emacs documentation gives `C-_` for this but notice that as well as the usual use of Ctrl you will probably need to use SHIFT for the *underscore* character. That said, key in `C-_` *four* times very slowly to see the effect:

```
C-_ C-_ C-_ C-_
```

- Not only does this negate the earlier changes but, assuming those were the *only* changes, the two asterisks in the mode line disappear to indicate that the text has not been changed since the file was opened.

3.28 Emacs kill and yank

- Click the `D` of `Dear` and key in:

```
C-k
```

- This is the `kill-line` command. The `Dear sir,` goes blank. To get rid of this blank line, give another:

```
C-k
```

- Killed text goes to the *kill-buffer* and can be brought back. Try:

```
C-y
```

- This is the `yank` command. Anything in the kill-buffer is yanked back.
- Kill and yank can be used as *cut and paste* in a word processor. Click the `I` at the beginning of the final paragraph and then give the `kill-line` command *five* times:

```
C-k C-k C-k C-k C-k
```

- The first four kill the two lines of the paragraph and the last kills the blank line before `yours sincerely`. Next click the `I` at the beginning of what has just become the final paragraph and then yank back the killed text:

```
C-y
```

3.29 Filling

- Click the `c` of `company`. Then, *including the spelling mistakes* and *without pressing* RETURN, key in:

```
spendid, manificent and most whizzo
```

- [After `whizzo` there should be a space.] When a line gets to the edge of the window the text simply carries onto the next line down even if it is in mid-word. Clearly this is all a bit of a mess so, remembering that `M` means `Meta` (and on a PWF this means using `Esc`), key in:

```
M-q
```

- This is the `fill-paragraph` command. It tidies up the paragraph so that each line runs up as close to column 70 as possible without going beyond column 70.
- Click somewhere in each of the other two paragraphs in turn and key `M-q` in each. All three paragraphs should now be filled to 70 characters.

3.30 Key bindings in Emacs

- Most commonly-used Emacs commands can be given in a slow way and in a quick way. The slow way involves keying in the command name as was demonstrated for `save-buffer` in section 3.13. It is much quicker to key in `C-x C-s`. This short version is called a *key binding*.

3.31 Commands without key bindings

- A good way to check spelling is to give the Emacs `ispell-buffer` command which has no key binding. It can of course be given by keying in its full name. As always when giving a command this way begin by:

  `M-x`

- In principle you now key in `ispell-buffer` but fortunately the Emacs completion facility can again save you effort. Key in as far as the `p` of `ispell`:

  `isp` SPACE-BAR SPACE-BAR

- The first SPACE-BAR supplies `ell` and the second supplies a hyphen. You now continue:

  `b` SPACE-BAR

- This time Emacs supplies `uffer`. Press RETURN to invoke the command and the Emacs spell-checker sets to work. Very likely the first word to be queried will be `spendid`. This word is highlighted and suggested correct versions appear at the top of the window. A message in the echo line explains how to get further information.
- The suggestions at the top are numbered `(0)`, `(1)`, etc. and, assuming one of these suggestions is acceptable, you key in the appropriate number. Very likely you key in:

  `2`

- The spell-checker then notes the next error, probably `manificent`, and again there are numbered suggestions. This time you probably need to key in:

  `0`

- The spell-checker probably queries `whizzo` next and is again likely to suggest some alternatives. If you want to change the word but are not happy with any of the suggestions key in:

  `r`

- Note that `r` stands for `r`eplace. The offending word appears in the echo line. Edit `whizzo` in the echo line (use the BACKSPACE-key) so that it is replaced by:

  `wizard`

- Press RETURN and the replacement takes effect. If you *are* happy with the spelling of a word that has been queried simply press the SPACE-BAR to leave the word unchanged.

3.32 Automatic filling of paragraphs

- The Emacs `auto-fill-mode` command saves you from having to key `M-q` in every paragraph. There is no key binding for this command so, using completion, go:

  `M-x au` SPACE-BAR SPACE-BAR `f` SPACE-BAR `m` SPACE-BAR

- Press RETURN and, from now on, paragraph filling will be automatic. Note that `Fill` appears in the mode line. To see what this means, first:

  Click the `.` at the end of `football game.`

- Move the cursor one place right and continue, *on the same line*, by keying in a new sentence:

  `It's a needle match that day and I simply must watch it.`

- The line now splits at a sensible place.

3.33 Emacs conclusion

- This is sufficient Emacs for one session. Follow usual practice: don't leave Emacs, simply save the latest version of the file and iconify the window. First save the file:

  `C-x C-s`

- Now iconify Emacs. Note that there is a summary of the Emacs commands used in this document on page 31.

3.34 Checking the files

- From the Terminal window check the entries in your home directory:

  `ls`

- There should now be about a dozen entries, about half of which relate to your Java programs. These include the source files (which have the `.java` extension), possible back-up files (which end `.java~`) and also the class files (which have the `.class` extension).
- To delete a file, use the `rm` command (standing for `remove`). For example, to delete the back-up file `jobletter798~` give the command:

  `rm jobletter798~`

- Give the `ls` command to check that the file has been removed.

3.35 The xlsfonts and xfd commands

- To see what founts are available, give the `xlsfonts` command. (The absence of the letter '`u`' is a consequence of American spelling!) Since there are several hundred founts, the output should be piped to the `more` command:

  `xlsfonts | more`

- There really are hundreds! To see them all keep pressing the SPACE-BAR. If you get tired, you can quit from the `more` command by pressing `q`.
- To gain a quick impression of what a given fount looks like, use the `xfd` command. For example, to look at the `r24` fount give the command:

  `xfd -fn r24 &`

- A table showing the characters in the `r24` fount appears. This is a *fixed-pitch* fount (all the characters are the same width) and fairly large. When you have finished looking, click `Quit` near the top-left hand corner of the `xfd` window.
- Two other founts are worth a brief look, `5x8` (very small) and `9x15` (the unbold version of `9x15bold`).

3.36 Driving Test

- On the last page of this document there is a driving test in which you have to exercise some of the facilities that have been introduced. Complete this test before finishing. It may be a good idea to read Postscripts A and B before finishing too.

3.37 Finishing

- To end the session, choose Logout in the Main menu and then click OK (and, perhaps, Close). This will clear away all your windows including any which are iconified. Wait for the Linux variant of the message window which refers to the Computer Misuse Act 1990 to appear before leaving the PC. Do not switch off the PC or the monitor.

3.38 Postscript A — Emacs commands

- In the following summary, the Emacs commands which have been mentioned in this document are shown in what is likely to be, very roughly, decreasing order of frequency of use. Since you are advised not to leave Emacs, `C-x C-c` is last in the list.
- The three columns show key bindings (where available), the full formal names of the commands and an informal description:

`C-x C-f`	`find-file`	Find (open) an existing file
`C-x C-s`	`save-buffer`	Save (update) an existing file
`C-x C-w`	`write-file`	Write a new file
`C-_`	`undo`	Undo a command
`C-g`	`keyboard-quit`	Go away, I didn't mean it
`C-d`	`delete-char`	Delete a character
`C-k`	`kill-line`	Delete from the cursor rightwards
`C-y`	`yank`	Bring back killed text
`M-q`	`fill-paragraph`	Tidy up a paragraph
`M-x`	`execute-extended-command`	For giving a command the long way
`C-x C-b`	`list-buffers`	List the buffers
`C-x 1`	`delete-other-windows`	Get rid of other Emacs windows
`C-x b`	`switch-to-buffer`	Change to another buffer
	`ispell-buffer`	Invoke the spell-checker
	`auto-fill-mode`	For automatic tidying of paragraphs
`C-x C-c`	`save-buffers-kill-emacs`	Cancel (leave) Emacs

3.39 Postscript B — Remote use of PWF Linux

- As readers should know, a considerable amount of information relevant to teaching is available via the Computer Laboratory web site `http://www.cl.cam.ac.uk/` and you might note that the Mozilla Firefox browser is recommended when using PWF Linux. Use the command `firefox &` to enter this system.
- From the Computer Laboratory home page, you should select the link Information for current students and look for the section heading Resource material under which you will find further links about PWF Linux.

- The link Information about accessing the PWF remotely advises that if you have access to a PWF Workstation running PWF Linux then use that, but...
- It is further explained that if you don't have access to a PWF Workstation running Linux then remote access to a shared facility is available. There are various ways of achieving this access and you should have no trouble understanding the explanation.

3.40 Postscript C — HWM and Motif

- Throughout this document a standard window manager has been assumed. There are other window managers available, each with numerous facilities for configuring to suit individual taste. Popular alternative window managers include the Hummingbird Window Manager (HWM) and Motif.

F.H. King 16 July 2007

Java Tick 1, Part A — Driving Test

To secure the first Java Tick, a two-part submission is required. Part A is the Driving Test described here. Part B is issued separately. To pass the Unix and X Driving Test, work through the following steps to produce a printout whose appearance is like the lower half of this page. Put the submission in your Ticker's Blue Rack as usual.

- Edit your `ComeAgain.java` file so as to incorporate two Java comment lines at the beginning as in the version of `ComeAgain.java` shown below. Use your name and suitable times.
- Save the amended source code and check that it compiles by using the `javac` command.
- Give the three commands shown below after `c201@pccl504:~>` prompts: these are the `more` command, the `javac` command and the `java` command.
- Next (and not shown on the lower half of this page) give the `cat` command to embark on a new file:

  ```
  cat >tick
  ```
- Highlight and copy the mini-session (as described in section 3.23) so that it forms the contents of the file `tick`. Terminate `cat` with `Ctrl-d`. Use Emacs to edit the file `tick` so that there is one blank line before each line which begins with a prompt.
- Print the file by giving the following command:

  ```
  lpr tick
  ```
- The output should appear on a printer in the room that you are working in. One might think of `lpr` standing for `l`ocal `pr`inter. Apart from your name and timings, the output should appear exactly as below. Pay particular attention to the blank lines.

```
c201@pccl504:~> more ComeAgain.java
//  JAVA ASSESSED EXERCISES.  SUBMISSION 1 FROM F.H. KING.
//  Estimated time to complete: 30 mins.  Actual time: 1 hour.

public class ComeAgain
 { private static int n;

   public static void main(String[] args)
    { n = 57;
      System.out.printf("Come in number %d please%n", n);
    }
 }

c201@pccl504:~> javac ComeAgain.java

c201@pccl504:~> java ComeAgain
Come in number 57 please
```