# An Introduction to Unix

*Geraint A Wiggins*

Based on 'An Introduction to Unix V' by Paul Brna.
In this document, text in *italics* is text input by the user, and followed with a < RETURN> .

## 1. Logging On

### 1.1. Direct Line

Go to section 2.

### 1.2. Prompt indicates a PAD

```
PAD>  call itspna
```

### 1.3. Prompt indicates a TCP

```
Hostname: XCALL
TCP:call itspna
```

## 2. Logging on (contd)

Prompt asks for your userid

```
login: aiperson
Password: qweretryt
```

The characters of your password will not be visible (echoed) on the terminal.

## 3. Logging Out

In reply to the Unix prompt, hold down < CTRL> and hit D:

```
itspna[12]: ^D
logout
```

## 4. Change/ Create a Password

```
itspna[23]: passwd
```

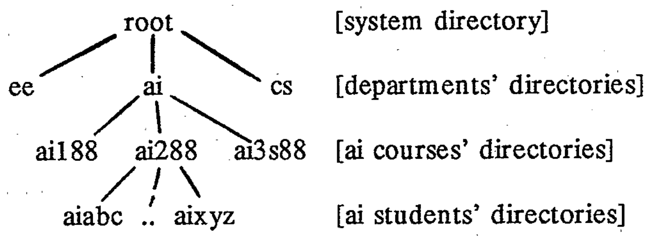
The longer your password is, the less likely it is that someone will guess it.

You must, of course, remember your password. However, DO NOT make it the same as your userid, or something obvious like your name or your boy/girlfriend's name.

## 5. The File Structure

There are two kinds of file in Unix. One is called a 'plain file' (usually just 'file'), and the other is called a 'directory'. The directory, as you might expect, is a list of files and (in Unix's internal language) where to find them.

A directory can refer to both kinds of file, and so, by drawing lines between the names of directorys and the files to which they refer, we can create a 'tree', like this:


root
ee
ai
cs
ai188
ai288
ai3s88
aiabc
..
aixyz
[system directory]
[departments' directories]
[ai courses' directories]
[ai students' directories]

The tree can be divided into 'sub-trees' by taking a directory and looking at just the files and directories referred to 'below' it.

You enter, at login, at the head of your personal sub-tree, which is headed by a directory named after your userid. This is called your 'home directory'.

Normally, you can look at all parts of the tree that have not been explicitly barred from you by the owner of the subtree you try to enter. You are not able to change anything except your own files, unless, again, you have been given special 'permission' to do so, by a file's owner.

## 6. The .login File

In each home directory there should be a file called .login. It contains a set of instructions which Unix will follow each time you log in, so that the system can be personalised to users' requirements.

Your .login has already been set up. If you wish to change it, please speak to Geraint Wiggins, the Undergraduate Computing Officer.

When you log in, the following should happen.

First, a message of the day from the system manager.

Second, something like:

```
TERM = (vt100)
```

This is to check what sort of terminal you are using, to allow the computer to drive it properly. Most of the terminals you will use will be 'vt100 compatible', and this is what will be assumed if you just hit < RETURN> at this point. Otherwise you must enter the type of your terminal and then hit < RETURN> . Some common terminal types are Wyse75 (for which you enter "ws") and Freedom 100 for which you enter "f100".

Third, a message of the day for the AI courses.

Then you should get the standard prompt for ai:

```
itspna[12]:
```

This is, first, to let you know which machine you are on (itspna), and second to tell you what number the current command is in the history list - but don't worry about that for now.

## 7. More about Terminals

If you REALLY want to know all the terminal types issue the following command:

```
itspna[27]: ls -x /usr/lib/terminfo/* | more
```

This produces vast reams of stuff. Best of luck in picking you which you want!

(The command means "list in row format the contents of all the subdirectories of /usr/lib/terminfo and paginate output")

Alternatively, you can set the terminal type by hand. EG:

```
itspna[56]: set term= vt52
```

NB: no spaces round the '= '.

## 8. Using the Mailer

The Unix mailer is called

    mail

It has two modes:

    reading mail
    sending mail

### 8.1. Reading Mail

Do

```
itspna[50]: mail
```

and then try " *?* " when in the mailer.

Conceptually, three things can happen to each message

    held in a system mailbox (if mail not read)
    saved in your own mailbox (a file called mbox) (if mail read but not thrown away)
    thrown away (lost forever)

It is possible to get mail to read your own mbox (to see your old mail) with:

```
itspna[67]: mail -f
```

### 8.2. Sending Mail

Some examples:

```
itspna[1]: mail ecmu21
```

goes to the account holder ecmu21 on the machine that you're on. However, you will need to use "mail" to send messages to people on other machines:

```
itspna[10]: mail helen@aiva
```

goes to user helen on computer aiva.

On entering mail, you are prompted for a subject field. Next, you give your message one line at a time.

Finish your message with ^D at the start of a line to send it on its way. If you make errors, use CTRL/C twice to kill the message. The aborted message will be saved in your home directory in a file called dead.letter.

## 9. Protecting your files

You can use the 'chmod' (change mode) command to protect your files from other users. Initially, your files can be read, searched and/or executed by everybody, but only written by you. Your can change these 'permissions' with respect you yourself, to your 'group' (other undergrads in your year) and everyone else. To find out more about chmod, use the Unix manual command, man, like this:

```
itspna[9999]: man chmod
```

Please don't automatically protect all your files from being read by others. If you have something private, fair enough. It will help the computing staff, however, if you are not too secretive. And please note that the superusers (ie the computing staff) can read, write or execute anything of which they are suspicious!

## 10. And Finally ...

### A Guide To the Essential Unix Commands

This is a very brief introduction to a handful of Unix commands. These should be enough to get you through the first couple of days.

First, some brief points about Unix. It is an operating system that is popular within industrial and academic computing circles. It is a large system and comes in a number of different versions (the one that you will be using is called UTX/3.2). The commands described here are basic ones that should be common to all Unix systems.

### File Manipulation Commands

To get a listing of the files in your current directory use the command "ls". There are two common arguments you can use with this command to provide you with more than a basic listing of files.

These are shown below.

| **command** | **what it does** |
|---|---|
| ls | list visible files and directories in current directory |
| ls -a | list all files and directories in current directory |
| ls -l | long listing of files and directories in current directory |
| ls -al | long listing of ALL files and directories in current directory |

Try the above commands to see the different outputs you get with each one.

The Unix manual is available from your terminal, so you can call up information on commands by using the "man" (short for manual) command together with the Unix command you are interested in, so:

```
    itspna[10]: man ls
```

will list the manual pages for the command "ls". The manual is terse and a bit cryptic when you first start. Try listing out the manual pages for some of the other commands described in this document.

Other basic file manipulation commands you will need to know include removing, copying, renaming and displaying your files. Each of these is shown below.

| **command** | **what it does** |
|---|---|
| rm filename | removes the file called filename |
| cp file1 file2 | makes a copy of file1 and calls it file2. If file2 already exists, it is overwritten. |
| mv file1 file2 | moves file1 to file2. If file2 already exists it is overwritten. The mv command effectively renames the file in this example, but you could use it to move a file to another directory. |
| mv prog /user1/program/prog | move the file called prog to the directory /user1/program and call it prog (as an exercise, after you have read about directories try creating a directory and moving a file to it). |
| more filename | displays the contents of filename one page at a time. Press the space bar to see the next page and press q when you want to quit. |

**Directories**

To see which directory you are in, type in the command "pwd" (print working directory). You can create directories to keep together files that are related in some way (for instance all your Prolog programs) using the mkdir command. To move to a different directory the "cd" command can be used. Some common commands relating to directories are shown below.

| **command** | **what it does** |
|---|---|
| pwd | Print the name of the current directory |
| cd<br>cd ~ | Move me to my "home" directory. |
| cd ~user | Move me to user's home directory. |
| cd .. | Move me back up one level in the directory tree structure. |
| mkdir progs | Make a directory called progs and make it a sub-directory of my current directory. |
| cd prologprog | Move to the directory called prologprog. |
| cd /ai/s1 | Move me to the directory called /ai/s1 |
| rmdir name | Remove the directory called name, but only if it is empty. |

**Redirecting input and output.**

Unix provides a facility whereby input and output can be redirected. In most of the modes that you will use Unix, input from your keyboard is the standard input and output to your screen is the standard output. This can be changed by using the redirection symbols "> " and "< ". If you need to redirect output to a file use the "> " symbol. The "< " is used to redirect the input. Examples of the use of both kinds of symbol is shown below.

| **command** | **what it does** |
|---|---|
| ls -al > filelist | List out the contents of the current directory and place this in the file called filelist. If filelist already exists, it is overwritten. |
| cat file1 file2 > file3 | Concatenate file1 and file2 to form file3. If file3 already exists then it is overwritten. |
| cat file1 file2 > > file3 | Concatenate file1 and file2 and place them at the end of file3. If file3 already exists then the new data is inserted at its end, not overwriting its contents. In general, if you do not want files overwritten when redirecting output then use "> > " instead of "> ". |
| mail < mailthings | Takes mail messages from the file called mailthings (perhaps previously created by an editor) and directs them to the mail program. Redirection of the standard input using "< " is used much less frequently than redirection of the standard output. |

**Connecting Processes with Pipes.**

Unix provides the facility to connect together processes (ie programs) using the pipe symbol "|". This will direct the output of one process to the input of another, creating a 'pipeline'. Some examples of this are shown below.

| **command** | **what it does** |
|---|---|
| ls -a \| wc -l | The overall effect of this command sequence is to print out the number of files in your current directory. The output from the ls command is passed to the wc (word count) process, which with the -l option, counts the number of lines. |
| who \| wc -l | This will tell you how many people are logged on to the system. It works in a similar way to the previous example. |

Pipelines can be as long as you like (within reason), and you can redirect input to the first program in the pipeline and output from the last as above.

**Background Jobs**

Unix is multiuser, multiprocess system. This means that (obviously) more than one user at once can use it. Also, it means that each user can do more than one thing at once. Suppose, for example, that you wanted to word process a file, called, say, essay, through the Unix standard text formatter, nroff. The command

```
nroff -ms essay
```

would do the processing, and display the results on your terminal, and

```
nroff -ms essay >  nr.output
```

would put the output in the file called nr.output. However, if your essay were very long and/or the machine were very busy, this might take a long time. To save wasting your time, you can do

```
nroff -ms essay >  nr.output &
```

The & means 'run this job in the background'. Unix will initiate the command, and run it as usual, but separately from the terminal. You are then free to do something else, or even log off, without affecting its operation.

Please note that running lots of background jobs will slow the system drastically, so PLEASE DON'T do this just to experiment. Note also that a process running in the background (for obvious reasons) can't take input from the terminal - it will stop and tell you so. If a background process writes data to the terminal, it may be very confusing for the user.