Unix Basics

Kirrily Robert
Training Co-ordinator
Netizen Pty Ltd

Unix Basics
by Kirrily Robert

Open Publication License

IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK OF USE OF THE OC IS WITH YOU. SHOULD THE OC PROVE FAULTY, INACCURATE, OR OTHERWISE UNACCEPTABLE YOU ASSUME THE COST OF ALL NECESSARY REPAIR OR CORRECTION.

5. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MIRROR AND/OR REDISTRIBUTE THE OC AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE OC, EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Additionally:

6. If you offer training based upon this Open Content, you must prominently display a notice stating whether or not you are a Netizen Certified Training Organisation on the Open Content itself and on any material advertising or publicising your training.

a) If you are a Netizen Certified Training Organisation, you must state that you are a Netizen Certified Training Organisation and display the Netizen Certified Training Organisation logo. You must also provide a URL for more information, namely http://netizen.com.au/services/training/ncto/

b) If you are not a Netizen Certified Training Organisation, you must state that you are not a Netizen Certified Training Organisation. You may not use the Netizen Certified Training Organisation logo. You must also provide a URL for more information, namely http://netizen.com.au/services/training/ncto/

Revision History

Revision 0.11999-09-29Revised by: KR
Initial version
Revision 0.91999-11-02Revised by: KR
Beta test stage
Revision 1.01999-11-10Revised by: KR
First edition
Revision 2.02000-02-02Revised by: KR
Second edition
Revision 2.12000-06-14Revised by: RL
Minor revision

Table of Contents

List of Tables

Chapter 1. Introduction

About this course

This one-day training module provides a basic introduction to the Unix operating system for those with no previous Unix experience.

Course outline

- What is Unix? (15 minutes)
- Logging in (30 minutes)
- The shell (60 minutes)
- *Morning tea break*
- Getting help (15 minutes)
- Files and directories (60 minutes)
- *Lunch break (60 minutes)*
- Editing (60 minutes)
- Users, groups and permissions (60 minutes)
- *Afternoon tea break*
- Processes (60 minutes)

Assumed knowledge

To gain the most from this course, you should already be familiar with personal computer operating environments such as Windows or MacOS.

Module objectives

- Understand what Unix is, the main features of Unix, and different flavours of Unix
- Know how to log in to a Unix account, change your password, and log out again

- Understand the purpose and features of the Unix shell, including command history, filename expansion (globbing), aliases, etc.
- Know how to find help of various kinds on a Unix system
- Understand Unix files and directories and how to manipulate them, including viewing, copying, moving and deleting
- Know how to use the **vi** editor to manipulate text files
- Understand the concepts of users, groups and permissions on a Unix system, and know how to use commands to identify users and groups, and identify and manipulate permissions on files and directories
- Understand the Unix concept of processes and job control, and know how to kill processes and jobs, and how to run processes or jobs in the background

Materials

The materials for this module include:

- These training notes
- A copy of the textbook, Unix in a Nutshell (System V/Solaris, 3rd edition) by Arnold Robbins
- Exercises and other files (on the training server and provided on floppy disk)

Platform and version details

This module is taught using a Unix or Unix-like operating system such as Linux or FreeBSD. Most of what is learnt will work equally well on other variants of Unix; your instructor will inform you throughout the course of any areas which differ.

The course notes

These course notes contain material which will guide you through the topics listed above, as well as appendices containing other useful information.

The following typographical conventions are used in these notes:

System commands appear in **this typeface**

Literal text which you should type in to the command line or editor appears as `monospaced font`.

Keystrokes which you should type appear like this: **ENTER**. Combinations of keys appear like this: **CTRL**-**D**

```
Program listings and other literal listings of what appears on the
screen appear in a monospaced font like this.
```

Parts of commands or other literal text which should be replaced by your own specific values appears *`like this`*

> **Note:** Notes and tips appear offset from the text like this.

> **Advanced:** Notes which are marked "Advanced" are for those who are racing ahead or who already have some knowledge of the topic at hand. The information contained in these notes is not essential to your understanding of the topic, but may be of interest to those who want to extend their knowledge.

> **Readme:** Notes marked with "Readme" are pointers to more information which can be found in your textbook or in online documentation such as manual pages or websites.

Chapter 2. What is Unix?

In this chapter...

In this section we will discuss a little of the history of the Unix operating system, some of its features, and the differences between different types (or "flavours") of Unix. We'll also look at some ways of thinking about Unix which may help you understand it better.

Unix history

The Unix operating system was developed around 1970 at Bell Labs by Ken Thompson and Dennis Ritchie. It was based on an earlier system called Multics; the name "Unix" was a pun on "Multics" because Unix was a cut down version of the earlier system.

During the 1970s, Unix was re-implemented in the C programming language, which allowed it to be easily ported between different hardware platforms. This resulted in greater popularity for the operating system, which began to be used widely in universities and eventually in commercial organisations.

As of the late 1990s, Unix runs on everything from small PCs and embedded processors to huge mainframes and supercomputers, and is used for tasks ranging from desktop computing and word processing to scientific and financial number-crunching.

Features of Unix

Multi-user, multi-tasking

Unix is a multi-user, multi-tasking operating system.

A multi-tasking operating system is one which can run multiple tasks (such as programs, applications, etc) at the same time. This is usually done by allocating a few milliseconds to each task in turn, so it appears that all the tasks are running simultaneously.

A multi-user operating system is one which can support multiple people using the same system at the same time. Each user can connect to the system and run their own programs without interfering with other users. Unix systems can easily support hundreds of users simultaneously.

Security

If you have many users using a computer system at the same time, it is important to ensure that they cannot read each other's files, interfere with each other's programs, pretend to be another user, or interfere with the underlying operating system. This is referred to as system security.

Unix systems provide security in several ways. Firstly, all users must identify themselves by logging in with a username and password. Once they are logged in, they may only read or write files, or run or stop programs, for which they have permission. The underlying operating system is also protected from interference by normal users.

Network capabilities

Unix systems are commonly used as servers for network services such as electronic mail, World Wide Web, file and printer sharing, and more. Unix servers provide these services through an interface called "sockets".

The TCP/IP protocol, on which the Internet runs, was originally developed at the University of California at Berkeley for BSD Unix systems.

Flavours of Unix

Unix is not a single operating system. Rather, it is a family of operating systems based on a common ancestor, the original Bell Labs Unix.

Unix variants (or "flavours", as they are often called) are usually either "SysV-ish" (that is, mostly similar to System V Unix, which came out of the original Unix at AT&T Bell Labs) or "BSD-like" (mostly similar to the BSD Unix variants developed later at UC Berkeley).

There are both commercial variants of Unix (such as Sun's Solaris, IBM's AIX, Compaq's Tru64 Unix (previously Digital Unix, previously OSF/1) and Hewlett-Packard's HP/UX) and non-commercial variants such as Linux and FreeBSD, NetBSD and OpenBSD.

This training module attempts to teach Unix in a way which is applicable to most Unix variants. When things differ between different Unix flavours, your trainer will point them out and may demonstrate the difference to you.

Understanding Unix

As a language

Learning Unix is like learning another language. Like human languages, Unix has its own vocabulary, grammar and syntax, all of which are initially difficult to grasp, but become easier with

practice.

If you have ever learnt a second language, you will know that the first stages are very difficult, and that it is only with perseverence and practice that you will become fluent and eventually start to think or even dream in that language.

As a tool

Unix, like all computer systems, is a tool. However, it’s not a hammer or a screwdriver -- Unix is a Triton workbench, with all kinds of attachments and spinning blades and specialised attachments like routers (pun very much intended).

If you’re used to using an ordinary hammer, or perhaps a different kind of tool like a sewing machine, you might look at a Triton workbench in bewilderment. Imagine trying to find the bit on a Triton workbench that’s used for banging in nails, or the embroidery attachment!

Nevertheless, once you understand how the tool works and what its strengths are, it can help you work much more efficiently and provide you with orders of magnitude more power than a hammer ever could.

Chapter summary

In this chapter you have learnt:

- Unix is an operating system developed around 1970 at Bell Labs by Ken Thompson and Dennis Ritchie
- Unix is a multi-user, multi-tasking operating system
- Unix has many security and networking features
- Unix comes in a variety of different "flavours", both commercial and Open Source.
- The two main flavours of Unix are "System V" and "BSD"
- Learning Unix is like learning a new language, and requires patience and practice to become fluent
- Unix is a very powerful tool, more similar to a Triton workbench than a hammer, and can help you work very efficiently once you learn how to use all its attachments

Chapter 3. Logging in

In this chapter...

In this section, you will learn how to log in to a Unix system, change your password, and log out again.

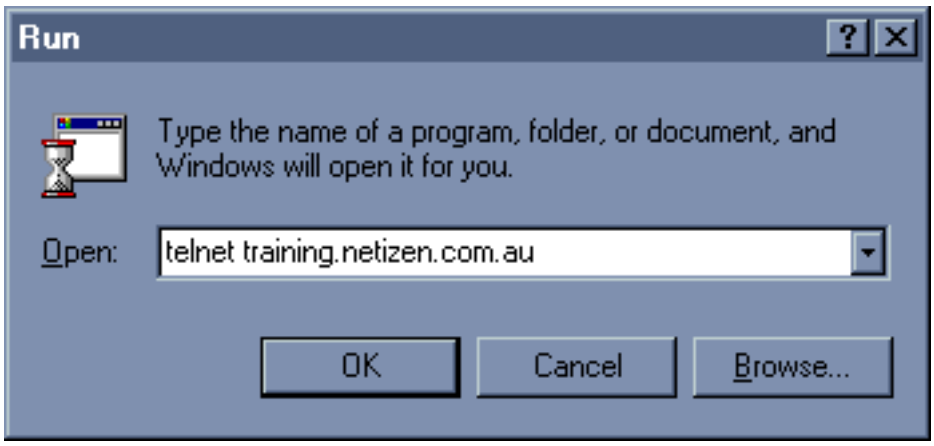
How to log in

Almost all Unix systems will require you to "log in" before you can use them. Logging in tells the system who you are, provides security and makes it possible to have a true multi-user environment.

To log in to a Unix system you either need to be sitting in front of the system's console, or connect to the system via a network of some kind.

The tool most commonly used to connect to an Internet-connected Unix system is called **telnet** and is available on most systems, including Windows PCs.

To run **telnet** on a Windows machine, choose "Run" from the Start menu, and type `telnet` into the dialog box that appears. You can also type `telnet` *`hostname`* (where *`hostname`* is the name of the Unix system you're connecting to) to save having to use the menus in the **telnet** program.



This screenshot shows the Run dialog from which you can run the **telnet** command.

Other telnet/terminal emulation programs for Windows include:

- TeraTerm
- QVTTerm/QVTNet

These and other telnet programs should be available from any good Windows software download site on the World Wide Web, such as http://www.tucows.com/.

> **Advanced:** Before the days of inexpensive PCs, people would use special terminals to connect to Unix systems. These were screens with keyboards but very little processor power (just enough to display things on the screen in text mode) which were sold by various hardware vendors. These terminals would be connected by a serial line (like a modem cable) directly to the Unix system.
>
> Some of the most popular terminals were the VT terminals sold by DEC, including the VT-100, VT-220, and VT-320.
>
> Modern telnet programs for PCs are also referred to as "terminal emulation" programs because they act like the old hardware terminals. In the setup menu of your telnet program, you should be able to choose which type of terminal to emulate. The most common is VT-100.

Exercise

Use **telnet** to connect to Netizen's training server. Your trainer will demonstrate how to do this using your PC. You will need to fill in the following details for use throughout this course:

Table 3-1. Details required to connect to the Netizen training server

Hostname or IP address	
Your username	
Your password	

You should see a screen like this:

```
Trying 203.30.75.3...
Connected to training.netizen.com.au.
Escape character is '^]'.

login:
```

You will have been given a username and password. Type your username in where it says "login", then press **ENTER**. You should see this:

```
login: train01
password:
```

Type in your password, then press **ENTER**. You will not see anything on the screen, for security reasons - even a row of asterisks could indicate the number of characters in your password to an observer.

If you have typed in your username and password correctly, you should see something like this:

```
Linux yt 2.2.12 #3 SMP Tue Sep 21 10:17:11 EST 1999 i586 unknown
```

```
Last login: Mon Nov  8 22:22:53 from 123.4.5.6
You have new mail.
%
```

This is the Unix command line prompt.

If you are having trouble logging in, check the following:

- Are you using the correct mix of upper and lower case letters? Unix usernames and passwords are case-sensitive, so an "a" is not the same as an "A".
- Sometimes the login screen does not understand the backspace key. If you are making mistakes and backspacing over them, this may be the problem. Just press **ENTER** a couple of times until you see a new login prompt.

What will you see?

Message of the day

On logging in, you may be presented with a "message of the day" (or MOTD) which is a message from the system administrator to the users. Often this message warns of upcoming maintenance or downtime for the system. Always read the MOTD carefully.

Terminal type

Some systems will ask you to specify your terminal type. If you are using a terminal emulation program, this is most likely to be "vt100".

Fortune cookie

Some systems will print out a "fortune cookie" when you log in -- often a short humourous quote or message. You can get a fortune cookie any time by using the **fortune** command.

The prompt

Finally, you will see a prompt. This may look like any of the following:

```
%
$
systemname:~>
user@systemname:~>
```

... or any of a number of different things (it's configurable -- we'll learn how later).

The prompt is basically the system telling you it's ready for you to type in a command. You can type in any Unix command at the command line, and press **ENTER** to run it.

Changing your password

To change your password, you need to use the **passwd** command. Simply type **passwd** into the command line, and press **ENTER**.

```
% passwd
Changing password for train01
Old password:
```

You need to type in your old password to prove that you have the right to change the password. This protects you a little in case you have wandered away from your keyboard and someone else tries to change your password without your permission.

Next, type in a new password then confirm it by typing it in a second time:

```
New password:
Re-enter new password:
Password changed.
```

Note that the passwords you type in never appear on the screen.

Some systems require you to use passwords which are difficult to guess, and may require that your password:

- is longer than a given number of characters (usually 5)
- contains numbers and/or mixed-case letters
- is not a common dictionary word
- is not related to your username
- is not the same as the one you previously used

> **Advanced:** The passwords are stored in encrypted format in the `/etc/passwd` file, along with other information about the user.
>
> The type of encryption used is "one-way" which means that the password stored in `/etc/passwd` cannot be decrypted. Instead, the password typed in by the user when logging in is encrypted in the same way, and the two encrypted passwords are compared.
>
> A common way of finding out someone's password is called a "dictionary attack", whereby all the words in a dictionary are encrypted and compared to the encrypted passwords in `/etc/passwd`. This is why many systems require that your password consist of mixed upper and lower case letters and numbers, and not be a common dictionary word.

Hints for thinking up a good password

- Take a word and substitute certain letters for numbers or other symbols. For instance, change all "i" characters to the number 1 or the | (pipe) symbol, or change all instances of the letter "s" to dollar signs.
- Take a memorable phrase, a line of a song, or a famous quotation, and use the first letter of each word as your password. For instance, "Now is the winter of our discontent" could become "Nitwood", or even "N1tw00d" (using the numbers one and zero instead of normal letters)

What to do if you forget your password

If you forget your password, you will need to contact your techical support staff or system administrator to get it changed. Note that they won't be able to tell you what it currently is -- even the system administrator can't find that out. You will have to be given a new password, which you can then change to something more memorable if necessary.

Exercise

1. Change your password using the **passwd** command

Logging out

To log out of the system, simply type **logout** and press **ENTER**.

Some systems also allow you to logout by typing **exit** or using **CTRL**-**D**.

> **Note:** Some systems are set up to automatically log you out after some period of inactivity.

Exercise

1. Try logging out and in again using either **CTRL**-**D** or the **logout** command.

Chapter summary

In this chapter you have learnt:

- how to log in to a Unix system
- how to change your password using the **passwd** command
- some techniques for creating secure passwords
- how to log out of a Unix system using either the **logout** command or other means.

Chapter 4. The shell

In this chapter...

In this section you will learn about the Unix shell, which processes commands typed by the user.

What is a shell?

The Unix operating system itself does not handle commands typed in by the user. Instead, there is a program called a shell which interprets user commands and passes messages to the underlying operating system.

What you see when you first log in to a Unix system is the shell. Usually there will be a prompt which looks something like this:

```
train01@vitaly:~>
```

The prompt varies considerably depending on the shell used and the specific setup. Modern setups usually show the username, the name of the Unix system, and the directory the user is currently in. Older systems may not have configured the shell to do this, and may simply show a percent sign (`%`) or a dollar sign (`$` depending on the shell used. In the examples given in these notes, we'll usually use `%` to indicate the command line prompt, like this:

```
% passwd
```

This is common usage, and you'll often see it in other books and documentation.

> **Readme:** Chapter 3 of the textbook, "The Unix Shell: An Overview" introduces the shell in more detail. It begins on page 201.

Different shells

There are several different shells available on most Unix systems. The two most basic ones are called the Bourne shell (named for its inventor) and the C shell (which is both a pun on "seashell" and an indication that it has some similarity to the C programming language). The programs which run these shells are called **sh** and **csh** respectively.

Two more modern shells are the Bourne Again shell (yes, another pun) which is called **bash**, and **tcsh** which is an enhanced version of **csh**.

> **Note:** The T in **tcsh** stands for TENEX, the name of a 1970s era operating system. You can read the story about it in the **tcsh** manual page by typing **man tcsh**.

Lastly, there are a number of less common shells such as:

- **ksh**
- **zsh**

> **Readme:** The Bourne and Korn shells are described in chapter 4 of the textbook, beginning on page 207. The C shell is described in chapter 5, beginning on page 260.

In this training module, we will primarily use **tcsh** and also take a brief look at **bash**. The notes will usually describe techniques to use in both shells, but you can mostly ignore the **bash** instructions for now, and just use them as a reference in the future.

> **Advanced:** Some systems will be set up to allow you to change your shell using the **chsh** command. Simply type **chsh** at the command line and follow the prompts. You will have to type in your password, and specify the full path to the shell you wish to use.
>
> For more information about the **chsh** command, type **man chsh**.

Using the shell

Simple commands

The most simple command in Unix is a single word, which is the name of a program to run.

```
% passwd
% fortune
```

To run a simple command, simply type the command's name and press **ENTER**. Note that program names in Unix are usually all lower case characters, and are case sensitive -- if you use upper case characters, you will get an error message.

> **Note:** Unix command names tend to be very short, often only a couple of letters. The original developers of Unix either suffered from very bad RSI or were too lazy to type long commands, and thus tended to abbreviate every command to as few characters as possible. Hence "password" became **passwd**, "copy" became **cp**, "make directory" became **mkdir**, and so on.
>
> And if you think that's confusing, just wait til you hear about some of the commands with really *obscure* names -- for instance, there's an email notification utility called **biff** that's named after the labrador retreiver at UC Berkeley that used to bark when the postman delivered mail.

Some commands take "arguments" or "parameters". These are extra parts which come after the command's name and vary the behaviour of the command.

For instance, to print out a short message to the screen:

```
% echo Hello
```

In the example above, **echo** is the command, and the word `Hello` is the argument.

To find a "long" fortune cookie message:

```
% fortune -l
```

In the example above, the `-l` (pronounced "dash-ell") is the argument.

To find all fortune cookie messages containing the word "Winston" (for instance, those which are quotes made by Winston Churchill):

```
% fortune -m Winston
```

In this case, the `-m Winston` are a command line switch and the value associated with that switch which means "find quotes which match the word 'Winston'".

An argument or parameter which starts with a hyphen is often referred to as a "command line switch" or simply a "switch". You can think of the command as having a dashboard or panel with a row of switches on it, and you flick down any switches you want to affect how the program runs. Some switches take values (as in the case of `-m Winston`), while others are simply on or off (as in the earlier example of `fortune -l` for a long fortune cookie).

> **Note:** The dash or hyphen used on Unix command line switches is similar in purpose to the slash (`/`) used under MS-DOS. Unix can't use a slash because it uses that character as a directory separator, where MS-DOS uses a backslash (`\`).

Some commands take a filename as an argument. For instance, the **cat** can be used to print a text file to the screen:

```
% cat myfile.txt
This is my file.
It contains several lines of text.
Here is some more text from my file.
%
```

> **Readme:** Common Unix commands are described in Chapter 2 of the textbook, starting on page 11.

Exercise

1. Try some of the commands given above by typing them at the shell prompt.

- **fortune**
- **fortune -l**
- **fortune -m Winston**
- **cat myfile.txt**

Filename wildcards

Sometimes when using a command which takes a filename for an argument, we want it to use several files at once. Sometimes we can specify all the filenames one after the other:

```
% cat myfile.txt otherfile.txt
This is my file.
It contains several lines of text.
Here is some more text from my file.
This is my other file.
It contains different text to myfile.txt
%
```

However, if we want to act on several files at once and they all have similar names, we can use "wildcards" to specify the group of files we want. The correct name for wildcards under Unix is *globs*, and the process by which the shell expands globs to match filenames is called *globbing*.

The most commonly used glob character is the asterisk, or star. It represents "anything", or more specifically, zero or more unknown characters in a filename.

To print out all files starting with the letter A:

```
% cat a*
```

To print out all text files:

```
% cat *.txt
```

There are other glob characters with other meanings, too. The character `?`, for instance, indicates any single character.

To match files whose names are a single letter followed by `.txt`:

```
% cat ?.txt
```

Square brackets (`[` and `]` can be used to specify a group of characters, any one of which may be matched. For instance, `b[aeiou]g` would match `bag`, `beg`, `big`, `bog`, and `bug`.

> **Readme:** Globbing is discussed in the **tcsh** manual page under "Filename substitution" and in the **bash** manual page under "Pathname expansion".

> **Advanced:** The shell performs *expansion* on its arguments, which means that it expands any wildcards out into a list of matching files, substitutes in any variables specified on the command line, and so on. This expanded command line is then passed to the program itself. It is important to note that programs under Unix do not ordinarily do their own expansion of arguments. This is contrary to the behaviour of MS-DOS, where all expansion is handled by the programs.

Exercise

Practice using **cat** to print out

- all files with the extension `.txt`
- all files whose names start with "a"

Filename completion

Many modern shells, such as **bash** and **tcsh** provide filename completion. This can save you typing out long file names. For instance, if you had typed:

```
% cat myf
```

in **tcsh**, you could press TAB to have it automatically complete the filename for you:

```
% cat myfile.txt
```

If the shell cannot complete the filename, it will complete as much as it is able to, and usually make a beeping sound.

Exercise

- Practice using the shell’s filename expansion

Command history

Most shells provide a command history. This is a facility which remembers commands you previously typed and allows you to repeat them easily.

Modern shells such as **bash** and **tcsh** allow you to use the **UP** and **DOWN** arrow keys to scroll forwards and backwards in your command history. Simply use the arrow keys to move up or down to the command you want, optionally edit it if you want to make any changes, then press **ENTER** to run it again.

All shells, including older shells such as **sh** and **csh** allow you to access the command history using special shell commands:

Table 4-1. Command history

Command	Meaning
history	list command history
!!	repeat previous command
!*n*	repeat command *n from history list*
!*string*	repeat most recent command starting with *string*
!$	last argument of previous command
^*old*^*new*	repeat last command, substituting *old with new*

Exercise

- Practice using the history commands listed above.

Redirection and pipes

Sometimes we want to save the output of a command to a file, such as when we ran **fortune -m Winston** and saw a long list of quotes by Winston Churchill which flew by too quickly to read.

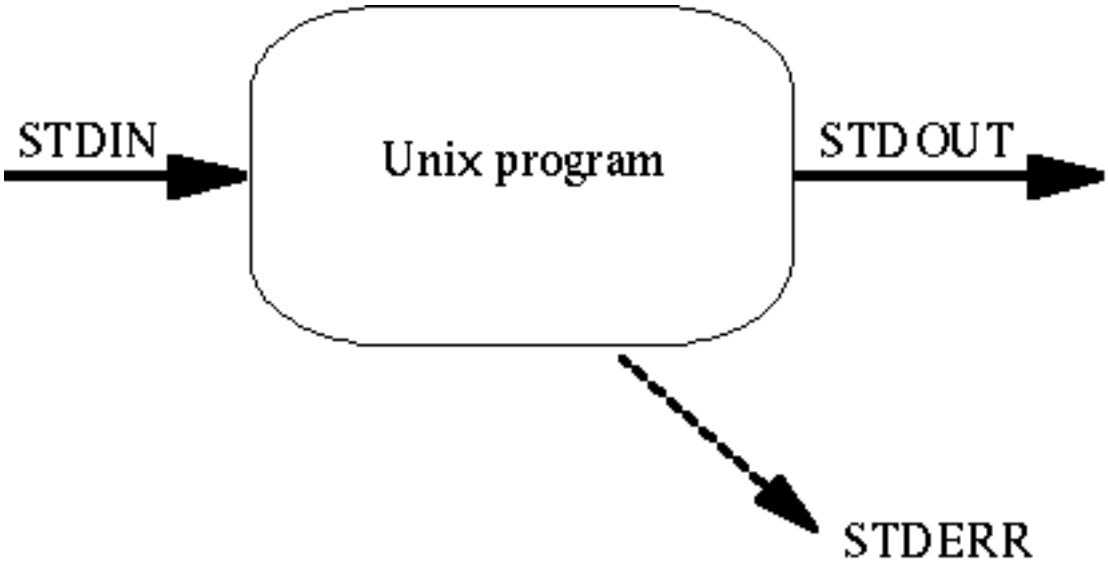
Other times, we want the output of one command to become the input to another command -- for instance, we could take the output of **cat** and use the **sort** command on it to sort the output.

The mechanisms used to achieve this under Unix are called "redirection" and "pipes".

Standard input, output and error

Programs typically have input (the data which is given to the program) and output (the data produced by the program). Programs may also produce error messages.

Under normal circumstances, the input to a program under Unix is the user typing things at their keyboard, and the output is to the terminal. These are called "standard input" and "standard output" respectively, and are often abbreviated to "STDIN" and "STDOUT". Error messages are usually also output to the terminal. This is referred to as "standard error" or "STDERR" for short.



This diagram shows the flow of data into a program through standard input, out via standard output, with errors coming out through standard error.

However, standard input doesn't necessarily have to be the keyboard, and standard output and errors don't necessarily have to be the terminal.

We can cause a program to take its input from a file as if the contents of that file were being typed at the keyboard. We can also send the standard output of a program to a file so it appears in that file exactly as it would have on the terminal. This is called redirection.

If we take the output of one command and feed it directly into the input of another, that's called a "pipe". You can imagine the programs as a series of machines connected together by pipes; you pour the input in at one end then it flows through each machine in turn via the pipes, and eventually comes out the other end.

Using redirection to output to files

To redirect the output of a command, we use the greater-than symbol, which looks like an arrow pointing out of the program and into the file:

```
% fortune -m Winston > quotes.txt
```

We can append to an existing file using a double greater-than sign. The following example searches the fortune cookies for quotes by Oscar Wilde and adds them to the quotes file we created earlier:

```
% fortune -m Wilde >> quotes.txt
```

Exercise

- Run the first command shown above to output a list of quotes by Winston Churchill to a file called `quotes.txt`
- Use the second command shown above to append some Oscar Wilde quotes to your `quotes.txt` file.

Using redirection for input, and using pipes to filter output

We can use our quotes file as input to another program, such as **sort**, by using the less-than symbol:

```
% sort < quotes.txt
```

We can combine them, so that the sorted output is sent to another file:

```
% sort < quotes.txt > sorted.txt
```

However, often we will combine many steps into one using a pipe. Pipes use the vertical-bar character which is usually on the same key as the backslash. The following command finds Winston Churchill quotes in the fortune cookie database, pipes the output through the **sort**, then redirects the output to a file.

```
% fortune -m Winston | sort > sorted.txt
```

We will be using redirection and piping much more widely in the *Unix Tools* training module, which follows today's training.

Configuring the shell

You can configure your shell to look or behave differently depending on your needs and preferences. Think of it as being similar to the Windows control panel, where you can choose various settings to customise the desktop and programs to your personal tastes.

Under Unix, configuration commands can be typed directly into the command line. However, typing them in every time you login to a Unix system would not be very useful. More usually, they're all kept in a file which is automatically read by the shell when it starts up (that is, when you log in).

Each shell has a different file that it keeps its configuration commands in:

Table 4-2. Default configuration files for various shells

Shell	Configuration file
sh	`.profile`

Shell	Configuration file
csh	`.cshrc`
bash	`.bashrc`
tcsh	`.tcshrc`

These files are usually found in your home directory. The leading dot on their names means that they are hidden files.

For now, we'll be entering configuration commands directly into the command line. Later, after we've learnt how to use a text editor, we'll create a `.tcshrc` file so that our configuration commands will be run whenever we log in to the system.

Setting the prompt

Most shells allow you to customise your prompt. Often, the prompt will be customised to show what system you are logged in to, what username you are using, or what directory you are in.

Details of changing your prompt are given below for **bash** and **tcsh**. To change your prompt in another shell, read the manual page for that shell.

Setting the prompt using tcsh

Under **tcsh** the prompt is set by setting the `prompt` variable:

```
training:~> set prompt="What is your command? "
What is your command?
```

The following characters are a sample of those which can be used to put special information into the prompt.

Table 4-3. Special characters in prompts for tcsh

Characters	Meaning
`%/`	The current working directory
`%M`	The full hostname (eg `training.netizen.com.au`)
`%m`	The first part of the hostname (eg `training`)
`%T`	Time of day in 24 hour format
`%n`	Username

Here are some examples:

```
% set prompt = "What is your command? "
What is your command? set prompt = "%m: "
training: set prompt = "%n@%m: "
train01@training: set prompt = "%n@%m:%/> "
```

```
train01@training:/home/train01>
```

> **Readme:** More information about special characters in prompts for **tcsh** can be found in the **tcsh** manpage.

Setting the prompt using bash

Under **bash** the prompt is set by changing the value of the `PS1` variable:

```
bash-2.01$ PS1="What? "
What?
```

The following table shows characters which can be used in a prompt and which will be substituted with a piece of useful information.

Table 4-4. Special characters in prompts for bash

Characters	Meaning
`\w`	The current working directory
`\H`	The hostname (eg `training.netizen.com.au`)
`\h`	The first part of the hostname (eg `training`
`\t`	Current time in 24-hour format
`\u`	Your username

> **Readme:** More information about special characters in prompts for **bash** can be found in the **bash** manpage, in the section labelled "PROMPTING".

Exercise

1. Change your prompt from the command line. Experiment with different prompts and find one you like.
2. A good one to try is **set prompt="%m:%/> "**

Aliases

Most shells provide the facility of setting "aliases", which are usually abbreviations for longer commands. For instance, if you wanted a shorter way to get a fortune cookie, you could make an alias so that you only had to type `f` instead of the full command.

The sections below look at how to create aliases in **bash** and **tcsh**. If you are using a different shell, you will need to read the relevant manual page to find out how to set aliases in that shell.

Aliases in **tcsh** and **bash** are set using the **alias** command and unset using the **unalias** command.

The **alias** command can be used to list the aliases you have set, or may takes two arguments -- the alias to be set, and what to expand it to -- to create an alias. Under **bash**, you will need to put an equals sign (=) between the two arguments and enclose the full version of the command you want executed in quote marks, if it is more than one word long.

Here is a **tcsh** example:

```
training:~> alias winnie fortune -m Winston
training:~> winnie
Men occasionally stumble over the truth, but most of them pick themselves
up and hurry off as if nothing had happened.
                -- Winston Churchill
training:~> unalias winnie
training:~> winnie
winnie: Command not found.
training:~>
```

Here is the equivalent **bash** example:

```
bash-2.01$ alias winnie="fortune -m Winston"
bash-2.01$ winnie
Men occasionally stumble over the truth, but most of them pick themselves
up and hurry off as if nothing had happened.
                -- Winston Churchill
bash-2.01$ unalias winnie
bash-2.01$ winnie
bash: winnie: command not found
```

> **Readme:** More information about aliases in **tcsh** can be found in the **tcsh** man page, under "Alias substitution". More information about aliases in **bash** can be found in the **bash** man page, under "ALIASES".

Exercise

1. Create an alias for the **fortune** command.
2. Use the **alias** with no arguments to list your aliases
3. Delete that alias using the **unalias** command.

Environment variables

Environment variables are essentially settings which tell the shell how to behave in certain circumstances. For instance, you can set the `EDITOR` variable to tell the shell what your favourite editor is, and it will look at that value and bring up the appropriate editor by default in some sitations.

Some of the environment variables you can set include:

Table 4-5. Some environment variables

Variable name	Meaning
`EDITOR`	Your favourite text editor
`PATH`	Where the shell should look for programs to run
`HOME`	The location of your home directory
`http_proxy`	The URL of your web proxy server
`NNTPSERVER`	The hostname of your Usenet news server
`MAIL`	The location of your email "inbox"

There are many other environment variables which you can set. Often, the manual pages for a program will have a section at the bottom entitled "ENVIRONMENT VARIABLES" which will list environment variables whose settings affect the functioning of the program in question.

Environment variables in csh and tcsh

To set environment variables in **csh** or **tcsh**, use the **setenv** command.

```
% setenv NNTPSERVER news.netizen.com.au
% setenv EDITOR /bin/vi
```

You can view environment variable settings by using the **setenv** command with no arguments.

```
% setenv
GROUP=fred
HOME=/home/fred
PATH=/bin:/usr/bin:/usr/local/bin
USER=fred
```

Environment variables in sh and bash

To set environment variables in **sh** or **bash**, you first set the value, then ensure it will be exported to any programs subsequently run by using the **export** command.

```
% EDITOR="/bin/vi"
% export EDITOR
```

You can view environment variable settings by using the **set** command with no arguments.

```
% set
GROUP=fred
```

```
HOME=/home/fred
PATH=/bin:/usr/bin:/usr/local/bin
USER=fred
```

Sample configuration files

Here are some sample configuration files for **tcsh** and **bash**. Later, after we've learnt how to use a text editor under Unix, we'll be creating our own configuration files to customise the shell environment to our own tastes. For now, these are just here as a reference for later use.

Note that lines beginning with a hash sign (`#`) are comments.

Sample `.tcshrc` configuration file

```
# set the prompt
set prompt="%m:%/> "

# set some aliases
alias f fortune
alias rm rm -i
alias dir ls -al

# set our favourite editor
setenv EDITOR /usr/bin/vim

# set our web proxy
setenv http_proxy http://proxy.netizen.com.au:8080/
```

> **Readme:** A default `.cshrc` file is given in the textbook on page 269.

Sample `.bashrc` configuration file

```
# set our prompt
PS1="\h:\w> "
export PS1

# some aliases
alias f fortune
alias rm rm -i
alias dir ls -al

# set some miscellaneous environment variables
MAIL=/var/spool/mail/skud
EDITOR=/usr/bin/vim
PAGER=/usr/bin/less
http_proxy=http://proxy.netizen.com.au:8080/
export MAIL EDITOR PAGER http_proxy
```

Chapter summary

- In Unix, the shell is what interprets what you type and turns it into commands for the system to run
- There are different shells that you can use, depending on your personal taste and what's installed on the system
- These include **sh**, **csh**, **bash**, **tcsh** and others
- **tcsh** is the shell primarily used for examples throughout this training module
- Simple commands are run by typing the name of the command and pressing **ENTER**
- More complex commands can be run by adding arguments to the command line
- Arguments which start with a dash or hyphen (-) are referred to as switches
- Groups of files can be selected by using wildcards (known as *globs* under Unix)
- Most modern shells will automatically complete filenames if you press **TAB**
- Most modern shells allow you to access and edit earlier commands using arrow keys; there are also special commands for manipulating earlier commands
- Redirection and pipes can be used to provide different input and output to programs
- The shell can be configured in various ways to suit your preferences
  - You can set the prompt to whatever you want
  - You can create aliases to provide command shortcuts
  - You can set environment variables to influence the behaviour of some programs
  - You can put all your configuration commands into a file which will be read when the shell starts

Chapter 5. Getting help

In this chapter...

In this section you will learn about the three main ways to find help when using a Unix system: manual pages, other documentation, and your system administrator.

Manual pages

Almost all Unix programs come with manual pages. These are terse, technical explanations of what the program does.

To read a manual page for any command type **man *commandname*** (substituting the appropriate command name, of course).

For instance:

```
% man passwd
```

This will give you information about the **passwd** command which we used earlier. Here are the first few lines of the **passwd** man page:

```
PASSWD(1)                                                 PASSWD(1)

NAME
       passwd - change user password

SYNOPSIS
       passwd [-f|-s] [name]
       passwd [-g] [-r|R] group
       passwd [-x max] [-n min] [-w warn] [-i inact] name
       passwd {-l|-u|-d|-S} name

DESCRIPTION
       passwd  changes  passwords for user and group accounts.  A
       normal user may only change the  password  for  their  own
       account,  the  super  user may change the password for any
       account.  The administrator of  a  group  may  change  the
       password  for  the  group.   passwd  also  changes account
       information, such as the full  name  of  the  user,  their
       login shell, or password expiry dates and intervals.
```

The man pages are presented to you using a pager program which allows you to view the text a screenful at a time. On most modern systems, this pager is a program called **less**. You can scroll through the man pages as follows:

Table 5-1. Navigating man pages

Movement	Keystroke
Down a page	**SPACE**
Up a page	`b`
Down one line	`j` or **ENTER**
Up one line	`k`
Search for a string	*/string*
Quit	`q`

> **Note:** The above table assumes that your system uses the **less** program as its pager. Some older systems use **more** which does not allow you to go back in the file, only forward. An even more antiquated pager called **pg** is very occasionally seen; if your system uses this, be very afraid.

Finding the right man page

If you have some idea what you want to do, but aren't sure of the exact command name, you can get a list of likely manual pages by using the **apropos** or **man -k** commands. These commands are roughly equivalent; both search for keywords in the single-line description of utilities and programs as given in their manual pages.

```
% man -k "card game"
canfield (6)         - the solitaire card game canfield
cribbage (6)         - the card game cribbage
% apropos "card game"
canfield (6)         - the solitaire card game canfield
cribbage (6)         - the card game cribbage
```

Categories for man pages

The manual pages on any Unix system are separated into categories. These categories vary slightly between systems, but typically they are:

Table 5-2. Manual page categories

Category number	Category contents
1	Executable programs or shell commands
2	System calls (functions provided by the kernel)
3	Library calls (functions within system libraries)

Category number	Category contents
4	Special files (usually found in /dev)
5	File formats and conventions eg /etc/passwd
6	Games
7	Macro packages and conventions eg man(7), groff(7).
8	System administration commands (usually only for root)

Sometimes it will happen that there are multiple manual pages with the same name, but in different sections of the manual. For instance, there is a manual page for the **passwd** command and also for the file with the same name which is found in `/etc/passwd`.

You can distinguish between these manual pages by typing **man *n command*** where *n* is the section/category number.

```
% man 5 passwd

PASSWD(5)                                                PASSWD(5)

NAME
       passwd - The password file

DESCRIPTION
       passwd  contains  various  pieces  of information for each
       user account.  Included is

            Login name

            Optional encrypted password

...
```

As you can see, this is a different manual page to the one you would see if you just typed **man passwd**. The system will, by default, choose the lowest-numbered manual section first, so the **man passwd** would default to **man 1 passwd** (the **passwd** command) over the file `/etc/passwd` which is documented in section 5, and would be viewed by typing **man 5 passwd**.

> **Note:** Some systems require you to use **man -s *section-number command*** instead of just **man *section-number command***.

How manual pages are written

Most manual pages start out with a name and brief description of the command in question, then a very brief (and rather arcane) summary of all available command line options.

The bulk of most manual pages is made up of a description of all the command line options to the command and what each one does. Typically there will be a few lines of discussion for every possible option. Some examples of usage may also be given.

At the bottom of the manual page, you will often find sections marked "BUGS" (known problems with the command), "FILES" (files which are related to the command), "ENVIRONMENT VARIABLES" which affect the behaviour of the command, "SEE ALSO" which lists other manual pages which are related, and "AUTHOR" which names the author of the program and often provides a contact email address.

Exercise

1. Take a look at some of the manual pages for programs we've looked at so far:
   - **fortune**
   - **passwd**
   - **cat**
   - **tcsh**

Other documentation

Sometimes (in fact, often) the man pages are too terse and technical for your needs. Man pages focus on the syntax of commands rather than on "How do I..." or "Why should I..." To find out that kind of information, you will need to read other documentation.

There are three main places to find other documentation:

- The documentation directory
- The World Wide Web
- Books

The documentation directory

On most systems, the directory `/usr/doc` or `/usr/local/doc` will contain additional documentation for installed software, including FAQs (Frequently Asked Question files), examples of use, etc.

WWW documentation

A great deal of documentation is available on the World Wide Web or elsewhere on the Internet. Many FAQs and tutorials are available, including:

- "Unixhelp for Users" from the University of Edinburgh, at http://unixhelp.ed.ac.uk/
- Norm Matloff's "Unix Tutorial Center" at the University of California at Davis, at http://heather.cs.ucdavis .edu/~matloff/unix.html (http://heather.cs.ucdavis.edu/~matloff/unix.html)
- "A Basic Unix Tutorial" from Idaho State University, at http://www.isu.edu/departments/comcom/unix/workshop/unixindex.html (http://www.isu.edu/departments/comcom/unix/workshop/unixindex.html)

Exercise

1. Use a search engine or web directory to search for Unix tutorials or other documentation. Some search engines include:
   - Google at http://www.google.com/
   - Altavista at http://www.altavista.com/
   - Yahoo at http://www.yahoo.com/
2. Share your findings with the class

Books

There are literally hundreds of books available about Unix and Unix applications, ranging from very simplistic "Dummies" books to highly technical tomes. Take the time to visit a good bookshop and browse their shelves.

Sysadmins: Care and feeding

If neither manpages nor other online documentation nor books have helped you solve your problem, you may need to consult your system administrator or other technical support staff. Here are some hints you will find useful:

- Read the manuals FIRST, and explain where you have already looked. Example: "I'm trying to use X to do Y, but the man pages for X don't seem to contain the information I need."
- Explain your eventual goal, not just the tools you are having trouble with. Example: "I'm trying to save my email to different folder each month", rather than "Do crontabs have an option for setting

the month”. The latter example will cause confusion, since your sysadmin or support person won’t understand what you’re actually trying to do.

- If you are getting an error message, copy it out in full. Example: "It said Permission Denied when I tried to cat /dev/audio" is much more useful than "It gave me an error message". The worst possible error report is "it doesn’t work". If you say this to your system administrator, do not expect any kind of helpful response.
- A sysadmin’s main job is keeping the systems running, not helping end-users. Because of this, they are often grumpy when you ask them for help, and may require appeasement. System Administrators often appreciate donations of chocolate, pizza, beer, Jolt cola, etc. Be nice to your sysadmins, and they will be nice to you.

Chapter summary

- Most documentation and help on Unix systems can be found in the online manuals
- The manuals are accessed using the **man** command
- The **apropos** command can be used to find which manual pages best match your needs
- Additional documentation can often be found in the `/usr/doc` or `/usr/local/doc` directories on a Unix system
- There are many Unix-related resources on the World Wide Web
- Most bookshops will have several books on Unix which may be of use to you
- As a last resort, you can consult your system administrator. Bribes of pizza, caffeine or alcohol are often a good way to gain their assistance.

Chapter 6. Files and directories

In this section...

In this section we look at the Unix file system and learn to find our way around by changing directories, listing files, and looking at the contents of files.

Everything is a file

Under Unix, almost everything is a file and can be viewed and manipulated via the file system. Hardware devices, directories, email, logs, configuration for programs - all are simply files which can be treated in exactly the same way.

Filenames

Filenames under Unix may contain any character other than `/` ("slash") which is reserved as the directory separator. Typically, filenames only use alphabetic and numeric characters, underscores, hyphens and dots, as other characters (such as dollar signs, spaces, etc) have a special meaning to the shell and can be annoying to work with.

Unix supports long filenames, usually up to 255 characters in length (this varies from system to system).

You do not need to give Unix files an extension like `.doc` or `.exe` - the operating system can figure out what type a file is by other means. Nevertheless, it is often useful to use a suitable extension to remind yourself what files are what, and to make it easier to manipulate groups of files (for example all C programs, which usually have the extension `.c`).

Filenames which begin with a dot are hidden and do not appear in directory listings unless you specifically ask for them.

Listing files

The **ls** command is used to list files in any directory. This is analogous to DOS's **DIR** command.

```
changes.sgml              help.sgml                 shell.sgml
editing.sgml              intro.sgml                unixbasics.sgml
filesanddirectories.sgml  loggingin.sgml            usersandgroups.sgml
furtherreading.sgml       processes.sgml            whatisunix.sgml
```

Long format

To get a long-format listing, showing permissions, ownership, file size, and other information, use **ls -l**.

```
-rw-r--r--   1 skud     skud          391 Nov  2 21:12 changes.sgml
-r--r--r--   1 skud     skud        13938 Nov  9 11:09 editing.sgml
-rw-r--r--   1 skud     skud         9238 Nov  9 11:42 filesanddirectories.sgml
-r--r--r--   1 skud     skud          973 Oct 20 00:45 furtherreading.sgml
-r--r--r--   1 skud     skud        10647 Nov  9 00:36 help.sgml
-r--r--r--   1 skud     skud         2280 Nov  8 21:15 intro.sgml
-r--r--r--   1 skud     skud        10883 Nov  8 23:12 loggingin.sgml
-r--r--r--   1 skud     skud         8810 Nov  2 16:00 processes.sgml
-r--r--r--   1 skud     skud        24875 Nov  9 00:06 shell.sgml
-r--r--r--   1 skud     skud         1739 Nov  8 23:14 unixbasics.sgml
-r--r--r--   1 skud     skud        11751 Nov  2 15:51 usersandgroups.sgml
-r--r--r--   1 skud     skud         6528 Nov  8 22:25 whatisunix.sgml
```

Showing hidden files

To show all files, including hidden files (which begin with a dot), use **ls -a**.

```
.               .project        b.txt           myfile.txt      spelling.txt
..              .tcshrc         email.txt       phonelist.txt
.plan           a.txt           flavour.txt     quotes.txt
```

The file called `.` (dot) is the current directory; the file called `..` (dot dot) is the directory above the current one. You can refer to these directories in commands like **ls ..** if you wish.

You can combine the two options by using **ls -al** or **ls -la** (the order is unimportant).

```
drwxr-sr-x   2 train01  staff        1024 Nov  9 11:44 .
drwxrwsr-x  13 root     staff        1024 Nov  3 09:11 ..
-rw-r--r--   1 train01  staff          77 Nov  3 15:13 .plan
-rw-r--r--   1 train01  staff          39 Nov  3 15:19 .project
-rw-r--r--   1 train01  staff          42 Nov  5 09:42 .tcshrc
-rw-r--r--   1 train01  staff          63 Nov  3 11:01 a.txt
-rw-r--r--   1 train01  staff          40 Nov  3 11:01 b.txt
-rw-r--r--   1 train01  staff         165 Nov  5 10:17 email.txt
-rw-r--r--   1 train01  staff         769 Nov  5 10:17 flavour.txt
-rw-r--r--   1 train01  staff          89 Nov  3 11:01 myfile.txt
-rw-r--r--   1 train01  staff         136 Nov  5 10:17 phonelist.txt
-rw-r--r--   1 train01  staff        5726 Nov  3 11:04 quotes.txt
-rw-r--r--   1 train01  staff         760 Nov  5 10:17 spelling.txt
```

Readme: For other options for the **ls** command, read the manpage by typing **man ls**.

Advanced: There are only a couple of letters of the alphabet which aren't options to **ls** - can you find what they are?

Exercise

1. Practice listing files in your home directory using various output formats:
   - normal
   - long format
   - show hidden files
   - long format with hidden files
2. Read the manual page for **ls** and try some of the other command line switches

The Unix system's file structure

Unix systems have a tree-like file structure, similar to DOS or Windows systems (which were, in a roundabout way, based on Unix).

The base of this tree is the "root directory" which is referred to as `/` ("slash"). Other directories branch out of the root directory, and each level of branching is separated by a slash, in a similar way to how DOS/Windows directories are separated by a backslash (\).

Typically, the following directories can be found under the root directory of a Unix system:

Table 6-1. Unix directories and their contents

Directory name	Contents
/bin	Binary files (executables)
/etc	System-wide configuration files
/dev	Device files
/home	User home directories
/lib	Library files
/sbin	Superuser binary files
/tmp	Temporary files
/usr	Files for users
/var	Various stuff (logfiles, incoming mail spools, etc)
/opt	Optional packages, often major software packages provided by the vendor as add-ons to the system

Viewing files

The cat command

The **cat** command can be used to print a file to screen.

> **Note:** Its name is short for "catenate", which (in Unix terms) means to print out a file. "Concatenate" means to join together two or more files (or other text) to form one big one.

```
% cat mytextfile
```

> **Advanced:** The output of **cat** can be redirected into another file or piped through another command.
>
> ```
> % cat mytextfile > newtextfile
> % cat mytextfile >> newtextfile
> % cat mytextfile | mail training@netizen.com.au
> ```

Exercise

1. Try this command (there are some files in your home directory which you can use).

More or less

For files which are longer than a screenful, you can use the **more** or **less** commands to view them.

The **more** command presents the file a screenful at a time. To scroll down a page, press SPACE.

The **less** command is an extended version of **more**, so called because it allows you to scroll back in the document as well as forward. Some systems do not have the **less** command, but most modern Unix variants do have it.

Exercise

1. Try using **more** and **less** to view files in your home directory.

Copying files

To copy files, use the **cp** command. This command takes multiple arguments - all except the last argument are the file(s) you want to copy, and the last argument is where you want to copy them to. If you specify more than one file to copy (by using wildcards/globs) then you must copy them to a directory. Wildcards and globs are expanded by the shell before being passed to the **cp** command.

```
% cp myfile.txt myfile.bak
% cp myfile.txt myotherfile.txt yetanother.txt backups/
% cp *.bak backups/
```

Exercise

1. Practice copying your files using the **cp** command

Moving or renaming files

To move or rename files, use the **mv** command. This command takes arguments just like **cp**, however the names of files to be moved may be directories, in which case the entire directory and all its contents will be moved to the new location.

```
% mv myfile.txt myfile.bak
% mv myfile.bak myotherfile.bak yetanother.bak backups/
% mv *.bak backups/
% mv backups/ /tmp/
```

Exercise

1. Practice moving or renaming your files using the **mv** command

Deleting files

To delete files, use the **rm**. This command takes any number of arguments, which are the files (or glob patterns of groups of files) you wish to delete. Note that you cannot delete directories using **rm** (at least, not without using special switches); you would use the **rmdir** command (which we will encounter shortly) for that.

```
% rm *.bak
% rm *.bak backups/ /tmp/*.bak /tmp/backups/*
```

Note: There is no undelete command in Unix. Once you delete a file, you cannot get it back except by restoring from backups. Therefore, it is often a good idea to alias **rm** to **rm -i** - this will ask you for confirmation before deleting files.

Exercise

1. Practice deleting files using the **rm** command. Don't delete anything you're not certain about -- perhaps only delete files you've created yourself during the previous parts of this training module
2. Just to be on the safe side, create a shell alias so that when you type **rm** it actually runs **rm -i** and asks you for confirmation before deleting. Look back at Chapter 4 (The Shell) for details on how to do this.

Changing directories

Changing directories under Unix is similar to DOS. The command used to change directories is **cd**.

Table 6-2. Changing directories

To do this...	Syntax example
Change to a specific directory	**cd /usr/local/bin**
Change to one directory up the tree	**cd ..**
Change to two directories up the tree	**cd ../..**
Change to your home directory	**cd or cd ~**
Change to a directory under your home directory	**cd ~/backup**
Change to another user's home directory	**cd ~*username***

Exercise

1. Practice changing directory using the different forms of the **cd** command
2. Does your shell prompt show you which directory you're in? If not, look back at Chapter 4 (The Shell) to see how to configure it to show you the current directory.
3. Change to `/usr/doc` and/or `/usr/local/doc` and see what documentation is available on your system.

Finding your current directory

To find out what directory you are currently in, use the **pwd** command. This stands for "print working directory" and prints out the name of the current directory to the screen.

```
% pwd
/home/train01
```

Exercise

1. Practice using the **pwd** command.

Creating and deleting directories

To create a new directory, use the **mkdir** command.

```
% mkdir documents
```

To delete a directory, use the **rmdir** command.

```
% rmdir documents
```

> **Advanced:** Note that this will not work if the directory has files in it. If you want to delete a directory and everything in it, type **rm -rf *`dirname`***

Exercise

1. Make a directory called `backups` in your home directory and copy all your text files into it.

Chapter summary

- Under Unix, almost everything is a normal file which can be viewed and manipulated in common ways
- The **ls** command is used to list files. Command line switches can be used to modify the behaviour of this command.

- The **pwd** command can be used to show the current directory.
- The **cat** command is used to print files to the screen
- **more** and **less** are used to print files to the screen a pageful at a time.
- The **cp**, **mv** and **rm** commands can be used to copy, move/rename, and delete files respectively.
- There is no undelete command in Unix -- beware!
- The **mkdir** command can be used to create directories and the **rmdir** to remove them.

Chapter 7. Editing

In this chapter...

In this chapter you will learn how to use the **vi** editor and be introduced briefly to other editors which may be available on Unix systems.

The vi editor

The most common editor on Unix systems is **vi**. Although its commands and syntax are quite difficult to learn, it is a very powerful text editor which will allow you to write or program very efficiently. Also, the fact that vi is often the only editor on many Unix systems means it is essential to know at least the basics.

> **Readme:** The **vi** editor is described in detail in chapter 8 of the textbook, starting on page 321. You can also find more information by typing **man vi**, which may point you to additional online documentation.

To use the **vi** editor, simply type **vi *`filename`***.

You will see something like this:

```
~
~
~
foo: new file: line 1
```

... a vertical row of tildes, and on the bottom an indication of the file you are editing and some status information such as the line you are on.

Command and Insert modes

When you first get into **vi**, you are in what is known as "command mode". In this mode, you can type various commands to affect a file you're editing, or you can enter "insert mode" to insert text into the file.

To enter "insert mode", type `i`. This allows you to enter text.

Type a sentence such as "Hello, world" then press ESC to return to command mode.

> **Note:** You can always type ESC to return to command mode. It's a good idea to get into the habit of doing this if you're not sure what mode you're in.

You can see what mode you're in by typing **:set showmode** (you'll need to be in command mode to do this, so type ESC followed by a colon, et cetera. See the section on Settings, below, for more options you can set.

Table 7-1. Inserting

Command	Effect
i	Insert before cursor
I	Insert at start of line
a	Insert after cursor
A	Insert at end of line
o	Open a line below the current line
O	Open a line above the current line

Movement commands

Table 7-2. Movement commands

Command	Effect
j	Go down one line
k	Go up one line
h	Go left one character
l	Go right one character
^F	Go forward one screen
^B	Go back one screen
^	Move to beginning of text on the current line
$	Move to the end of the current line
{	Move to before the current paragraph
}	Move to after the current paragraph
G	Go to the end of the file
nG	Go to line *n*

Most movement commands can be prefaced by a number, and will be repeated that number of times. For instance, **10j** will go down ten lines.

> **Note:** While some newer systems will allow you to use arrow keys to move around in **vi**, it is nevertheless a good idea to get used to using the `hjkl` navigation keys in case you encounter a system where the arrow keys do not work. It will also allow you to edit files faster if you are a

touch typist, as you will not need to move your hands away from the home keys to navigate.

Simple text editing commands

Table 7-3. Simple text editing commands

Command	Effect
x	Delete the character under the cursor
dd	Delete the current line
dw	Delete to the end of the current word
u	Undo last command
.	Repeat last command

Like the movement commands, most of these simple text editing commands can be made to repeat by prefacing the command with a number. For instance, **6dd** will delete six lines.

Searching and replacing

Table 7-4. Searching and replacing

Command	Effect
/*string*	Search for *string*
n	Find next occurrence of *string*
N	Find previous occurrence of *string*
:s/*string/replacement*	Replace first instance of *string on this line with replacement*
:s/*string/replacement/g*	Replace all instances of *string on this line with replacement*
:%s/*string/replacement/g*	Replace all instances of *string in the entire file with replacement*

Cutting and pasting

The clipboard in **vi** is referred to as the "buffer". Text which is cut or copied is stored in the buffer until another cut or copy operation overwrites it. You can paste as often as you want from the buffer.

The **dd**, **dw** and **x** commands from the previous section all place the deleted text into the buffer. They

are equivalent to "cut".

In **vi**, "copying" is known as "yanking" text into the buffer. To yank text, the following commands may be used:

Table 7-5. Yanking

Command	Effect
yy	Yank the current line into the buffer
yw	Yank the word under the cursor into the buffer

To paste the contents of the buffer, the following commands may be used:

Table 7-6. Pasting

Command	Effect
p	Paste after the cursor
P	Paste before the cursor

Some fun and useful commands

Table 7-7. Some fun and useful commands

Command	Effect
~	Change the case of a character (upper case becomes lower case, and vice versa)
J	Join two lines together

Saving and quitting

Table 7-8. Saving and quitting

Command	Effect
:w	Write (save) the file
:w ***filename***	Write (save) as ***filename***
:wq	Write and quit
:q!	Quit without writing

Settings

There are a number of settings you can set either from command mode or from a configuration file called `.exrc` which lives in your home directory.

Exercise

1. Open the file `learnvi.txt` in **vi** and use it to practise using the editor

Variants of vi

There are variants of **vi** such as **vim**, **nvi** and **elvis**, all of which provide additional functionality on top of the basic **vi** commands.

Other editors

Here are some brief instructions for other editors which you may find on Unix systems, in the form of brief "cheat sheets" for each one.

Table 7-9. Layout of editor cheat sheets

Running	Recommended command line for starting it.
Using	Really basic howto. This is not even an attempt at a detailed howto.
Exiting	How to quit.
Gotchas	Oddities to watch for.

emacs

Running

```
% emacs
```

Using

Complicated; see the "Help" facility by typing **CTRL**-**H**.

Exiting

CTRL-**X** then **CTRL**-**C**.

Gotchas

This editor takes a lot of system resources to run, as it contains many extra functions other than text editing, and a full-blown programming language for writing new functions.

> **Readme:** The **emacs** editor is described in chapter 7 of the textbook, starting on page 302.

pico

Running

```
% pico -w filename
```

Using

- Cursor keys should work to move the cursor.
- Type to insert text under the cursor.
- The menu bar has ^X commands listed. This means hold down **CTRL** and press the letter involved, eg **CTRL**-**W** to search for text.
- **CTRL**-**O** to save.

Exiting

Follow the menu bar, if you are in the midst of a command. Use **CTRL**-**X** from the main menu.

Gotchas

Line wraps are automatically inserted unless the -w flag is given on the command line. This often causes problems when strings are wrapped in the middle of code and similar. \\ \hline

Help

CTRL-**G** from the main menu, or just read the menu bar.

joe

Running

```
% joe filename
```

Using

- Cursor keys to move the cursor.
- Type to insert text under the cursor.
- **CTRL**-**K** then **S** to save.

Exiting

- **CTRL**-**C** to exit without save.
- **CTRL**-**K** then **X** to save and exit.

Gotchas

Nothing in particular.

Help

CTRL-**K** then **H**.

jed

Running

```
% jed
```

Using

- Defaults to the emacs emulation mode.
- Cursor keys to move the cursor.
- Type to insert text under the cursor.
- **CTRL**-**X** then **S** to save.

Exiting

CTRL-**X** then **CTRL**-**C** to exit.

Gotchas

Nothing in particular.

Help

- Read the menu bar at the top.
- Press **ESC** then **?** then **H** from the main menu.

Chapter summary

1. The **vi** editor is the most common text editor under Unix and can be found on most systems.
2. Other editors such as **emacs**, **pico** and others can be found on some systems.

Chapter 8. Users, groups and permissions

In this chapter...

As mentioned in Chapter 2 (What is Unix?), Unix is a multi-user operating system. This chapter deals with Unix's concepts of users, groups of users, and the permissions which allow them to read and modify files, and run programs.

Users

The /etc/passwd file

Users on a Unix system are listed in the `/etc/passwd` file. This is the file which is looked at when you log in, to make sure that you are a valid user of the system. Each line of the `/etc/passwd` file looks something like this:

```
fred:0xw/4Pj0nYrBc:1000:1000:Fred Foonly:/home/fred:/usr/bin/tcsh
```

The fields in the `/etc/passwd` file are separated by colons (:). In order, they are:

Table 8-1. Fields in `/etc/passwd`

Field	Description
username	the username which the user uses to login
password	the user's password (in encrypted form)
uid	the user ID (a numeric identifier for each user)
gid	the user's default group ID (groups are discussed later)
GECOS field	textual information about the user (usually including the user's real name)
home directory	the user's home directory, where they will find themselves directly after logging in
default shell	the shell to run when the user logs in

who, whoami and w

You can find out what users are logged into a system by using the **who** command.

```
% who
skud     ttyp0    Oct 19 23:07 (stan29.zip.com.au)
penny    ttyp3    Oct 19 23:43 (vurt.net)
thorfinn ttyp5    Oct 19 21:48 (mycroft.tertius.net.au)
ajc      ttyp6    Oct 14 13:47 (203.18.243.205)
```

The output of this command shows who's logged in, what virtual terminal they are connected to, when they logged in, and what system they're logged in from.

You can also type **who am i** to find out who you are logged in as (the "am i" is a special kind of argument to **who**).

```
% who am i
vitaly!skud     ttyp0    Oct 19 23:07 (stan29.zip.com.au)
```

Note that the system name is also given in this output; in this case, "vitaly".

The command **whoami** is different from **who am i**, and simply shows you your user name:

```
% whoami
skud
```

The **w** shows a list of users logged in and what they're doing:

```
% w
3:43pm  up 8 days,  4:33,  6 users,  load average: 1.05, 0.79, 0.46
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU  WHAT
skud     ttyp0    :0               Sat 8pm  1.00s 15.06s  0.21s  w
skud     ttyp3    :0               Sun12pm  9:28   3.34s  3.13s  ssh skud hiro.n
skud     ttyp5    :0               Mon10am  8.00s  0.91s  0.09s  man w
```

Exercise

1. Practice using the **who**, **whoami** and **w** commands.

The finger command

The **finger** gives more information about a user. As well as showing information from the **/etc/passwd** file and whether (and how many times, and from where) they're logged in, it can print out the contents of certain files the user has created to give more information about themself.

```
% finger skud
Login: skud                             Name:
Directory: /home/skud                   Shell: /usr/bin/tcsh
On since Tue Oct 19 23:07 (EST) on ttyp0 from stan29.zip.com.au
   1 second idle
On since Tue Oct 19 22:01 (EST) on ttypf from stan29.zip.com.au
   12 minutes 36 seconds idle
New mail received Thu Aug 12 11:51 1999 (EST)
```

```
     Unread since Mon Aug  9 11:54 1999 (EST)
No Plan.
```

If you create a file called `.plan` (remembering that the leading dot makes it a hidden file) you can enter any text you like and have it show up when someone fingers you:

```
% finger skud
Login: skud                             Name:
Directory: /home/skud                   Shell: /usr/bin/tcsh
On since Tue Oct 19 23:07 (EST) on ttyp0 from stan29.zip.com.au
   1 second idle
On since Tue Oct 19 22:01 (EST) on ttypf from stan29.zip.com.au
   12 minutes 36 seconds idle
New mail received Thu Aug 12 11:51 1999 (EST)
     Unread since Mon Aug  9 11:54 1999 (EST)
Plan:
To finish writing the Unix Basics training course, then go to sleep.
```

Likewise, a file called `.project` is often used to describe a user's current project:

```
% finger skud
Login: skud                             Name:
Directory: /home/skud                   Shell: /usr/bin/tcsh
On since Tue Oct 19 23:07 (EST) on ttyp0 from stan29.zip.com.au
   1 second idle
On since Tue Oct 19 22:01 (EST) on ttypf from stan29.zip.com.au
   12 minutes 36 seconds idle
New mail received Thu Aug 12 11:51 1999 (EST)
     Unread since Mon Aug  9 11:54 1999 (EST)
Project:
Training for Netizen

Plan:
To finish writing the Unix Basics training course, then go to sleep.
```

You can also finger users on remote systems by using **finger user@their.system.name**, however many systems are set up to refuse finger requests from remote systems.

Exercise

1. Create `.plan` and `.project` files in your home directory. Use the **finger** command to see their output.

Groups

Groups of users, such as "staff" or "web-users" or "admin" can be created by the system administrator. These are listed in the `/etc/group` file. Groups are used to set permissiosn for files, so that (for instance) only staff can get at documents in a special staff directory.

The /etc/group file

The `/etc/group` contains lines which look like this:

```
staff:x:100:skud,fred,mary,bill
```

The fields are group name, password (usually not used), numeric group ID, and a comma-separated list of the users who are in that group.

The groups command

To find out which groups you are a member of, use the **groups** command.

```
% groups
skud staff www admin
```

Exercise

1. Use the **groups** to see what groups you're in.

Permissions

Finding a file's permissions

A file's permissions can be found by using the long form of the **ls**, i.e. **ls -l**:

```
-rw-r--r--   1 skud     skud          391 Nov  2 21:12 changes.sgml
-r--r--r--   1 skud     skud        13938 Nov  9 11:09 editing.sgml
-rw-r--r--   1 skud     skud         9238 Nov  9 11:42 filesanddirectories.sgml
-r--r--r--   1 skud     skud          973 Oct 20 00:45 furtherreading.sgml
-r--r--r--   1 skud     skud        10647 Nov  9 00:36 help.sgml
-r--r--r--   1 skud     skud         2280 Nov  8 21:15 intro.sgml
-r--r--r--   1 skud     skud        10883 Nov  8 23:12 loggingin.sgml
-r--r--r--   1 skud     skud         8810 Nov  2 16:00 processes.sgml
-r--r--r--   1 skud     skud        24875 Nov  9 00:06 shell.sgml
-r--r--r--   1 skud     skud         1739 Nov  8 23:14 unixbasics.sgml
-r--r--r--   1 skud     skud        11751 Nov  2 15:51 usersandgroups.sgml
```

```
-r--r--r--   1 skud     skud         6528 Nov  8 22:25 whatisunix.sgml
```

There are ten characters at the start of each line which describe the file's type and permissions.

The first character shows whether the item listed is a directory or a symbolic link. If it is a directory, the letter `d` will appear there. If it is a link, the letter `l` will appear there. Otherwise it will simply show a hyphen, meaning that the file is just a normal file.

The next nine characters are in three groups of three. The groups are "owner", "group", and "world" and describe the permissions each of those groups of users has with regard to the file. The permissions that they can have are any of "read", "write" and "execute".

The owner of a file is the person whose username appears in the fourth column of the **ls -l** output. The group of a file is the group name which appears in the fifth column of the output.

"Read" permission means that someone may look at the contents of a file but not change them.

"Write" permission means that someone may change the contents of a file, or may rename it, delete it, etc.

"Execute" permission means that a someone has the ability to run that file as an executable program.

Let's look at an example:

```
-rw-------   1 fred     staff       84354 Oct 17 21:09 myfile.txt
-rw-r-----   1 fred     staff        4705 Aug  1 15:37 meeting_minutes.txt
drwxr-xr-x  15 fred     staff        2048 Oct 28 15:00 public_html
```

The first file, `myfile.txt`, is readable and writable by Fred and nobody else.

The second file, meeting_minutes.txt, is readable and writable by Fred, and also readable by any other user in the `staff` group.

The third file is actually a directory, and is probably the directory containing Fred's personal homepage (`public_html` is the customary name for a directory for a user's homepage). It is readable and executable by everyone (fred, his group, and "other") because a directory must be executable to allow people to change into that directory. It is also writable by Fred, so that he can create new files in that directory.

Changing a file's permissions

A file's permissions are changed by using the **chmod** command. This command has two forms of syntax, one of which uses easily-remembered abbreviations, and one of which uses octal (base-8) notation for permissions.

The easier form uses the following letters for the groups of users to whom permissions apply:

Table 8-2. Abbreviations for groups

User	`u`

Group	`g`
Other	`o`
All	`a`

Table 8-3. Abbreviations for permissions

Readable	`r`
Writable	`w`
Executable	`x`

Here are some examples of usage:

Make a file readable by everyone (the following two commands are equivalent):

```
% chmod a+r myfile.txt
% chmod +r myfile.txt
```

Make a file writable by the user and group:

```
% chmod ug+w myfile.txt
```

Remove execution permissions for group and other:

```
% chmod go-x myscript.sh
```

> **Advanced:** The octal form of this command uses three octal digits to represent the states of the owner, group and world permissions. "Readable" is equal to 4, "writable" is 2, and "executable" is 1. Simply add together the permissions for each group.
>
> For instance, to make a file readable and writable by the owner only, use **chmod 600 myfile.txt**. To make it readable and executable by everyone, but only writable by the owner, **chmod 755 myfile.txt**.

Chapter summary

1. Users on a Unix system are specified in the `/etc/passwd` file
2. The **whoami** shows you who you're logged in as.
3. The **who** and **w** commands can show you who's logged in to the system at the moment.

4. The **finger** command shows you more information about users on the system (or on another system connected to the network)
5. You can create files called `.plan` and `.project` to customise your **finger** output
6. Groups of users are specified in the `/etc/group` file
7. You can find out what groups you are in by using the **groups** command
8. A file can have read, write and execute permissions set for each of "user", "group" and "other"
9. A file's permissions can be changed using the **chmod** command

Chapter 9. Processes

In this chapter...

Multi-tasking operating systems such as Unix are able to run more than one program (or "process") simultaneously. This chapter explores the commands used for managing processes under Unix.

What is a process?

As mentioned earlier, Unix is a "multi-tasking" operating system. This means that it can run many programs or applications simultaneously.

Unix refers to each currently-running program as a "process". Every process has a process id (pid) which is a number. The first process to run when the system starts up is process number 1, and subsequent processes are given consecutively larger numbers.

Every process running is associated with a user -- usually the user who started the process. Processes may also have a "controlling terminal" -- the virtual terminal from which the process started, and to which its input and output are attached. However, some processes do not have a controlling terminal; these are called "background processes" and we'll look at them later.

Listing processes - the ps command

You can use the **ps** command to list processes. Here is a sample process listing:

```
yt:~> ps
  PID TTY STAT TIME COMMAND
  12539  p3 S    0:00 -csh
  12835  p0 S    0:00 -csh
  13072  p4 S    0:00 -csh
  13141  p5 SW   0:00 (tcsh)
  13198  p3 S    0:01 vim processes.sgml
  13203  p6 S    0:00 -csh
  13204  p6 R    0:00 ps
```

The **ps** without any switches or other arguments shows processes which are being run by you, which have a controlling terminal. (We'll find out about processes with no controlling terminal later.) The PID listed is the "process ID" -- a numeric identifier which uniquely identifies each process running on the system.

Exercise

1. Use the **ps** command to see what processes you have running. Can you identify them all?

Command line switches to ps

The command line switches to **ps** vary significantly between different flavours of Unix. However, since most systems support BSD style switches, those are the ones which will be described here.

Table 9-1. BSD-style command line switches for ps

Switch	Meaning
`l`	long format
`a`	show all processes (not just your own)
`u`	user format; shows username and start time
`x`	show processes with no controlling terminal
`w`	wide listing (`ww makes it even wider`)

The following process listing format is very common, but tends to produce a lot of output. Pipe its output through **less** or some other filter to ensure that you can see important information.

```
% ps auxw
USER       PID %CPU %MEM  SIZE   RSS TTY STAT START   TIME COMMAND
daemon     150  0.0  0.0  1104     0  ?  SW  Oct 25   0:00 (portmap)
mail     14902  0.0  0.0  1712     0  ?  SW  Oct 26   0:01 (sendmail)
root         1  0.0  0.1  1016    72  ?  S   Oct 25   0:03 init [2]
root         2  0.0  0.0     0     0  ?  SW  Oct 25   0:13 (kflushd)
root         3  0.0  0.0     0     0  ?  SW  Oct 25   0:11 (kupdate)
root         4  0.0  0.0     0     0  ?  SW  Oct 25   0:00 (kpiod)
root         5  0.0  0.0     0     0  ?  SW  Oct 25   0:31 (kswapd)
root       120  0.0  0.5  1348   212  ?  S   Oct 25   0:05 /sbin/syslogd
root       122  0.0  0.0  1288     0  ?  SW  Oct 25   0:00 (klogd)
root       142  0.0  0.1   996    48  ?  S   Oct 25   0:00 /sbin/kerneld
root       154  0.0  0.0  1304     0  ?  SW  Oct 25   0:00 (inetd)
root       182  0.0  0.1  1020    56  ?  S   Oct 25   0:02 /usr/sbin/gpm -
m /dev/psaux -t ps2 -l "a-zA-Z0-9_.:~/\300-\326\330-\366\370-\377"
root       245  0.0  0.0  1904     0  ?  SW  Oct 25   0:00 (safe_mysqld)
root       252  0.0  0.1  4040    44  ?  S N Oct 25   0:00 (mysqld)
root       261  0.0  0.1  2180    52  ?  S   Oct 25   0:01 /usr/sbin/apache
root       272  0.0  0.0  1136     0   2 SW  Oct 25   0:00 (getty)
```

Advanced: The most popular Unix variant which uses completely different switches for **ps** is Solaris. To find out about the command line switches on that (or any other) flavour of Unix, type **man ps**. Note that a BSD-style **ps** is available on Solaris systems in `/usr/ucb/ps`

Exercise

1. Experiment with the effects of some of the command line switches
2. If the output is too long for you to read easily, pipe it through the **less** pager by typing **ps *switches* | less**

Killing processes

You can kill any process which you own (but not those owned by other users) by using the **kill** command:

```
% ps
  PID TTY STAT TIME COMMAND
  12539  p3 SW   0:00 (tcsh)
  13141  p5 SW   0:00 (tcsh)
  13198  p3 S    0:07 vim processes.sgml
  13279  p6 R    0:00 ps
% kill 13198
% ps
  PID TTY STAT TIME COMMAND
  12539  p3 SW   0:00 (tcsh)
  13141  p5 SW   0:00 (tcsh)
  13279  p6 R    0:00 ps
```

You can send other signals to processes using the **kill** command's optional switches. You can give an signal name as an optional switch, or a signal number. Some of the common signal names and numbers are listed in the manual page for **kill**, or you can get a list of available signal names with **kill -l**

```
% kill -l
HUP INT QUIT ILL TRAP ABRT BUS FPE KILL USR1 SEGV USR2 PIPE ALRM TERM STKFLT
CHLD CONT STOP TSTP TTIN TTOU URG XCPU XFSZ VTALRM PROF WINCH POLL PWR UNUSED
```

The most commonly used signal is KILL, which forces a process to be killed even if a normal **kill** command cannot kill it. This is often useful if a program has "hung" and will not respond to anything. The signal number for KILL is 9, therefore the following two commands are equivalent:

```
% kill -KILL 13198
% kill -9 13198
```

Exercise

1. Try killing some of your processes (and don't be surprised if it logs you out when you kill your own shell!)
2. Can you kill other people's processes?

Background processes and job control

Running a process in the background

On Unix, it is possible to run processes in the background. This means that the process becomes detached from its controlling terminal and runs more-or-less invisibly to the user.

To start a process running in the background, simply put an ampersand (&) after the command on the command line.

```
% cp -r * /tmp &
[1] 13319
```

Exercise

1. Run a **vi** editor session in the background

Listing running jobs

To get a listing of running jobs, use the **jobs** command.

```
% jobs
[2]  + Suspended (tty output)        top
[3]  - Running                       tar -c /www > www.tar
```

> **Note:** Note that a job is not the same as a process. A job may have many different processes associated with it. For instance, if you create a pipeline on the command line (that is, connect multiple programs together using pipes) there will be several processes running, but they constitute only one job.

Exercise

1. List your currently-running jobs. Do you see your **vi** session there?

Bringing a backgrounded process to the foreground

As you can see, when you run a program in the background it will tell you the process ID it has been given, and also a job number.

> **Note:** The Bourne Shell (**sh**) does not have job control; this section assumes you are using a more featureful shell such as **bash**, **tcsh**, **csh** or **ksh**.

This job number allows you to bring a job to the foreground using the **fg** command. The following command would bring job number 1 to the foreground:

```
% fg %1
```

Exercise

1. Bring your **vi** session to the foreground using **fg**

Backgrounding an already-running job

Lastly, if you wish to send an already-running program to the background, you can do this by first pressing **CTRL**-**Z** to suspend it, then typing the **bg** command:

```
% yt:~/work/training/unixtools> tar -c /www > www.tar
```

(CTRL-Z is pressed at this point)

```
Suspended
% bg
[3]    tar -c /www > www.tar &
```

Exercise

1. Put your **vi** session into the background using **CTRL**-**Z** and **bg**

Killing a running job

You can also kill a running job with the **kill** by giving it the job number as an argument:

```
% kill %3
[3]  - Terminated                  tar -c /www > www.tar
```

Exercise

1. Kill off your backgrounded **vi** session using **kill**
2. How else could you have killed the process?

Chapter summary

1. Processes can be listed using the **ps** command.
2. A job can be started in the background by adding an ampersand (`&`) to the end of the command line
3. Running jobs can be listed using the **jobs** command
4. A backgrounded job can be foregrounded using **fg %*jobnumber***
5. An already-running job can be backgrounded using **CTRL**-**Z**
6. A process can be killed using the **kill *process-id*** command
7. A job can be killed using the **kill %*jobnumber*** command

Chapter 10. About Linux

What is Linux?

Linux is an operating system based on Unix, developed by a Finnish man named Linus Torvalds and a worldwide community of developers.

Linux is released under an "open source" license called the GPL (GNU Public License), which allows the licensee to:

- Obtain the software for no cost, or for the cost of media (eg CD copying)
- Likewise obtain source code to the software
- Modify the software
- Redistribute the software, providing that it retains the same license

Linux has a significant minority share of the operating system market, and is expected by many industry analysts to grow to a majority share in the next few years.

The Linux operating system itself contains very few tools, applications, or user-friendly elements. For this reason, many companies or non-profit ventures have chosen to package the Linux operating system with a collection of other software. These packaged software collections are called "Linux distributions", and include:

- Redhat Linux (http://redhat.com/)
- Debian/GNU Linux (http://debian.org/)
- Caldera OpenLinux (http://www.calderasystems.com/)
- TurboLinux (http://www.turbolinux.com/)
- SuSE Linux (http://www.suse.com/)

In total, there are over a hundred Linux distributions. Each targets a different part of the market. For instance, Caldera OpenLinux is renowned as being very easy to use; TurboLinux specialises in supporting Asian character sets such as Japanese; SuSE is very popular in Europe; Redhat target corporate desktop users; Debian/GNU Linux requires all its software to meet certain licensing standards and tends to appeal to more technical users.

Hardware requirements

Linux can run on many kinds of hardware, including Intel-based PCs, Macintosh hardware, DEC/Compaq Alpha hardware, and more. This section deals only with Intel-based PC hardware.

Although people will tell you that Linux will run on just about any hardware, in real terms you need at least a 386, or preferably a 486 or better. To use any kind of graphical user interface, you will want a high-end 486 or a low-end Pentium.

Recommended: Pentium 100MHz or better

In terms of RAM, you will probably want at least 8Mb for a non-graphical setup, or 32Mb for a setup with a graphical user interface. You can run a graphical interface on less memory, but it will be painfully slow.

Recommended: 32-64Mb of memory

A full Linux distribution will usually require about a gigabyte of disk space, but you seldom need to install a full distribution. A very cut-down system may only take 100Mb of disk, or a reasonably usable system with a GUI and quite a lot of applications may take about 500Mb. You can install Linux on a second hard disk or disk partition to make a "dual boot" machine that runs both Linux and Windows or another operating system.

Recommended: 1Gb of hard disk space

In addition to this basic hardware, you will probably also want:

- A 3.5" floppy drive
- A CD-ROM drive
- A modem or network card (for Internet access or LAN use)
- A 3-button mouse (you can use a 2-button mouse, but 3-button is better)
- A keyboard (any kind)
- A monitor and graphics card (note that some of the newest accellerated video cards are not yet supported by Linux)

Getting Linux

Linux can be downloaded from the Internet "for free", but keep in mind that you will have to pay for connection time or for the data you download, that that a normal Linux distribution can be 600Mb or more. You would also need a CD burner to create a CD of the downloaded data.

It is almost always easiest to buy a Linux CD in one of various forms:

- Cheap CDs, paper sleeve, no instructions (about $2-$5)
- "Box set" versions of Linux, usually including CDs with full instructions in the form of a book (about $50 or more)
- "Pocketbook" distributions, for example the APC Pocketbook, containing CDs and briefer instructions (about $20 for 2 distributions)

These can be obtained from:

- Specialist Linux businesses such as EverythingLinux (http://www.everythinglinux.com.au) or Linux System Labs (http://www.lsl.com.au)
- Technical bookshops
- Computer swapmeets

Most specialist Linux businesses will take orders online and will accept credit cards or COD.

Getting help

Linux can be daunting for a new user. When you first install Linux, and for your first six months or more of use, you will probably want some help.

Help is available from various places, including:

Linux User Groups (LUGs)

Many Linux User Groups exist around Australia and the world. Some Australian ones include:

- Linux Users of Victoria (LUV) (http://www.luv.asn.au/)
- Sydney Linux Users Group (SLUG) (http://www.slug.org.au/)
- Home Unix Machine Brisbane Users Group (HUMBUG) (http://www.humbug.org.au/)

A list of all Australian Linux User Groups can be found on Linux Australia (http://www.linux.org.au/) website. Linux.org also has a list of LUGs around the world (http://www.linux.org/users/index.html).

Linux consultants and businesses

Many businesses now offer Linux support and consulting. Netizen (http://netizen.com.au/) is one of these businesses. Others are listed on the Linux Australia (http://www.linux.org.au/) website.

Online resources

A wealth of online resources are available. Some good starting points include:

- Linux.org (http://linux.org)
- Linux Weekly News (http://lwn.net)
- Linuxnewbie.org (http://linuxnewbie.org)

Chapter 11. Conclusion

What you've learnt

Now you've completed Netizen's Unix Basics module, you should be confident in your knowledge of the following fields:

- What is Unix? Unix features, different flavours of Unix.
- Logging in, changing your password, logging out.
- The Unix shell -- its purpose, features, and use
- Finding help on Unix systems, including manual pages and other sources of information
- Files and directories; listing files in various formats; copying, moving, renaming and deleting files and directories; viewing files in various ways Editing text files using the **vi** editor
- The concept of users and groups under Unix; how to find information about users and groups
- Showing and setting file permissions under Unix
- Using Unix's processes and job control; listing processes and jobs; running processes and jobs in the background; killing processes and jobs
- The Linux operating system; Linux's license benefits; hardware required to run Linux; where to get Linux; where to get help.

Where to now?

To further extend your knowledge of the Unix and related technologies, you may like to:

- Borrow or purchase the books listed in our "Further Reading" section (below)
- Follow some of the URLs given throughout these course notes, especially the ones marked "Readme"
- Practice using Unix from day to day
- Set up a home Linux box and start playing with it
- Join a Linux or Unix user group
- Extend your knowledge with further Netizen courses such as:
  - Unix Tools (you are probably taking this module tomorrow)
  - Perl programming
  - Advanced Unix

Information about these courses can be found on Netizen's website (http://netizen.com.au/services/training/). A diagram of Netizen's courses and the careers they can lead to is included with these training materials.

Further reading

Unix history and culture

"The evolution of the Unix Time-sharing System", Dennis M. Ritchie, http://cm.bell-labs.com/cm/cs/who/dmr/hist.html

Online Unix tutorials

"Unixhelp for Users" from the University of Edinburgh, at http://unixhelp.ed.ac.uk/

Norm Matloff's "Unix Tutorial Center" at the University of California at Davis, at http://heather.cs.ucdavis.edu/~matloff/unix.html

"A Basic Unix Tutorial" from Idaho State University, at http://www.isu.edu/departments/comcom/unix/workshop/unixindex.html

Appendix A. ASCII Pronunciation Guide

Table A-1. ASCII Pronunciation Guide

Character	Pronunciation
!	bang, exlamation
*	star, asterisk
$	dollar
@	at
%	percent
&	ampersand
"	double-quote
’	single-quote, tick
( )	open/close bracket, parentheses
<	less than
>	greater than
-	dash, hyphen
.	dot
,	comma
/	slash, forward-slash
\	backslash, slosh
:	colon
;	semi-colon
=	equals
?	question-mark
^	caret (pron. carrot)
_	underscore
[ ]	open/close square bracket
{ }	open/close curly brackets, open/close brace
|	pipe, or vertical bar
~	tilde (pron. “til-duh”, wiggle, squiggle)
‘	backtick