Unix Tools

Kirrily Robert
Training co-ordinator
Netizen Pty Ltd

Unix Tools

by Kirrily Robert

Revision History

Revision 0.11999-10-19Revised 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

Table of Contents

1. Introduction 7

About this module 7
Course outline 7
Assumed knowledge 7
Module objectives 7
Materials 8
Platform and version details 8
The course notes 9

2. Why learn Unix tools? 10

3. Tools for manipulating text 11

In this chapter... 11
Formatting text with **fmt** 11
Counting words, lines and characters with **wc** 13
Checking spelling with **ispell** 14
Comparing files with **diff** 15
Sorting text files with **sort** 16
Finding unique lines in a text file with **uniq** 18
Heads and tails - viewing parts of a text file 19
Chapter summary 21

4. Tools for compression and archiving 22

In this chapter... 22
Compressing (zipping) files with **gzip** 22
Viewing compressed files with **zcat** 23
Archiving files with **tar** 23
Chapter summary 25

5. Tools for finding things 26

In this chapter... 26
Using **whereis** and **which** to find programs 26
Finding files with **locate** and **find** 26
Using **xargs** to act on found files 28
Matching text in files with **grep** 28
Chapter summary 29

6. System and network information tools 31

In this chapter... 31
Finding the operating system with **uname** 31
Viewing performance statistics with **uptime** 31
Viewing processes and resource usage with **top** 32
Finding the name of the system with **hostname** 32
Looking up a system’s IP number or name using **nslookup** 33
Seeing if a network-connected system is reachable using **ping** 34

Checking the route to a networked system using **traceroute** 35
Chapter summary 35

7. Date and time tools 37

In this chapter... 37
Printing the date and time with **date** 37
Viewing a calendar with **cal** 37
Running a job at a given time with **at** 39
Running a job at a regular intervals with **crontab** 40
Some notes about Year 2000 compliance 40
Chapter summary 41

8. Tools for communication 42

In this chapter... 42
Sending messages to other users with **write** 42
Real-time chat with **talk** 42
Sending and receiving electronic mail 43
Chapter summary 45

9. Internet tools 47

In this chapter... 47
Transferring files with **ftp** 47
World Wide Web tools 49
Chapter summary 50

10. Using shell scripts to write your own tools 52

In this chapter... 52
What is a shell script? 52
Simple shell scripting commands 52
Running shell scripts 53
Conditional and looping constructs 53
Chapter summary 54

11. Conclusion 55

What you’ve learnt 55
Where to now? 55
Further reading 56

A. ASCII Pronunciation Guide 57

List of Tables

3-1. **ispell** keystroke commands 14
4-1. **tar** actions 23
8-1. **mail** commands 44
9-1. Some useful commands in the FTP client 48
A-1. ASCII Pronunciation Guide 57

Chapter 1. Introduction

About this module

This training module ("Unix Tools") teaches a range of tools and utilities found on the Unix operating system, as well as how to combine them together to perform more complex activities.

Course outline

- Why learn Unix tools?
- Tools for manipulating text
- Tools for compression and archiving
- *Morning tea break*
- Tools for finding things
- *Lunch break (60 minutes)*
- System and network information tools
- Tools for communication
- *Afternoon tea break*
- Internet tools
- Shell scripting

Assumed knowledge

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

You should have already attended Netizen’s *Unix basics* training module, or have equivalent prior knowledge of the Unix operating system.

Module objectives

- Understand the philosophy behind Unix tools, how they interact, and why they are useful
- Recognise and use tools for manipulating text files, including formatting, word counting, spell checking, finding differences, sorting, finding unique entries, viewing parts of a file, etc
- Understand Unix compression and archiving tools including **tar** and **gzip**, and how to view compressed files with **zcat**.
- Understand various files for finding programs, files and text within files; know how to perform subsequent actions on files that are found
- Recognise and use system information tools: find the operating system version; view performance statistics, processes and resource usage; find the network name of the system; test network connectivity; look up IP addresses or hostnames
- Understand and use date and time tools: print the current date in various formats; print calendar output for any month or year; run jobs at a pre-determined time or at regular intervals; understand how Unix represents dates internally and the effect on Y2K compliance
- Use communications tools on the system: send messages to other users; perform real-time chat; send and receive electronic mail
- Recognise and use Internet tools: transfer files with FTP; use command-line web browsers and other web user agents
- Understand the use of shell scripts to write custom tools
- *Understand and use shell redirection and piping to combine Unix tools for greater power and efficiency*

Materials

The materials for this module include:

- These training notes
- A copy of the textbook, Unix in a Nutshell
- 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. Why learn Unix tools?

The Unix philosophy is very different to that of PC systems such as Microsoft Windows or MacOS. On a Unix system, you will not often find huge integrated applications which perform many functions within each program. Instead, Unix comes with a large collection of small tools.

When creating Unix tools, the motto is "do one thing and do it well". These small, efficient tools can then be connected together through shell redirection and pipelines (as taught in Netizen’s Unix Basics training module) or by creating customised scripts which do what you want.

The way to become a Unix power user is to learn what tools are available, and how to combine them together in effective ways to achieve your goals. This training module helps you achieve this by introducing tools of various kinds and progressively tying them together.

Chapter 3. Tools for manipulating text

In this chapter...

In this chapter, we’ll discover some of the tools that can be used to manipulate ordinary text files under Unix. Many of these are similar to the functions built into word processing applications or office suites under Windows or other PC operating systems, but in Unix they are separate, standalone tools which can be tied together in many different ways.

Formatting text with fmt

Imagine that you’ve got a text file containing a letter or other paragraph-based text. Because most Unix text editors don’t linewrap by default, some of the lines are too long or too short, as in this example text from the Unix Basics training module:

```
% cat flavour.txt
Unix is not a single operating system.  Rather, it is a family of operating sys-
tems 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) or
"BSD-like" (mostly similar to the BSD Unix variants developed at
UC Berkeley).

There are both commercial variants of Unix (such as Sun’s Solaris,
IBM’s AIX, 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 applica-
ble to most Unix variants.  When things differ between different Unix
flavours, your trainer will point them out and may demonstrate the
difference to you.
```

Or, perhaps you have multiple files with different formattings that you want to combine into one consistently formatted file.

You can make this messy text neatly formatted using the **fmt** command. This will take any number of filenames as arguments and concatenate them together into neatly-formatted output. Lines will be wrapped at 75 characters (to fit into the default Unix terminal width of 80 characters).

```
% fmt flavour.txt
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) or "BSD-like"
(mostly similar to the BSD Unix variants developed at UC Berkeley).

There are both commercial variants of Unix (such as Sun’s Solaris, IBM’s
AIX, 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.

%
```

You can specify a different width by using the `-w` command line switch.

```
% fmt -w 50 flavour.txt
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) or "BSD-like"
(mostly similar to the BSD Unix variants
developed at UC Berkeley).

There are both commercial variants of Unix
(such as Sun’s Solaris, IBM’s AIX, 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.

%
```

Output is sent to STDOUT, and can therefore be redirected to another file using standard shell redirection techniques:

```
% fmt flavour.txt > flavour_formatted.txt
%
```

Readme: Other command line switches can be found by reading the manual page using **man fmt**.

Exercises

1. Experiment with using **fmt** to format some of the text files in your home directory.

Counting words, lines and characters with wc

If you’re writing an essay or article (or set of training materials!) you might want to know how many words you’ve written. Or, if you’re programming, you might be interested in how many lines long your programs are, or just the size of a file in characters.

Under Windows, this function would be built into a word processing application. On Unix systems, a separate **wc** command can be used to count words, lines or characters in all kinds of files (not just word processor documents).

This example shows the basic use of **wc**, using the files which contain the text of the Unix Basics training course:

```
% wc *.sgml
     16      17     391 changes.sgml
    500    1257   11580 editing.sgml
    361    1207    9238 filesanddirectories.sgml
     32      74     973 furtherreading.sgml
    210     746    5777 help.sgml
     77     239    2281 intro.sgml
    264    1069    7524 loggingin.sgml
    284    1272    8810 processes.sgml
    770    2898   22212 shell.sgml
     85     143    1743 unixbasics.sgml
    411    1569   11751 usersandgroups.sgml
    174     862    5779 whatisunix.sgml
   3184   11353   88059 total
```

The three columns show the number of lines, words, and characters in each file. The last line in the output shows the total of each.

There are three flags which can be used to specifically show the characters, words, or lines in a file:

```
% wc -c shell.sgml
  22212 shell.sgml
% wc -w shell.sgml
   2898 shell.sgml
% wc -l shell.sgml
    770 shell.sgml
```

You can also pipe output of any other command through **wc**. Often, you’ll use **wc -l** to find out how many lines of output another command produces:

```
% who | wc -l
     15
```

```
% ls /usr/bin | wc -l
   1103
```

Exercises

1. Use **wc** to find out how many words, lines and characters are in the file `flavour.txt`
2. How many files are there in the `/etc` directory?

Checking spelling with ispell

Many systems have a spell-checking program called **ispell** installed. Most usually, the command is called with one argument which is the name of the file to spellcheck.

```
% ispell spelling.txt
```

A full-screen spellchecking program will be brought up on your terminal.

The **ispell** screen shows the name of the file you’re spellchecking and the current mis-spelled word that it’s dealing with -- in this case, "sinle", a mis-spelling for "single". Just below that is the line which contains the mis-spelling, to give you some context to understand where the mis-spelled word appears in your text.

Below that is a list of possible alternatives for the word, numbered from zero upwards. At the bottom of the screen is a list of options:

Table 3-1. ispell keystroke commands

Command	Meaning
`number`	Any number from 0 upwards, matching the list of words on the screen. If the correct version of the word you want is on the list, just choose that number to replace it.
r	Use this option to replace the mis-spelled word with a word which isn’t on the list.
a	Use this option if the word isn’t mis-spelled at all and you want **ispell to leave it as it is.**
i	Insert the mis-spelled word into the dictionary so that it won’t show up as mis-spelled in future.
l	Lookup a word to see if it’s in the dictionary.
u	Uncapitalise the word.

Command	Meaning
q	Quit, asking you whether you want to save or not.
x	Exit without saving.
?	Help screen

When you've finished spellchecking the file, **ispell** will exit, leaving a backup coming of your file in `*filename*.bak`.

Readme: You can find out more about **ispell** by typing **man ispell**.

Exercises

1. Use **ispell** to spellcheck the file `spelling.txt`

Comparing files with diff

The **diff** command can be used to compare two files. It takes two filenames as arguments, and outputs any lines which are different. The lines from the first file are marked with less-than signs (<) and the lines from the second file are marked with greater-than signs (>). Line numbers are also given in the output.

```
% diff spelling.txt spelling.txt.bak
1,2c1,2
< 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 is not a sinle operatign system.  Rather, it is a family of
> operating systems based on a comon ancestor, the original Bell Labs Unix.
4c4
< Unix variants (or "flavours", as they are often called) are usually either
---
> Unix variants (or "flavours", as they are often called) are usally either
8c8
< There are both commercial variants of Unix (such as Sun's Solaris,
IBM's
---
> There are both comercial variants of Unix (such as Sun's Solaris,
> IBM's
%
```

This command is particularly useful when you have two files which are quite similar but you are unsure as to what differences they may contain.

If you just want to know whether two files differ or not, you can use the `-q` command line switch:

```
% diff -q spelling.txt spelling.txt.bak
Files spelling.txt and spelling.txt.bak differ
```

The `-i` argument can be used to ignore the case of the text, so that "A" will match "a" (and vice versa).

Other output formats are also available. For instance, the `-u` command line switch specifies "unified format" output, and the `-y` switch specifies "side by side" output format.

> **Readme:** The manual page for **diff** describes other options for this command. Use **man diff** to read about them.

Exercises

1. Experiment with using different formats of **diff** to examine the differences between different text files.

Sorting text files with sort

The sort command takes any number of filenames as command line arguments, sorts all the lines in them in ASCII order (which is like alphabetical order except that all the upper case letters come before the lower case letters), and prints the sorted result to STDOUT.

The following example shows a simple phone list file, unsorted, and then the results of sorting it.

```
% cat phonelist.txt
Rubble, Barney         9876 5432
Flintstone, Fred       9888 7777
Simpson, Homer         9555 1234
Jetson, George         9333 2211

% sort phonelist.txt
Flintstone, Fred       9888 7777
Jetson, George         9333 2211
Rubble, Barney         9876 5432
Simpson, Homer         9555 1234
```

Sorting in reverse

The `-r` flag sorts the file in reverse order:

```
% sort -r phonelist.txt
```

```
Simpson, Homer          9555 1234
Rubble, Barney          9876 5432
Jetson, George          9333 2211
Flintstone, Fred        9888 7777
```

Sorting numerically

The `-n` flag sorts the file numerically rather than alphabetically.

```
% cat numbers.txt
193
12
4
1

% sort numbers.txt
1
12
193
4

% sort -n numbers.txt
1
4
12
193
```

Can you see what's happening here?

Using pipes and redirection with sort

Note that you can use **sort** as a target for redirection or to pipe output through. Both the following commands are equivalent to the one given above. Can you see why?

```
% sort < phonelist.txt
% cat phonelist.txt | sort
```

Readme: More information about the **sort** command can be found by typing **man sort**.

Exercises

1. Practice using **sort** to sort text files.
2. What command line switch could be used to make **sort** case-insensitive? (It's in the man page)

3. What option could you use to only output unique items?

Finding unique lines in a text file with uniq

If you want to find the unique lines in a file you can also use the **uniq** command.

For instance, the file `email.txt` contains email addresses, and we might want to extract only the unique addresses:

```
% cat email.txt
skud@netizen.com.au
training@netizen.com.au
fred@example.com
barney@example.com
training@netizen.com.au
wilma@example.com
barney@example.com
training@netizen.com.au

% sort email.txt | uniq
barney@example.com
fred@example.com
skud@netizen.com.au
training@netizen.com.au
wilma@example.com
```

Now imagine a different example, where you've asked each of the people in your work group to send you an email detailing what sort of drinks they'd like to have kept in the tea-room. You've saved them as separate files called *`persons-name`*`.drink`

```
% ls *drink
barney.drink  fred.drink    mary.drink
betty.drink   joe.drink     wilma.drink

% cat *drink
coffee
diet coke
pepsi
coffee
coffee
coke
diet coke
tea
orange juice
coke
tea
lemonade
coffee
tea
```

You’ll want to make a shopping list with each item only appearing once, so we can use **uniq** to achieve this:

```
coffee
coke
diet coke
lemonade
orange juice
pepsi
tea
```

Note that you do have to provide already-sorted input to **uniq**. But before you decide that this is no more useful than **sort -u**, note that there are some other options which provide added functionality.

The `-c` switch prints a count of how many times each item appears in the list:

```
% sort *drink | uniq -c
      4 coffee
      2 coke
      2 diet coke
      1 lemonade
      1 orange juice
      1 pepsi
      3 tea
```

Which are the most popular drinks? We can sort the output of the previous command using the **-n** and **-r** flags to sort in reverse numeric order:

```
% sort *drink | uniq -c | sort -rn
      4 coffee
      3 tea
      2 diet coke
      2 coke
      1 pepsi
      1 orange juice
      1 lemonade
```

Readme: Other options for **uniq** can be found by typing **man uniq**.

Exercises

1. Practice using the **uniq** on some of the text files in your home directory.

Heads and tails - viewing parts of a text file

As you know, the **cat** command can be used to print out a file to STDOUT. However, sometimes you know that you only want to see the first few lines, or the last few lines. You could use **more** or **less** and just page down to the part you needed, but (as is often the case with Unix) there are commands that are designed to do exactly what you want. They are **head** and **tail**.

```
% head /usr/dict/words
Aarhus
Aaron
Ababa
aback
abaft
abandon
abandoned
abandoning
abandonment
abandons

% tail /usr/dict/words
zoologically
zoom
zooms
zoos
Zorn
Zoroaster
Zoroastrian
Zulu
Zulus
Zurich
```

You can even pipe the output of another command through **head** or **tail**.

```
% ls /usr/bin | head
822-date
GET
HEAD
Mail
MakeTeXPK
POST
PilotManager
X11
[
a2p

% ls /usr/bin | tail
zip
zipgrep
zipinfo
zipnote
zipsplit
zless
zmore
znew
zone
zsoelim
```

You can use a command line switch which is a number, to specify how many lines you want to see:

```
% ls /usr/bin | head -5
822-date
GET
HEAD
Mail
MakeTeXPK

% ls /usr/bin | tail -3
znew
zone
zsoelim
```

It’s particularly useful to be able to run **tail** on files which are constantly having text added onto their end, like logfiles. There’s even a command line switch to keep showing lines as they’re added onto the end: **tail -f *`filename`***.

Exercises

1. What are the first dozen files in the `/etc` directory?

Chapter summary

- The **fmt** can be used to format text files so they are neatly line-wrapped
- The **wc** can be used to count words, lines or characters in any kind of text file
- **ispell** is a spellchecker which can be used on any text file
- The **diff** command compares two files and prints out the differences between them
- **sort** can be used to sort the lines in a textfile into asciibetical order
- **uniq** can be used to find the unique lines in an already-sorted text file, or to count how many times each line appears
- The **head** and **tail** commands can be used to show only the top of bottom of a text file

Chapter 4. Tools for compression and archiving

In this chapter...

It’s often useful to be able to compress large files so they take up less disk space, or create archives of collections of files to make them easier to store or transfer. This chapter looks at compression and archiving tools for Unix.

Compressing (zipping) files with gzip

A single file can be compressed (made smaller) using the **gzip** command.

```
% ls -l flavour.txt
-rw-r--r--   1 skud     skud          769 Nov  4 19:21 flavour.txt
% gzip flavour.txt
% ls -l flavour.txt.gz
-rw-r--r--   1 skud     skud          472 Nov  4 19:21 flavour.txt.gz
```

The file has been compressed from 769 bytes to 472 bytes, and a file extension of `.gz` has been added.

To uncompress a file again, use **gunzip**:

```
% gunzip flavour.txt.gz
% ls flav*
-rw-r--r--   1 skud     skud          769 Nov  4 19:21 flavour.txt
```

Advanced: Some Unix systems may not have **gzip** installed. The older **compress** and **uncompress** achieve a similar result, but use a `.Z` extension instead of `.gz`. The compression used by **compress** is not as efficient as that used by **gzip**; that is, the resulting compressed file will be larger than the one produced by **gzip**.

A new compression program which is gaining popularity is **bzip2**. This compression scheme is slightly more efficient and leaves a `.bz2` extension on the compressed file. The matching decompression tool is called **bunzip2**.

Exercises

1. Practice compressing and uncompressing files using **gzip**.

2. Compare the sizes of compressed files produced with **gzip**, **compress**, and **bzip2**, if these are available on the system you are using.

Viewing compressed files with zcat

You can view the contents of a file compressed with **gzip** using the **zcat** command.

```
% gzip email.txt
% zcat email.txt.gz
skud@netizen.com.au
training@netizen.com.au
fred@example.com
barney@example.com
training@netizen.com.au
wilma@example.com
barney@example.com
training@netizen.com.au
```

There's also a **zless**, which is the equivalent of **zcat** ***`filename`*****.gz | less**.

Exercises

1. Practice viewing the contents of compressed files using **zcat**.

Archiving files with tar

What if you want to bundle up a set of files to transfer or store as a single entity? For instance, you might want to make a backup of your entire home directory, or create an archive of your writing to send to a friend by email.

The command used to do this is **tar**, which stands for "tape archive" (because it was originally used to create archives for tape backup).

There are three main things you can do with a tape archive file: create a new one, extract files from one that's already created, or just look at the contents of it.

Table 4-1. tar actions

Action	Command line switch
Create	`-c`
Extract	`-x`

Action	**Command line switch**
View/list contents	`-t`

The **tar** also requires a `-f` switch to tell it what `.tar` file to create or to extract files from. The first example below creates a `.tar` file called `myfiles.tar` containing all the text files in the current directory. The second example extracts all the files in `myfiles.tar` into the current directory.

```
% tar -cf myfiles.tar *.txt
% tar -xf myfiles.tar
```

The `-v` switch makes **tar** more verbose, printing out the names of files as it works with them:

```
% tar -cvf myfiles.tar *.txt
email.txt
flavour.txt
phonelist.txt
spelling.txt
```

Readme: Other options for **tar** are numerous. Read **man tar** to find out what they all are.

Archives created with **tar** are often compressed using **gzip** or some other compression program to make them smaller. The resulting file usually has an extension of `.tar.gz` (or sometimes `.tgz`) and is commonly referred to as a "tarball". Tarballs are a common format for Unix software which is made available for download via the WWW or by FTP.

Advanced: If you download tarballs and wish to install them, the usual method is to use the following commands:

```
# un-tar the file
% tar -xzvf tarball.tar.gz

# change to the directory where the extracted stuff ended up
% cd tarball

# configure the software prior to installation
% ./configure

# compile the software
% make

# change to the superuser account
% su -

# install the software
% make install
```

Exercises

1. Read the **tar** manual page. What does the `-k` flag do?
2. Practice creating and extracting archives using **tar** by creating an archive of your home directory

Chapter summary

1. The **gzip** can be used to compress (zip) files
2. Other compression utilities include **compress** and **bzip2**
3. **tar** can be used to create, list, or extract an archive of many files
4. Archives which are created with **tar** and then compressed with **gzip** are often referred to as "tarballs"
5. Unix software is often distributed as tarballs

Chapter 5. Tools for finding things

In this chapter...

Using whereis and which to find programs

The **whereis** command tells you where a program is on the system:

```
% whereis gzip
gzip: /bin/gzip /usr/man/man1/gzip.1.gz
```

As you can see, the **gzip** program itself is in `/bin/gzip` and its manual page, in `/usr/man/man1/gzip.1.gz` is also shown.

To find out what would actually be run when you type a command, you can use the **which** command:

```
% which grep
/bin/grep
% which rm
rm:      aliased to rm -i
```

Finding files with locate and find

locate

The **locate** command can *only* be used to find files by their names. It takes an argument which is either a plain text string (in which case it finds all files whose names contain that string) or a glob pattern (in which case it finds all files whose names match the pattern).

```
% locate locate
/usr/bin/locate
/usr/lib/locate
/usr/lib/locate/bigram
/usr/lib/locate/code
/usr/lib/locate/frcode
/usr/man/man1/locate.1.gz
/usr/man/man5/locatedb.5.gz
/usr/X11R6/man/man3/XdbeAllocateBackBufferName.3x.gz
/usr/X11R6/man/man3/XdbeDeallocateBackBufferName.3x.gz
/usr/X11R6/man/man3/XtAllocateGC.3x.gz
/var/lib/locate
/var/lib/locate/locatedb
```

```
/var/lib/locate/locatedb.n
```

locate uses a database of files which is automatically updated on a regular basis, usually overnight.

Exercise

1. Try using the **locate** to find files which match a fixed string or a glob pattern, for instance all files matching `*train*`

find

The **find** can be used to find files matching a given pattern under a directory. The arguments it takes are: where to search (the directory to start at), what to match on (any number of things!), and what to do with the output.

The following example searches the directory `/etc` for any files matching the glob pattern `*.conf` and prints the output:

```
% find /etc -name "*.conf" -print
/etc/pcmcia.conf
/etc/resolv.conf
/etc/host.conf
/etc/nsswitch.conf
/etc/adduser.conf
/etc/updatedb.conf
/etc/syslog.conf
/etc/isapnp.conf
/etc/ld.so.conf
/etc/inetd.conf
...
```

The output of **find** is often lengthy, so you may wish to pipe it through **less** so that it doesn't fly past too fast to read.

Readme: find has a plethora of command line options. Read all about them by typing **man find**.

Exercise

1. Practice using **find** to find files whose names match a certain pattern
2. What happens if you don't put any match conditions on the **find** command?
3. Read **man find** and experiment with some of the other matching options

Using xargs to act on found files

Sometimes you want to perform an action on each of the files found by **find** or some other command. One way of doing this is to pipe the output of **find**, which is simply a list of files, to the **xargs** command. Simply put the command you wish to run on each file after **xargs** on the command line:

```
% find /etc -name "*.conf" -print | xargs wc
     12      41     269 /etc/pcmcia.conf
     11      27     231 /etc/resolv.conf
      2       4      26 /etc/host.conf
     18      49     406 /etc/nsswitch.conf
     48     258    1646 /etc/adduser.conf
     11      53     362 /etc/updatedb.conf
     72     200    1691 /etc/syslog.conf
     24     141     776 /etc/isapnp.conf
      3       3      52 /etc/ld.so.conf
     61     324    2403 /etc/inetd.conf
```

Exercise

1. Practice using **xargs** with **find** on files in your home directory.

Matching text in files with grep

The **grep** command can be used to find text within text files. Imagine, for example, that you wanted to find any text files containing the string "Fred" in them:

```
% grep Fred *.txt
phonelist.txt:Flintstone, Fred        9888 7777
```

As you can see, Fred appears in our phonelist file.

grep is case sensitive. Use the `-i` switch to make it case insensitive:

```
% grep -i fred *.txt
email.txt:fred@example.com
phonelist.txt:Flintstone, Fred        9888 7777
```

You can use **grep** as a target for redirection or piping, too:

```
% ps auxw | grep net
root       154  0.0  0.1  1304    72  ?  S   Oct 26   0:00 /usr/sbin/inetd
root     20403  0.0  0.4  2060   160  p3 S    23:24   0:04 ssh skud hiro.netizen.com.au
root     20649  0.0  0.6  1820   248  S2 S    23:57   0:00 /usr/sbin/pppd call netizen
skud     16642  0.0  0.0  1848     0  ?  SW  Oct 28   0:00 (axnet)
skud     21702  0.0  0.9  1104   388  p6 S    08:01   0:00 grep net
skud     25455  0.4  0.1 30524    76  ?  S   Oct 31  35:31 (netscape)
skud     25456  0.0  0.0 16672     0  ?  SW  Oct 31   0:00 (netscape)
```

> **Advanced:** There are two other versions of **grep** available on most systems. An extended version called **egrep** allows you to use "regular expressions" (which are like wildcards or globs, only moreso) when matching text.
>
> ```
> % egrep '(fr[aeiou]d|barney)@example\.c..' *.txt
> email.txt:fred@example.com
> email.txt:barney@example.com
> email.txt:barney@example.com
> ```
>
> (the syntax of the regular expressions is explained in the manual page for **egrep**)
>
> The **fgrep** command takes an argument `-f filename` and the contents of that file are treated as a list of fixed strings, separated by newlines, to be matched on.
>
> ```
> % cat fgreplist
> fred
> barney
> wilma
> betty
>
> % fgrep -i -f fgreplist *.txt
> email.txt:fred@example.com
> email.txt:barney@example.com
> email.txt:wilma@example.com
> email.txt:barney@example.com
> phonelist.txt:Rubble, Barney          9876 5432
> phonelist.txt:Flintstone, Fred        9888 7777
> ```

Where the **grep** family of commands can be really useful is when combined with **find** and **xargs**:

```
find . -name "*.txt" -print | xargs grep -i Fred
./exercises/email.txt:fred@example.com
./exercises/phonelist.txt:Flintstone, Fred        9888 7777
```

> **Readme:** The **grep** family of commands are documented in the manual pages (type **man grep** to read them) or on page 89 of the textbook.

Exercise

1. Read the **grep** manual page. What does the `-c` flag do? What option can be used to print any lines which *don't* match the pattern?
2. Practice using **grep** on files in your home directory

Chapter summary

1. The **whereis**and **which** commands can be used to find programs and associated files on the system.
2. **find** can be used to find any file anywhere on the system, by a range of different criteria
3. **locate** can be used to find all files whose filenames match a certain pattern
4. **xargs** takes a list of filenames (such as the output of **find**) and uses each one as an argument to another command such as **grep**
5. **grep** (and **egrep** and **fgrep**) can be used to find text within a file

Chapter 6. System and network information tools

In this chapter...

There are many tools which can be used on Unix to find out information about the system, including finding out the operating system version, disk space usage, system performance, etc. Likewise, a whole family of tools exist to help with network diagnostics and other network-related information.

Finding the operating system with uname

You can find out what operating system is running with **uname**.

```
% uname
Linux
```

The `-a` switch shows more information:

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

Exercises

1. Try using the **uname** to find out what operating system the training server is running

Viewing performance statistics with uptime

You can find out how long the system's been up, and what it's performance is, with **uptime**:

```
% uptime
  9:00am  up 10 days,  2:53,  8 users,  load average: 1.37, 1.11, 1.12
```

The output of the **uptime** command shows the current system time, how long the system has been up, how many users are currently logged in, and the "load average".

Unix systems are designed to be stable, and do not usually require rebooting even when installing or upgrading major software. This means that uptimes in months or years are not uncommon. Most

Unix systems are only rebooted for hardware upgrades or during power outages if they are not on a UPS (uninterruptable power supply).

Load average is an indication of how overloaded the systems resources are. Because a Unix system can handle many processes at once, sometimes a queue forms when too many processes are trying to be run. The load average is an indication of how long that queue is, averaged over three different time periods (the first one is the shortest time period), which is why the three values differ.

A load average of less than 1 means that all processes are being dealt with immediately. A load average of 1 means the system is busy all the time. Anything between 1 and 3 means that there is a queue, but that the system is probably still usable. Anything greater than 3 will start to noticeably impact performance and users will be able to tell that the system is slow.

Exercises

1. Try using the **uptime** command to see how long the system has been up and how heavily loaded it is.

Viewing processes and resource usage with top

The **top** command shows which processes are using the most system resources. Its output looks like this:

The screen is updated every few seconds to show the latest information.

Exercises

1. Try using the **top** command. What processes are using the most resources on this system?

Finding the name of the system with hostname

The **hostname** command shows you the name of the system you are using.

The `-d` switch shows the domain name of the system. The `-f` switch shows the fully qualified domain name, which is the hostname and the domain name joined together.

```
% hostname
training
```

Under Linux, the `-d` switch shows the domain name of the system. The `-f` switch shows the fully qualified domain name, which is the hostname and the domain name joined together. This is not consistent with other flavours of Unix, however.

```
% hostname -d
netizen.com.au
% hostname -f
training.netizen.com.au
```

Exercises

1. Try using the **hostname** command to find the name of the system you are currently using.

Looking up a system's IP number or name using nslookup

Internet addresses exist in both numeric form and a more memorable format based on words. For instance, the system called `training.netizen.com.au` is also known by its numeric address, `203.30.75.3`

Sometimes you will want to translate from one format to another. The command to do this is nslookup:

```
% nslookup training.netizen.com.au
Server:  ns.netizen.com.au
Address:  203.30.75.2

Name:    vitaly.netizen.com.au
Address:  203.30.75.3
Aliases:  training.netizen.com.au

% nslookup 203.30.75.3
Server:  ns.netizen.com.au
Address:  203.30.75.2

Name:    vitaly.netizen.com.au
```

```
Address:  203.30.75.3
```

The "Server" is the domain name server which is answering your queries.

You can use **nslookup** in "interactive" mode by not giving it a hostname or IP address to look up:

```
% nslookup
Default Server:  localhost
Address:  127.0.0.1

>
```

In interactive mode, you can choose what server to use for your queries and perform all kinds of extra tricks.

> **Readme:** To find out more about using **nslookup** in interactive mode, type **man nslookup**.

> **Advanced:** The default DNS server(s) to use are defined in the file `/etc/resolv.conf`. You can find out more about this file by using **man 5 resolv.conf**.

Exercises

1. Try looking up the IP number for an Internet-connected system such as your company's mail or web server or a popular web site.

Seeing if a network-connected system is reachable using ping

You can use the **ping** to see whether a system connected to the network is reachable:

```
% ping training.netizen.com.au
PING training.netizen.com.au (203.30.75.2): 56 data bytes
64 bytes from 203.30.75.3: icmp_seq=0 ttl=249 time=31.2 ms
64 bytes from 203.30.75.3: icmp_seq=1 ttl=249 time=17.1 ms
64 bytes from 203.30.75.3: icmp_seq=2 ttl=249 time=12.7 ms
64 bytes from 203.30.75.3: icmp_seq=3 ttl=249 time=12.5 ms
```

Press **CTRL-C** to make it stop. If you only want it to do a certain number of pings, use the `-c` switch.

Exercises

1. Try pinging different Internet-connected systems, some nearby on the network and some distant. Your trainer will be able to suggest suitable systems to ping.

Checking the route to a networked system using traceroute

Sometimes when you are trying to connect to a remote system, there will be a network problem. When reporting such problems, it is often useful to provide the output of the **traceroute** command for diagnostic purposes. This command lists all the "hops" between your system and the remote system, following the same path as your data would normally follow:

```
traceroute to news.com.au (165.69.1.212), 30 hops max, 40 byte packets
 1  reason.netizen.com.au (203.30.75.126)  0.835 ms  0.804 ms  0.754 ms
 2  minos.sprint.com.au (203.20.104.7)  4.518 ms  2.497 ms  2.875 ms
 3  medea.labyrinth.net.au (203.30.143.10)  10.173 ms  7.519 ms  17.186 ms
 4  202.139.11.133 (202.139.11.133)  9.148 ms  11.659 ms  13.789 ms
 5  s10-6.sb1.optus.net.au (202.139.0.21)  67.417 ms  34.336 ms  37.652 ms
 6  atmsr-1-2.si1.optus.net.au (192.65.89.130)  31.507 ms  25.161 ms  34.931 ms
 7  NewsLTD.si1.optus.net.au (202.139.18.2)  55.119 ms  34.264 ms  33.873 ms
 8  NewsLTD.si1.optus.net.au (202.139.18.2)  42.145 ms  46.665 ms  40.199 ms
 9  165.69.1.212 (165.69.1.212)  44.015 ms  45.648 ms  27.394 ms
```

The **traceroute** program tries three times to get to each system along the way, and tells you how many milliseconds it took in each case. If there is a problem between your site and the next site, such as a system which has crashed or a cable which has broken, you'll see a line which looks like this:

```
 5  * * *
```

Readme: For more information about **traceroute** options and output formats, type **man traceroute**.

Exercises

1. Try tracing the route to an Internet-connected server such as one of the ones you pinged in an earlier exercise.

Chapter summary

1. The **uname** can be used to find out what version of the Unix operating system is running on a specific server
2. **uptime** can be used to see how long since a system was last rebooted and how heavily loaded it is
3. **top** shows a constantly-refreshing view of the most resource-hungry processes running on a Unix system
4. **hostname** can be used to find the name of the server you are using
5. **ping** can be used to find out if a server is reachable on the network
6. **traceroute** traces the network route between your system and a remote system
7. **nslookup** can be used to convert hostnames to IP numbers and vice versa

Chapter 7. Date and time tools

In this chapter...

There are a number of tools under Unix related to telling the time or date. In particular, tools which can run commands at a certain date or time make automating tasks under Unix very simple indeed.

Printing the date and time with date

The **date** command prints out the current date and time in human-readable format:

```
% date
Fri Nov  5 10:43:26 EST 1999
```

You can change the output format using special sequences of characters which begin with percent (`%`) signs:

```
% date +'%Y-%m-%d'
1999-11-05
```

You can also ask **date** to give you output for times specified in just about any format, by using the long switch `--date`:

```
% date --date '3 days ago'
Tue Nov  2 10:49:43 EST 1999
% date --date '6 months'
Fri May  5 10:49:58 EST 2000
```

You can get output in UTC format (which is pretty much similar to GMT, Greenwich Mean Time), by using the `-u` switch.

```
Thu Nov  4 23:52:02 UTC 1999
```

Exercise

1. Practice using the **date** command to print out various dates in different formats

Viewing a calendar with cal

The **cal** prints out a calendar for the current month:

```
% cal
   November 1999
 S  M Tu  W Th  F  S
    1  2  3  4  5  6
 7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30
```

You can get a calendar for a full year by using the year as a command line argument:

```
% cal 2003
                              2003

      January               February               March
 S  M Tu  W Th  F  S   S  M Tu  W Th  F  S   S  M Tu  W Th  F  S
          1  2  3  4                     1                     1
 5  6  7  8  9 10 11   2  3  4  5  6  7  8   2  3  4  5  6  7  8
12 13 14 15 16 17 18   9 10 11 12 13 14 15   9 10 11 12 13 14 15
19 20 21 22 23 24 25  16 17 18 19 20 21 22  16 17 18 19 20 21 22
26 27 28 29 30 31     23 24 25 26 27 28     23 24 25 26 27 28 29
                                            30 31
       April                  May                   June
 S  M Tu  W Th  F  S   S  M Tu  W Th  F  S   S  M Tu  W Th  F  S
       1  2  3  4  5               1  2  3   1  2  3  4  5  6  7
 6  7  8  9 10 11 12   4  5  6  7  8  9 10   8  9 10 11 12 13 14
13 14 15 16 17 18 19  11 12 13 14 15 16 17  15 16 17 18 19 20 21
20 21 22 23 24 25 26  18 19 20 21 22 23 24  22 23 24 25 26 27 28
27 28 29 30           25 26 27 28 29 30 31  29 30

        July                 August              September
 S  M Tu  W Th  F  S   S  M Tu  W Th  F  S   S  M Tu  W Th  F  S
       1  2  3  4  5                  1  2      1  2  3  4  5  6
 6  7  8  9 10 11 12   3  4  5  6  7  8  9   7  8  9 10 11 12 13
13 14 15 16 17 18 19  10 11 12 13 14 15 16  14 15 16 17 18 19 20
20 21 22 23 24 25 26  17 18 19 20 21 22 23  21 22 23 24 25 26 27
27 28 29 30 31        24 25 26 27 28 29 30  28 29 30
                      31
      October               November              December
 S  M Tu  W Th  F  S   S  M Tu  W Th  F  S   S  M Tu  W Th  F  S
          1  2  3  4                     1      1  2  3  4  5  6
 5  6  7  8  9 10 11   2  3  4  5  6  7  8   7  8  9 10 11 12 13
12 13 14 15 16 17 18   9 10 11 12 13 14 15  14 15 16 17 18 19 20
19 20 21 22 23 24 25  16 17 18 19 20 21 22  21 22 23 24 25 26 27
26 27 28 29 30 31     23 24 25 26 27 28 29  28 29 30 31
                      30
```

To get a specific month for a specific year, use **cal** ***`month year`***:

```
% cal 11 2000
   November 2000
 S  M Tu  W Th  F  S
          1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
```

```
19 20 21 22 23 24 25
26 27 28 29 30
```

Readme: The manual page for **cal** contains some interesting historical information about the program's handling of old Gregorian calendar dates.

Exercise

1. Practice using the **cal** command
2. Take a look at the calendar for the year 1752. What's going on in September? The manual page will explain it in more detail.

Running a job at a given time with at

The **at** command can be used to set a task to run at a particular time. It takes an argument which is a the time when you want the job to run, in just about any format.

Readme: The specific time formats accepted by **at** can be found by typing **man at**.

After typing **at** ***`time`***, you can type in commands to run, one on each line. Press **CTRL**-**D** to end your commands.

```
% at 17:00
at> echo "Time to go home!"
at>
```

To see what jobs you have queued, type **atq**:

```
% atq
1       1999-11-05 17:00 a skud
```

To remove a job, use the **atrm** ***`job_number`*** command:

```
% atq
1       1999-11-05 17:00 a skud
% atrm 1
% atq
%
```

Exercise

1. Set an **at** job to run in about five minutes time, perhaps copying some files for you (as if it were a backup job)
2. Practice using **atq** and **atrm**

Running a job at a regular intervals with crontab

You can also schedule jobs to run at a regular interval using **crontab**. The name stands for "chronological table" and in fact the file that stores the information about jobs to run is just a text file (usually stored in `/var/spool/cron/crontabs/`*`username`* containing fields for:

- minute
- hour
- day of month
- month
- day of week
- command to run

> **Readme:** For more information about the format of the `crontab` file, see **man 5 crontab**.

The `crontab` file is not edited directly. Instead, we use the **crontab** to edit or view it. The `-e` switch invokes our favourite editor (as set in the `EDITOR` environment variable) to edit the file. The `-l` switch simply lists entries in our crontab.

```
% crontab -l
# DO NOT EDIT THIS FILE - edit the master and reinstall.
# (/tmp/crontab.XXXXa02218 installed on Mon Aug 16 12:09:06 1999)
# (Cron version -- $Id: datetime.sgml,v 1.2 2000/02/02 06:32:42 skud Exp $)

# run a backup every night at midnight
0 0 * * * /home/skud/bin/backup.sh

# index the website once a week at 2am on Sunday
0 2 * * 0 /www/htdig/bin/rundig
```

Some notes about Year 2000 compliance

Unix stores its dates internally as the number of seconds since 00:00, January 1st, 1970. Dates are represented as a 32-bit number, and aren't expected to run out until 2038. By this stage, it is hoped that all Unix systems will be using 64-bit dates.

However, this doesn't mean that data stored elsewhere on Unix systems (for instance in text files or databases) follows the same date convention. Some Y2K problems may therefore be experienced even on Unix systems.

Chapter summary

1. The **date** can be used to find the current date and time in a variety of formats, or to output any date you want in any format
2. The **cal** is used to print out a calendar for any month and/or year
3. A job can be run at a given time in the future with **at**
4. Jobs can be scheduled to run at regular intervals with **crontab**

Chapter 8. Tools for communication

In this chapter...

This chapter introduces some commands for communicating with other users on the system, including real-time chat and email.

Sending messages to other users with write

You can use the **write** command to send messages to other users who are logged in to the same system. Simply type **write** ***`username`*** then enter a line or two of text to be sent to that user.

```
% write skud
Hi Skud, what’s new?
```

And end-of-file character must be sent to end the message. This is usually **CTRL-D**

Exercise

1. Practice by sending messages to other trainees. You can see who’s logged in by using the **who** or **w** commands

Real-time chat with talk

If you want to chat to another user in real-time, you can use the **talk** command. This sends a message to their terminal letting them know that you want to talk to them, and asking them to respond with a similar command:

```
% talk fred
```

You can also send talk requests to people on other systems:

```
% talk mary@hostname.example.com
```

The other person’s system will show them this message:

```
Message from TalkDaemon@his_machine...
talk: connection requested by your_name@your_machine.
talk: respond with: talk your_name@your_machine
```

Once they respond, you will see a blank screen and can start typing to each other. Press **CTRL-C** to exit the talk program.

> **Note:** There are other variants on **talk** such as **ntalk** and **ytalk**. These have some additional features, for instance **ytalk** allows more than two people to talk at once, and splits the screen into multiple windows to keep different people's words from getting mixed up.

talk etiquette

It's considered bad manners to send talk requests to people you don't know, or to people who have expressed a wish not to receive talk requests.

When using **talk**, it is customary to end your own sentences with a blank line by hitting **ENTER** twice. This lets the other person know that you're ready for them to speak, and stops the messages from getting tangled up.

Toggling messages on and off with mesg

If you don't wish to receive talk requests, you can use the command **mesg -n**. People often put this in their shell initialisation file (eg `.tcshrc` or `.bashrc`) so that messages are turned off by default. To turn messages on again, use **mesg -y**.

Sending and receiving electronic mail

Using the mail command

All Unix systems have a command called **mail** which can be used to send and read mail from the command line.

Sending email

The **mail** can be used to send email to anyone on the local system or any Internet connected system

It will ask you for the subject line for your message, then allow you to type in a message on STDIN (standard input), using the end-of-file character (**CTRL-D**) to end your input. It may also ask you to fill in other mail headers (such as the Cc: header) when you have finished.

```
% mail training@netizen.com.au
Subject: A quick question
Just wondering if you could tell me some more about the following
courses:
```

```
Unix Basics
Unix Tools
Introduction to Perl
Internet Fundamentals

Are they going to run in Melbourne in the near future?

Thanks in advance,

Fred
^D
Cc: info@netizen.com.au
%
```

The **mail** command takes an optional argument, `-s`, which is the subject line for the email. Note that if the subject contains more than one word, you'll need to put quotes around it.

The **mail** program can also act as a pipe or as a target for shell redirection. The first two of the following examples have exactly the same effect.

```
% mail training@netizen.com.au < myfile.txt
% cat myfile.txt | mail training@netizen.com.au
% mail -s hello training@netizen.com.au < myfile.txt
% mail -s "The file you requested" training@netizen.com.au < myfile.txt
```

Reading email

The **mail** program can also be used for reading mail.

```
% mail
  1  friend@example.com    Tue Nov  2 16:12  22/556   "Hi!"
>N2  skud@netizen.com.au   Tue Nov  2 23:14  15/389   "this is a test."
&
```

Simply typing **mail** will give you a list of recent email messages. The "current" message will have a greater-than sign next to it. The **mail** program will them show you a prompt which is an ampersand (`&`) and wait for you to enter a command.

Table 8-1. mail commands

Long form of command	Abbreviated form	Meaning
`print`	`p`	Print current message
n/a	`+`	Print next message
n/a	`-`	Print previous message
`delete`	`d`	Delete current message
`mail`	`m`	Send an email message

Long form of command	Abbreviated form	Meaning
`reply`	`r`	Reply to current message
`quit`	`q`	Quit the mail client

Exercise

1. Practice sending email to other trainees
2. Try sending the output of the earlier "drinks" exercise to yourself, using pipelines or redirection.

Other email clients

There are numerous text-mode email clients available for Unix. This section briefly introduces a few of the more popular ones.

mutt

This email client has an interface largely based on an earlier client, **elm**. It supports attached files, address books (aliases), sorting email by threads, and much more.

The inbox screen in **mutt**:

This screen is what you see just before sending a message; you can edit any headers, add and delete attachments, etc.

pine

The **pine** email client is considered by many to be the most user-friendly text-mode email client for Unix.

mh

The **mh** mail handler is not so much a mail client as a suite of small, interconnected tools for dealing with email. Several clients have been written which use **mh** as their underlying tools, including **xmh** and **exmh** (which both run under the X windowing system). Some other mail clients such as **mutt** can also handle **mh** format email.

Chapter summary

1. The **write** command can be used to send short messages to other users on the system
2. **talk** can be used to perform real-time chat with other users on the system, or with users on remote systems
3. You must have messages switched on with **mesg y** to receive messages from other users; use **mesg n** to prevent people from sending you messages
4. The **mail** can be used interactively or as part of a command line to send email to other users on your system or elsewhere on the network
5. Other mail clients such as **mutt** and **pine** are often available on Unix systems, and are usually more user friendly than the standard **mail** command.

Chapter 9. Internet tools

In this chapter...

In this chapter we look at some other tools for accessing Internet services from the Unix command line, including FTP and WWW clients.

Transferring files with ftp

FTP stands for File Transfer Protocol, and is used to transfer files from one system to another. An FTP client program connects to an FTP server and can upload or download files.

There are many FTP servers on the Internet which store archives of free software or shareware for various platforms. One of the most popular FTP servers in Australia is at `mirror.aarnet.edu.au`.

The **ftp** command invokes an FTP client program. It takes an argument which is the hostname of the FTP server to connect to.

```
% ftp mirror.aarnet.edu.au
```

The client will connect to the FTP server, which will usually print a welcome message then ask you for a password:

```
% ftp mirror.aarnet.edu.au
Connected to mirror.aarnet.edu.au.
220-
220- Welcome to Mirror.AARNet.EDU.AU in Australia
220-
220 Mirror.AARNet FTP server ready.
Name (mirror.aarnet.edu.au:skud):
```

You will have to enter a username and password for the FTP server. In the case of a public FTP archive such as `mirror.aarnet.edu.au`, you can log in as `anonymous` and use your email address as the password. If you are connecting to an FTP server where you have your own account, you can login with your own username and password.

```
ftp mirror.aarnet.edu.au
Connected to mirror.aarnet.edu.au.
220-
220- Welcome to Mirror.AARNet.EDU.AU in Australia
220-
220 Mirror.AARNet FTP server ready.
Name (mirror.aarnet.edu.au:skud): anonymous
331 Guest login ok, send your complete e-mail address as password.
Password:
230-
230- Welcome to Mirror.AARNet.EDU.AU in sunny Brisbane, located at ITS,
230- University of Queensland, Australia.
230-
...
ftp>
```

Once you are connected to the FTP server, you will see a prompt that looks like `ftp>`. It is now waiting for you to type in commands. To see which commands are available, type `help`.

Table 9-1. Some useful commands in the FTP client

Command	Meaning
ls	List files
cd *`directory_name`*	Change directories
cd *`directory_name`*	Change directories on the local system (ie not on the FTP server)
get *`filename`*	Download a file from the FTP server
put *`filename`*	Upload a file to the FTP server
bin	Change to binary mode for uploading/downloading binary files
ascii	Change to ascii mode for uploading/downloading text files
hash on	Print out hash marks (`#) to show downloads in progress`
mget *`filenames`*	Get (download) multiple files (performs glob expansion)
mput *`filenames`*	Put (upload) multiple files (performs glob expansion)
prompt n	Don't ask yes/no questions for each file when using `mget or mput`

Exercise

1. Connect to `mirror.aarnet.edu.au`
2. Login as anonymous and use your email address as the password
3. Perform a directory listing to see what files and directories are available
4. Practice changing directories
5. Change to binary mode
6. Download a file

7. Turn **hash on** and download another file, and note the difference

World Wide Web tools

The lynx browser

A commonly available text-mode web browser is **lynx**. To view a website using **lynx**:

```
% lynx http://netizen.com.au/
```

This screenshot shows **lynx** viewing Netizen's training website

You can get **lynx** to output a single web page to STDOUT (standard output) by using the `-dump` switch:

```
% lynx -dump http://netizen.com.au/services/training/ > training.txt
```

The `-source` switch will dump out the HTML source instead of the formatted text of the page:

```
% lynx -source http://netizen.com.au/services/training/ > training.html
```

Unfortunately, many websites are not designed to be readable with a text mode browser. This is what the Channel 9 television station's website at http://www.ninemsn.com/ looks like:

This is the result of poorly written websites, and not the fault of the browser; websites which are written correctly should degrade gracefully to be perfectly workable in a text only medium.

Exercise

1. Use **lynx** to view some different websites

The w3m browser

Another text-mode browser which is gaining popularity is **w3m**. This browser has some support for tables, frames, stylesheets, and other layout and stylistic markup.

Exercise

1. Use **w3m** to view some different websites

wget

The **wget** command can be used to download web pages without viewing them.

```
% wget http://netizen.com.au/services/training/
--00:37:22--  http://netizen.com.au:80/services/training/
           => ‘index.html’
Connecting to proxy.labyrinth.net.au:8080... connected!
Proxy request sent, awaiting response... 200 OK
Length: unspecified [text/html]

    0K -> ....

00:37:24 (5.79 KB/s) - ‘index.html’ saved [4250]
```

This can be done recursively, so that a whole website can be sucked down with one command. The program reads each web page in turn and follows any links on that page.

```
% wget -r http://netizen.com.au/services/training/
--00:37:22--  http://netizen.com.au:80/services/training/
           => ‘index.html’
Connecting to proxy.labyrinth.net.au:8080... connected!
Proxy request sent, awaiting response... 200 OK
Length: unspecified [text/html]

    0K -> ....

00:37:24 (5.79 KB/s) - ‘index.html’ saved [4250]

--00:38:36--  http://netizen.com.au:80/images/biglogo.gif
           => ‘netizen.com.au/images/biglogo.gif’
Connecting to proxy.labyrinth.net.au:8080... connected!
Proxy request sent, awaiting response... 200 OK
Length: 5,040 [image/gif]

    0K -> ....                                                   [100%]

00:38:39 (2.90 KB/s) - ‘netizen.com.au/images/biglogo.gif’ saved [5040/5040]

--00:38:39--  http://netizen.com.au:80/
           => ‘netizen.com.au/index.html’
Connecting to proxy.labyrinth.net.au:8080... connected!
Proxy request sent, awaiting response... 200 OK
Length: unspecified [text/html]

    0K -> ...

00:38:40 (5.85 KB/s) - ‘netizen.com.au/index.html’ saved [3765]
```

Chapter summary

1. The **ftp** command can be used to transfer files to and from FTP sites
2. The **lynx** and **w3m** text mode browsers can be used to browse the World Wide Web from the Unix command line
3. **wget** can be used to download web pages in batch mode

Chapter 10. Using shell scripts to write your own tools

In this chapter...

In this script you will learn how to combine Unix commands together in a file which can be run as a program. This is known as shell scripting, and is an easy way to create custom applications under Unix.

What is a shell script?

A shell script is simply a file which contains shell commands. You can edit a shell script using any text editor such as **vi**. Shell scripts are very similar to MS-DOS batch files.

Simple shell scripting commands

Running programs from shell scripts

To run a program from a shell script, just put the command on a line by itself in the file:

```
echo "Creating backup..."
cd ~
mkdir backup
cp * backup/
tar -cvf backup.tar backup/
gzip backup.tar
echo "Finished backup..."
```

Outputting text with echo

The **echo** command is often used to output a message to STDOUT from inside a shell script. See the above example for a typical use of **echo**.

Putting comments in your shell scripts

Any line starting with a hash sign (`#`) is a comment. Comments are ignored when the script is run, and are just there for information:

```
#
# backup script
# written by Kirrily Robert, November 10th 1999
#

echo "Creating backup..."

# make backup directory under my home directory
cd ~
mkdir backup

# copy files to backup directory
cp * backup/

# make a tarball of the backup directory
tar -cvf backup.tar backup/
gzip backup.tar

echo "Finished backup..."
```

Running shell scripts

First of all, your shell script must be executable. The **chmod** command is used to achieve this:

```
% chmod +x myscript.sh
```

You will also need to make the first line in your script specify the shell you wish to use to interpret your shell script. This is often `/bin/sh`. The first line should start with the characters `#!` which are referred to as "shebang" (because "hash-bang" sounds like "shebang").

```
#!/bin/sh
```

To actually run the script, you will need to type its name on the command line. If the directory it's in is not in your path, you will need to specify what directory it's in as well. Usually this is the current directory:

```
% ./myscript.sh
```

Conditional and looping constructs

You can use condition constructs (if/then/else) and looping constructs (for instance "while" and "for") in your shell scripts. These will probably not be useful to you unless you are familiar with programming languages.

> **Readme:** The syntax for conditional and looping constructs varies depending on what shell you are using. Read the manual pages for your shell or chapters 3 and 4 of the textbook for more information.

Chapter summary

- Shell scripts are like MS-DOS batch files, and are simply a text file which lists commands to run
- Comments start with a hash sign
- Your script must be executable and start with a shebang line to run it

Chapter 11. Conclusion

What you’ve learnt

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

- Why Unix tools are different from tools on other platforms, and how the theory behind them ("do one thing and do it well") can provide greater power and flexibility
- How to use **wc**, **ispell** and **fmt** to provide word processing functionality such as formatting, word count and spell checking
- How to sort files, find differences between them, or find unique entries in files
- How to view parts of files using **head** and **tail**
- How to compress and archive files on a Unix system
- How to locate programs, files and text within a file using tools such as **whereis**, **find** and **grep**
- How to find system information such as OS name and version or resource usage
- How to find information about networked computers, including simple fault diagnosis, from the Unix command line
- How to use date and time tools to run automated jobs
- How to communicate with other users of the Unix system, both in real time chat and via electronic mail
- How to use Unix-based Internet tools such as FTP and various web tools

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:

- 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, vertical bar, or
~	tilde (pron. “til-duh”, wiggle, squiggle
`	backtick