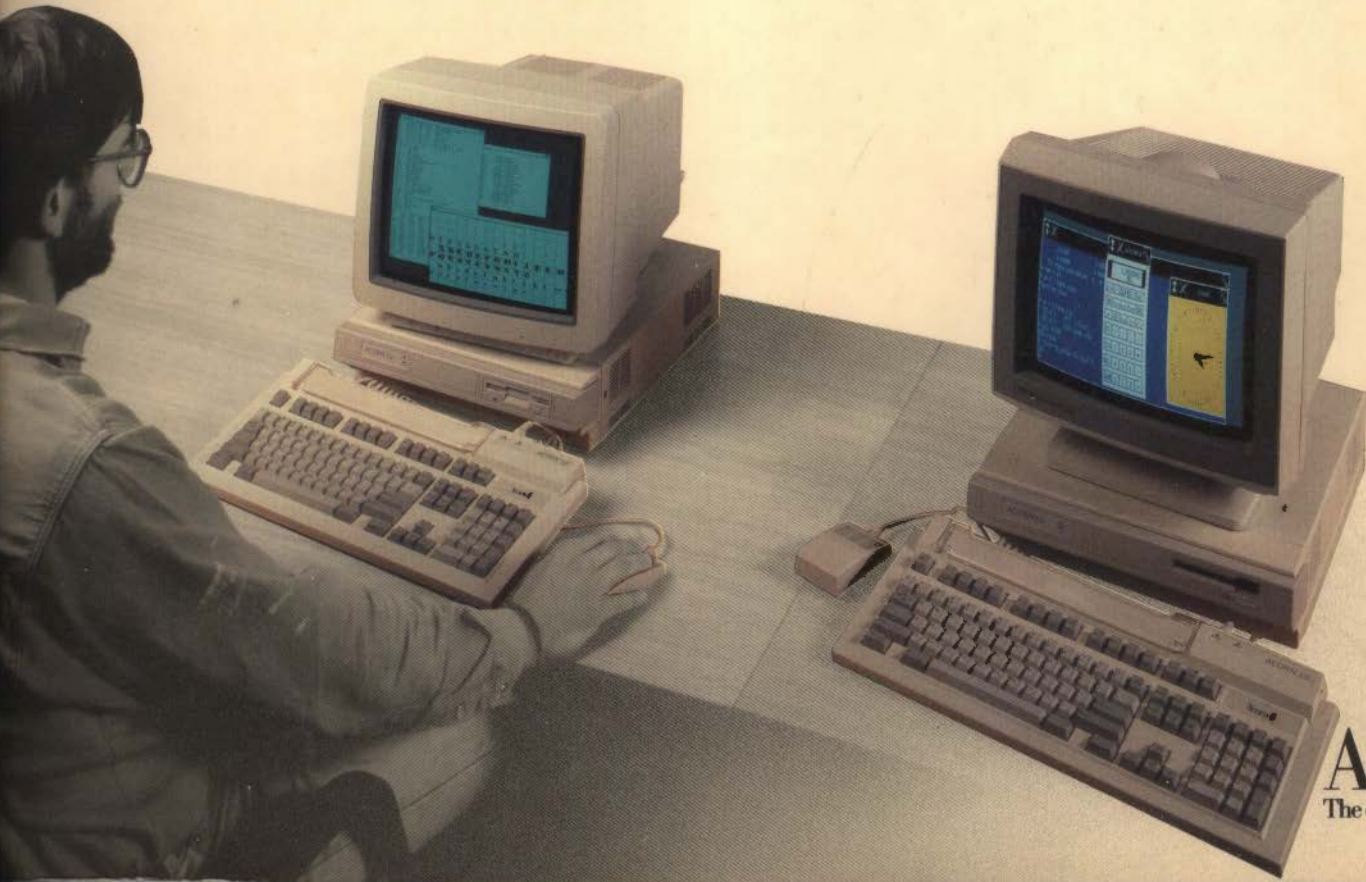


ACORN RI40

RISC iX USER GUIDE

3027/21381



Acorn
The choice of experience.

Copyright © Acorn Computers Limited 1988

Neither the whole nor any part of the information contained in, nor the product described in this manual may be adapted or reproduced in any material form except with the prior written approval of Acorn Computers Limited.

The product described in this manual and products for use with it are subject to continuous development and improvement. All information of a technical nature and particulars of the product and its use (including the information and particulars in this manual) are given by Acorn Computers Limited in good faith. However, Acorn Computers Limited cannot accept any liability for any loss or damage arising from the use of any information or particulars in this manual.

All correspondence should be addressed to:
Customer Support and Service
Acorn Computers Limited
Fulbourn Road
Cherry Hinton
Cambridge CB1 4JN

Within this publication, the term 'BBC' is used as an abbreviation for 'British Broadcasting Corporation'.

ACORN, ARCHIMEDES, ARM, and ECONET are trademarks of Acorn Computers Limited.

POSTSCRIPT is a trademark of Adobe Systems, Inc.

UNIX is a trademark of AT&T.

DEC and VAX are trademarks of Digital Equipment Corporation.

MS-DOS is a trademark of Microsoft Corporation.

ETHERNET is a trademark of Xerox Corporation.

X WINDOW SYSTEM is a trademark of the Massachusetts Institute of Technology.

Published December 1988: Issue 1
ISBN 1 85250 058 1
Published by Acorn Computers Limited
Part Number 0483,708

COMPUTING DEPT

About this guide

Readership of this guide

This guide is an introduction to the use of RISC iX – Acorn's own implementation of the UNIX operating system.

Early sections describe UNIX per se for people who have never used UNIX before, introducing the most popular commands, programs and tools of the UNIX operating system.

Later sections contain more specialist information on text editing, networking, communication, the windowing environment plus information on where to find out more about your workstation.

Overview

Here is a summary of what you will find in this guide:

Chapters

Introducing RISC iX – contains a brief description of the characteristics of RISC iX.

Overview of UNIX – describes the basic concepts of the UNIX operating system, including the structure of the filing system and how it works.

Using UNIX – introduces the basic commands of UNIX, including the shell and the commands that you can use to create and manipulate files.

Using the UNIX shell – describes how to use the shell to simplify existing commands and to create some of your own commands.

Text editing – describes the text editors available on your system with a brief description of how to use a few of them.

Networking and NFS – if your workstation is connected to a network, this chapter shows you how to access other workstations and file systems on the network.

Reference sections

Communicating with other systems and users – details the utilities available to transfer information to other users of your workstation and to users on other systems.

Using the X Window System – describes what the X Window System is and how to use some of the facilities it provides.

Further uses of RISC iX – introduces some of the other things that you can do with RISC iX that are beyond the scope of this guide, but which you may like to pursue. Likely sources of information for each of these uses are also provided.

The reference sections at the back of the guide contain supplementary information.

Trouble-shooting – helps you to locate the source of your problem should you get into difficulty using your workstation.

Command summaries – summarises all the commands available on your RISC iX workstation and their use.

RISC iX manual pages – contains a selection of reference manual pages.

An extensive bibliography is also included along with an index to help you find your way around the guide.

Conventions used in this guide

The following typographical conventions are used throughout this guide:

Convention	Meaning
<DELETE>	Press the key indicated.
<CTRL-D>	Hold down the first key and press the second.
↵	Press the RETURN key
login:	Text displayed on the screen.
cat	Text that you type in.
<i>filename</i>	A variable, where you should substitute what the word represents.

For example:

```
login: guest ↵
```

Another example:

```
$ cat filename ↵
```

where *filename* is the name of the file; for example, **readme1**:

```
$ cat readme1 ↵
```


Contents

Introducing RISC iX	1
Overview of UNIX	3
Using UNIX	19
Using the UNIX shell	41
Text editing	63
Networking and NFS	109
Communicating with other systems and users	129
Using the X Window System	161
Further uses of RISC iX	179
Bibliography	185
Reference Section A: Trouble-shooting	187
Reference Section B: Command summaries	195
Reference Section C: RISC iX manual pages	211
Index	

Introducing RISC iX

What is RISC iX?

RISC iX is a port to the ARM processor of the Berkeley 4.3 UNIX operating system (4.3BSD) with SVID extensions, Network File System (NFS) software, the X Window System and window managers.

Here is a general list of the software supplied:

- Berkeley BSD 4.3 Kernel with System V virtual memory extensions, compatible with the System V Interface Definition, SVID.
- Device drivers for many peripherals.
- Berkeley 4.3 toolkit.
- Assorted User Contributed Software (UCS).
- System Administration tools.
- C Compiler with ANSI C and pcc (Berkeley) compatibility considerations.
- ARM Assembler.
- Sun NFS Version 3.2.
- X11 Window System Release 2 with awm, twm and uwm window managers.
- disc formatters for floppy discs and hard discs.
- Data interchange tools: transfer to/from MS-DOS and ADFS floppy discs.

Additional operating systems:

- Acorn's RISC OS, with separate disc partition (standard).
- MS-DOS emulation, using RISC OS (optional).

This chapter has only given you a thumbnail sketch of RISC iX. The remaining chapters will help you to get further acquainted and to steer you through all the available software. Examples will be provided throughout the guide and pointers to other sources of information (documentation, system tutorials etc.) will also be given, to encourage you to learn more about RISC iX.

To get started, set up your system, switch on and log in as described in the *Operations Guide*. Then turn to the next chapter in this guide, *Overview of UNIX*; read through the text and try some of the examples on your workstation as you proceed.

Overview of UNIX

Introduction

If you are a newcomer to UNIX, the next three chapters of this guide are for you – they provide enough information for you to start using your system by introducing:

- the underlying concepts of the UNIX operating system (as succinctly as possible),
- the basic UNIX commands, and
- the uses of the UNIX shell.

The remaining material in this guide assumes knowledge of these three chapters, so take your time reading them and where possible, try out the examples on your system as you proceed.

The `guest` directory that you logged into in the *Operations Guide* should assist you in this respect as it contains many of the example files and directories that are used throughout this guide. So if you haven't done so already, log in to your system as `guest` and then begin reading this guide, stopping occasionally to try out the commands for yourself.

What is UNIX?

UNIX is an operating system consisting of a set of software programs that act as a link between your computer and you, the user. It controls the computer and gives you an efficient and flexible computing environment. In addition, UNIX also provides a whole host of very powerful commands that can help you in your work.

UNIX is a **multi-user** operating system. This means that it can support more than one user (multiple users) at any one time. For example, if your workstation is on a network, you can be busy typing commands on your workstation and meanwhile another user from another workstation on the network can also log in to and use your workstation. The good thing is that you are unaware of this additional user, because UNIX looks after it all.

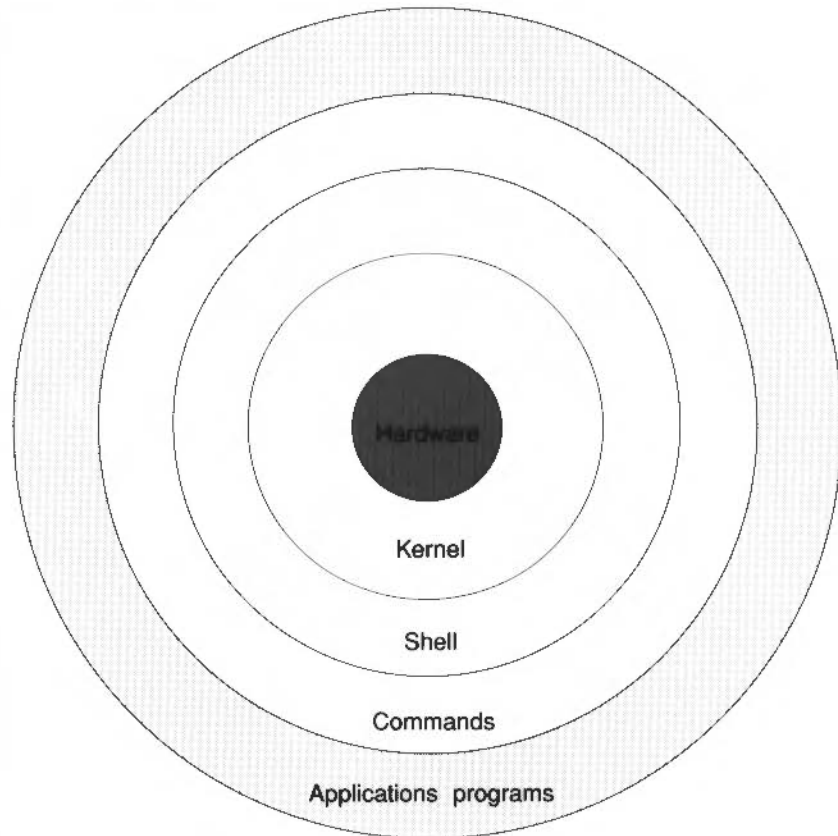
Even if your workstation is not connected to a network you can still take advantage of the multi-user concept. Just as UNIX can cope with more than one user, it can also cope with more than one task. For example, if you are doing something on your system that takes a long time, UNIX allows you to start doing another task while you are waiting for the first one to finish. This ability to run tasks concurrently is known as **multi-tasking**.

The concept of a computer operating system, what it does and how it does it, is beyond the scope of this guide. If you would like more information, have a look at a suitable book – for example, *Fundamentals of Operating Systems* by A M Lister.

The UNIX operating system can be split up into three parts:

- The core ‘operating system’ – this controls the computer hardware and at its centre is the core of the UNIX system, the **kernel**. The kernel controls access to the computer hardware, manages the computer memory and allocates computer resources between the various tasks the computer is performing.
- The **commands** – UNIX has many commands designed to help you with your work. There are commands for electronic communication, text editing and layout, system administration, and commands to help you with program development. A typical programming language, like C, can be used in combination with these commands to develop an application program suitable for running on a UNIX system. You may already have such an application running on your system that has already been written from this environment – for example, the X Window System or a desktop etc.
- The **shell** – this is the part of UNIX you see on the screen. It takes your typed input and ‘interprets’ what you type, so that the computer can process it. The shell is also a programming language in its own right that can be used for a variety of purposes. For example, to set up a personalised user environment, redirect input and output to files and run commands in the background. Although the shell shares many of the characteristics of standard UNIX commands and can be used as a standard command, its role as a command processor sets it apart from the other commands in UNIX.

If you like, you can visualise these three parts of UNIX as a series of interdependent concentric rings with the hardware of your system at the core and the application you are running, if any, on the periphery.



The kernel is a program called `vmunix` which controls the hardware of your system. It is loaded when the system is started and runs continuously until the system is shut down. The commands and applications programs are tools which you use to do your work. Sandwiched between these two layers is the shell which acts as the interface between the kernel and you.

The rest of this chapter elaborates on each of the three parts of the UNIX operating system. Therefore, let's look first at the kernel and its related topics.

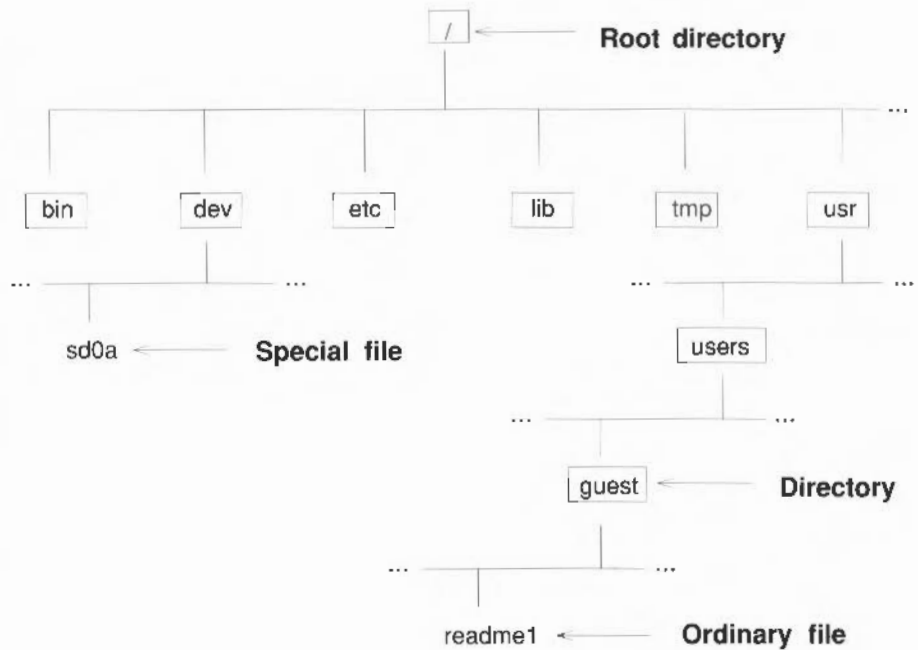
The kernel

Fortunately for you as a user, there is very little of the kernel that you are exposed to. Most of the duties it performs are done automatically and without the need for your intervention. However, the one feature of the kernel that you do see, and need to appreciate before you go any further, is the filing system.

The UNIX filing system

The UNIX filing system or file system for short, provides you with a simple and efficient means of storing, organising and referring to all the information available on your system.

It is based upon a hierarchical tree structure. For example, the diagram below shows the typical structure of part of the UNIX file system:



UNIX recognises three different types of file that together comprise the complete file system:

- An **ordinary file** is simply a collection of alphanumeric characters or binary data. These files are used to store textual information or programs that you write.
- A **directory** is a file maintained by the operating system that is used for organising the structure of the operating system as shown above. A directory may contain files as well as other directories called **sub-directories**. These sub-directories can in their turn contain both files and further sub-directories (sub-sub-directories), and so on, theoretically *ad infinitum*.

Directories normally contain a set of related files. For example, you may create a directory containing files representing all the memos you write – this is good file system management as it helps you to keep track of the files you create.

In the file system diagram, all directories are placed in boxes to distinguish them from files.

- A **special file**, which is the file used to represent a physical device on your system, such as your floppy disc drive. There is at least one special file corresponding to each physical device on your system.

Some operating systems require you to be specific about which type of file you are using and limit you in the ways you can use each type of file. With UNIX however, this is not the case – all files, even special files, are treated alike.

This not only simplifies the structure of the file system but makes it easy for you to use. For example, if you need to access your floppy disc drive in the course of a program you are writing, you just specify the name of the device as you would any other one of your files.

The root directory

The '/' symbol represents the **root directory**. All the other files and directories that comprise the UNIX file system are below this directory. Before looking at where you fit into the file system, let's have a look at some of the more important directories contained in the root directory:

/bin	Contains executable versions of the most common UNIX commands that you will use. For example, the command <i>cat</i> that you used in the <i>Operations Guide</i> to display a file, lives in this directory.
------	---

/dev	Contains the special files that UNIX uses to represent the physical devices that you have on your system. For example, this directory contains a special file for your hard disc drive.
/etc	Similar to /bin, but contains the programs and files that a system administrator uses. These are usually collectively referred to as the System Administrator's Toolbox .
/lib	Contains most of the available programming and language libraries that are installed on the system.
/tmp	A directory where you can create and store temporary files.
/usr	Contains more UNIX commands and software libraries similar to the those found in /bin and /lib respectively and also a users directory for users of the system.
	/usr/bin – executable versions of some of the less common UNIX commands.
	/usr/lib – less common programming and language libraries.
	/usr/users – directory where you and other users of the system normally store their files and directories. Each user will have his or her own directory below which are further sub-directories containing their files. For example user guest uses the directory /usr/users/guest.
	Note that this directory may not always be called users and may not even be in the directory usr – on your system this directory may be called /usr2 or /u or something similar.

Your place in the file system

In short then, the directory `/usr/users/username`, or some similar name, is the place in the file system where you create files and directories and is the only directory that is structured and controlled by you. The other parts of the file system are either controlled by other users or controlled automatically by the UNIX operating system.

Be aware however, that the structure we have defined is merely a quick sketch of what is contained in the file system and where certain types of files reside. As UNIX has evolved, the guidelines for locating certain types of files have altered and many inconsistencies have resulted. For example, as described above `/usr` has a directory called `/usr/bin` that contains files similar to those found in `/bin`. Therefore, as you search through the file system, don't be surprised to find similar types of files located in different directories!

Also, particularly if your system is on a network, your system may have had its structure changed by your System Administrator to suit the needs of the resources available in your computing environment and to make more efficient use of them.

When you first log in, UNIX places you automatically in a specific part of the file system called your **home directory**. This is normally under `/usr` - ie our example home directory for user `guest` is `/usr/users/guest`.

Any commands you give at first will normally take effect on your home directory, unless you specify otherwise - for example, if you issue the `ls` command without any options, to list files and directories, it is the files and directories in your home directory that will be listed.

Within your home directory, you can create files and additional directories to organise them, you can move and delete these files and directories, and you can control who can access your files and directories. You have full responsibility for everything you create in your home directory because you own it.

Your home directory is a vantage point from which to view all the files and directories it holds. It is also a point from which to view the rest of the file system all the way up the directory tree to the root directory.

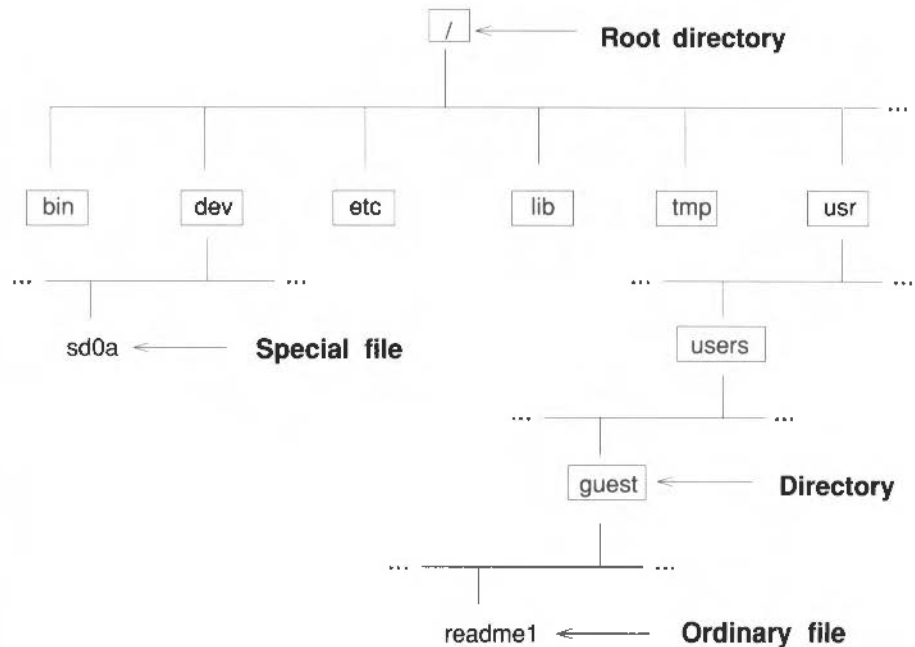
When you first log on, your home directory is your **current working directory** (CWD). To access files and directories in another directory, you have either to change your current working directory, or specify a **pathname** that points to another part of the file system.

Pathnames

As you can imagine, with all the files and directories which are created over a period of time, it could get pretty confusing finding your way around the file system. This is one of the reasons why some form of addressing mechanism is necessary in order to identify the location of files and directories.

A pathname is an address of a file or directory. For example, if your current working directory as shown in the diagram below is `/usr/users/guest`, then there is no confusion if you address the file `readme1`. The system will understand that the file referred to is the one in your current working directory.

However, if you want to refer to the special file `sd0a`, which is down a completely different track, you have to give a full pathname telling the system how to get there. It's like being in a town – if you're already in Mafeking Terrace, there is no confusion if you are asked to go to number 7. But if you're asked to go to number 5 Jubilee Street in another town altogether, you have to be given more information than just number 5.



Finding the address of a house depends (to some extent, anyhow) on where you are standing at the time. Similarly, finding the address of a file in the file system depends on which is your current working directory at the time.

Pathname syntax

A typical full pathname would be written:

```
/usr/users/guest/readme1
```

The initial `/` refers to the root directory. Following this (below root in the directory structure) is `usr/users`, which is the home directory of user `guest`.

`guest` is a sub-directory of `users`, and `readme1` is a file within the `guest` directory.

Files and directories are all separated by the character `/`, so you can use this syntax to refer to any file or directory in the file system.

Security in the file system

Because the UNIX operating system is a multi-user system, you are not working alone in the file system – you and other system users can follow path names and run system commands to move to various directories and to read and use files belonging to one another, if they have permission to do so. So you may choose to protect your files and directories against an unwanted or accidental intrusion.

In general, the files and directories created by individual users can be protected by those users, since they are the owners. The important files and directories that are created in the root directory are owned by a **super-user** who uses the login name `root` and who looks after your system.

If your system is on a network then it is usually your System Administrator who is `root`. If you are the only user of your system, you may also be `root` and it will be up to you to look after your system. For more information, refer to the *RISC iX System Administrator's Manual*.

root – the super-user

`root` is a privileged user with access to all the files and directories on the system, including yours. `root` looks after the security of the system by controlling whether you can use the system and most importantly, where on the system you are allowed to create and remove files and directories.

Protecting your files and directories

Do not be put off by the idea of someone else being able to read your files. `root` is a trusted user of the system and will prove a useful ally in helping you find your way around the system and hopefully, saving you from potential disasters such as accidentally deleting all your files.

The amount of security on the system is left to the discretion of `root`. However, within the confines of the restrictions on security imposed by `root`, you are able to protect your files and directories against other ordinary users, sometimes referred to as 'mortals'.

For more information about `root` and how to log in as `root`, refer to the *Operations Guide*.

UNIX allows you some degree of security on the files and directories you create by allowing you to determine their **access permissions**. For example, who is allowed to read, alter and execute your files and more importantly, who is not.

In the file system, users can act independently of each other but they can also act under the umbrella of a group name. For example, a team of programmers working on a new windowing system could be in a group called `windows`.

`root` normally defines which group you belong to and also which other groups you can have access to. For example, one of the programmers in the group `windows`, may be a regular contributor to the company newsletter – this is a group called `newsfolk`. So `root` can also assign the programmer to be able to access the files owned by this group as well. The members of this group may even be placed in a sub-directory of the home directory. For example, `/usr/users/windows` followed by their user name.

The benefit of being in such a group is that you can allow its members to have special privileges in terms of access to the particular files and directories that the other users of the system who are not in the group or do not have access to the group, are denied. For example, the directory containing the latest version of the source-code for the new window system could be set up so that only the team of programmers in the group `windows` could edit the files in that directory.

Types of protection

If your system has been set up in groups, you may find that you are in an extra directory beneath `/usr/users`. For example, suppose there are three users; `wshakespeare`, `thardy` and `lcarroll` on your system. They could be placed in a group called `writers` and the home directory for `wshakespeare` would then be:

```
/usr/users/writers/wshakespeare
```

and `wshakespeare` could protect his files so that only the users belonging to the group `writers` could access his home directory and the files in his home directory.

The access permissions that you can attach to each file and directory that you own, can be split up into three types:

- permission to read a file and copy its contents
- permission to write changes into a file
- permission to run an executable file
- none of the above, ie no access.

Each of these types of protection can be applied to one of three different classes of user:

- the owner – called the **user** of the file
- the **group** to which the owner belongs
- **other** users of the system.

For example, you can protect one of your files so that only you have read and write access to the file, members of your group have read access and users outside your group have no access.

When you first create a file, the initial access permissions are set up by the UNIX system. But once you have created your file you are free to change these permissions as often as you like.

The access permissions that you set up for files can also be set up for directories but the meanings of the permissions are slightly different.

For example, permission to **read** a file means you can examine its contents – permission to read a directory means that you can display the names of the files in that directory but you are not able to read the contents of the files themselves or move to that directory. This has to be set up for each individual file.

Write permission to a directory means that you can add new files to that directory and likewise remove files from that directory even if you have no write permission for these files! So be careful about setting write permissions on directories as they override the permissions that you have set for each individual file contained in the directory.

Execute permission is required for you to access the files contained in that directory and the sub-directories beneath it. For example, if you have execute permission for a directory, but no read permission, then you can use files in that directory as long as you know their name – without read permission you cannot list the contents of the directory.

You may at first find the option of protecting files and directories against yourself slightly peculiar. However, experience shows that quite often the biggest danger to the existence of a file is the owner of that file!

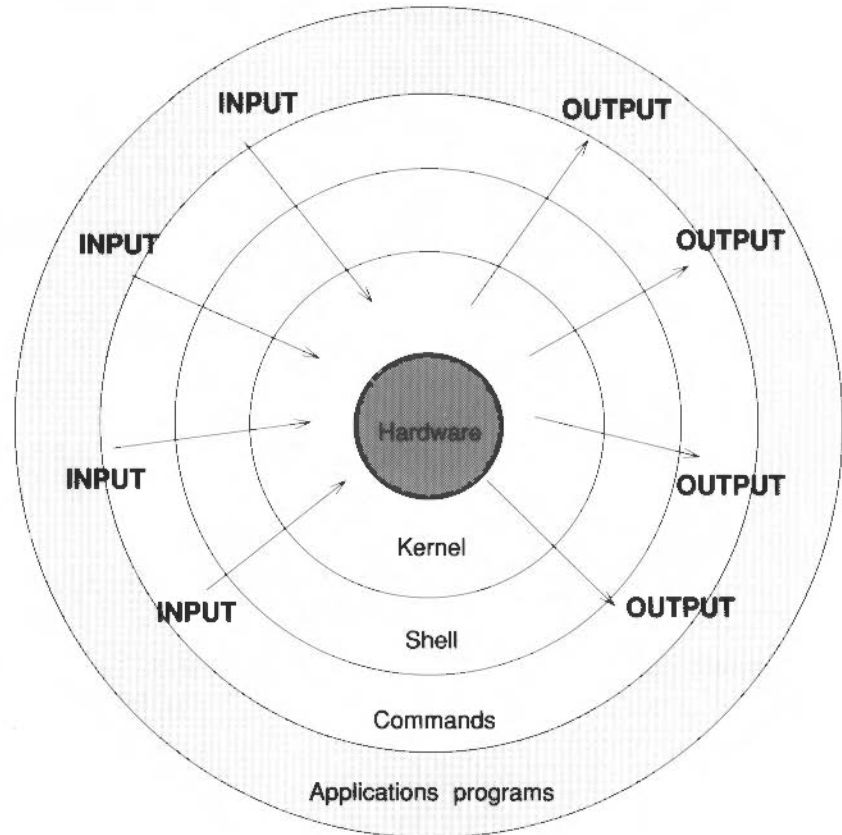
Although you will rarely deny yourself permission to read one of your own files, it is quite often useful to prevent yourself from accidentally writing over an existing file by denying yourself write access permission for the file. For example, if you are editing a new file and decide to save it under the name of a file that already exists, you will normally receive a warning message from the editor that you are using.

The actual UNIX command you use to set the protection on your files and directories is explained in the next chapter, *Using UNIX*.

Input and output - the shell

Now that you are familiar with the UNIX file system, the next step is to introduce you to the means by which the information you input to the system is understood and acted upon and how the system outputs any information back to you – ie how you interact with the system.

All your interactions with the system are controlled by the **shell**. The shell is a UNIX program that acts as an interpreter between you and the heart of the UNIX system, the kernel. Remember the diagram at the start of this chapter:



The shell *wraps* around the kernel and acts as the interface between the kernel and you by running commands when you type them, redirecting input and output and expanding wildcard characters etc.

When you type a command, the shell translates your request into a language that the kernel understands and can act upon: for example, by calling requested programs into memory and executing them.

Because of this ability to translate your commands, the shell is known as the **command line interpreter (CLI)**.

The request you enter is considered input, and the shell takes this input and searches through one or more directories for the program you specified. When the program is found, the shell alerts the kernel. The kernel then takes over and follows the program's instructions and executes your request. When this is complete, the shell takes over again and asks you for more input or tells you it is ready for a fresh command.

As well as being a command line interpreter, the shell is also quite a powerful programming language that you can use to tailor how you interface to the system to suit your own needs and requirements. For example, you can redirect any messages generated by a program to a file, instead of back to your screen.

You can also use the shell to make the output generated by one program be the input of another program – this facility is probably one of the most powerful attributes of the UNIX system.

The capabilities of the shell are fully described in the chapter, *Using the UNIX shell*.

The commands

The commands of the UNIX operating system form a set of individual programs that you can run separately or in combination to produce results that you can use.

For example, `ls` is a command you have already met in the *Operations Guide*, which lists out the files and directories contained within a directory. Most operating systems have commands like this but the advantage of most of these commands in UNIX is their added flexibility.

For example, `ls` lists out files in lexicographic order, but you can use `ls` with an option to list out the files in chronological order instead.

Using the features of the shell you can combine these quite simple software commands to produce very powerful tools for processing text, managing information, communicating with other systems and users etc.

There is also a suite of program development commands that you can use with a standard programming language like C, to create sophisticated applications programs that run under UNIX. For example, graphics-based desktops, business software packages etc. In fact, your system may be supplied with such an application.

The most common of these basic UNIX commands and how to use them, is described in the next chapter, *Using UNIX*.

Using UNIX

Introduction

This chapter assumes that you have logged in to your system as guest.

Now that you have got some idea of the basic principles of the UNIX operating system and how it works, this chapter moves on and introduces you to the basic tools and commands of UNIX.

To benefit from this chapter, read through the text and try some of the examples on your own system as you proceed.

Be warned however, although UNIX has many commendable features, one feature that is sorely lacking is any sort of diagnostic information for the novice UNIX user.

Most of the commands described below execute silently to the user if they work correctly and in some cases remain just as quiet if they don't! So just because you didn't get an error message back when you typed a command, don't presume that your command worked. Moreover, if you do receive an error message it is sometimes decipherable only by a UNIX expert.

The commands in this chapter and throughout the rest of the guide will be described with sufficient clarity to circumvent any problems, but if you are in any doubt, ask a more experienced UNIX user or have a look at the reference section *Trouble-shooting*, at the back of this guide.

The commands described in this chapter are grouped as follows:

- commands to help you find your way around the file system
- commands to create and manipulate files and directories
- a selection of some of the most useful miscellaneous commands.

Many of the commands described have additional, more sophisticated uses that are more relevant to the advanced UNIX user and are not documented here. However, at the end of this chapter you are directed to other sources of information that describe each command in more detail.

Format of commands

Before introducing the commands and before you start trying any of the commands on your workstation, read the following section which shows the structure of a typical UNIX command.

The format for typing commands in UNIX is:

```
commandname [options] [arguments] ...
```

or:

```
commandname [options] arguments ...
```

where:

commandname is the name of the command that you want to execute.

[*options*] is one or more of a number of optional modifiers which affect the way the command behaves.

[*arguments*] is one or more of a number of optional files or directories on which the command is to operate.

... dots that specify any number of *arguments* may be entered.

The brackets around the words indicate that the command can be used without having to specify any options or arguments at all. If there are no brackets, then an option or argument must be included.

For example, the command to list out the contents of a directory is `ls` (short for *list*). The command format of `ls` would be shown as:

```
ls [options] [arguments] ...
```

The above format shows that `ls` can be used without any options or arguments. For example:

```
ls
```

`ls` can also be used with just an option. Options usually begin with a minus sign to distinguish them from arguments and each option is normally represented by a single lower-case letter.

For example:

```
ls -l
```

The `-l` option is short for *long* and gives a more detailed listing than that given by the command `ls`.

`ls` can also be used with a series of arguments. In the case of `ls`, these arguments are just names of existing files and directories, which can just be appended alongside the command. For example, to list the files contained in a specific directory, you could type:

```
ls directoryname
```

The command can be used with both options and arguments:

```
ls -l filename
```

which gives a long listing of the file specified.

You can even specify multiple options and arguments on the same line. For example:

```
ls -l -t directoryname filename
```

which gives a long listing and also sorts the file and directory specified, by the date last modified (the `-t` option).

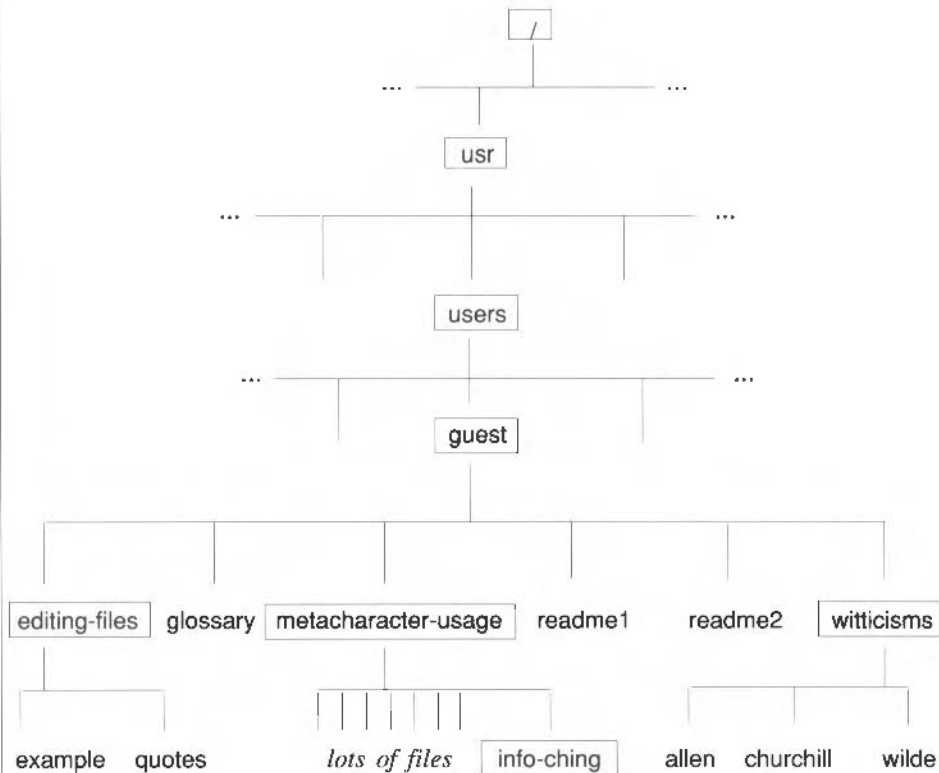
This can be abbreviated to:

```
ls -lt directoryname filename
```

Other instructions may also be added to the command line: for example, to redirect the output produced from such commands. These instructions together with how to use them are discussed in the next chapter, *Using the UNIX shell*.

Finding your way around the file system

To show you how to find your way around the file system, let's use the structure in your current directory (`/usr/users/guest`) and show how the commands can be employed to move around this file system:



The next few sections describe the commands to move around the above file system. If you haven't already done so, log in to your workstation as `guest` and try out the commands. You can then use the commands you learn to move around the entire file system.

Finding out your pathname

To find out where you are in the UNIX file system, type:

```
pwd
```

and your current working directory will be displayed on your screen (`pwd` is short for *print working directory* – although it displays the information on your screen rather than printing it on paper, it is still a useful mnemonic).

For example, if you log in as `guest` and at the normal shell prompt type `pwd`, your location in the file system would be shown as:

```
/usr/users/guest
```

which is the home directory of `guest`. This command is also useful when you lose track of where you are in the file system, which quite often happens at the beginning!

So as a quick check of where you are in the file system you could type `pwd` followed by the command `ls -l` that will show you all the files and directories that are contained in the directory you are in.

Changing your current working directory

To change your current working directory, type:

```
cd directoryname
```

To use `cd` (short for *change directory*), the *directoryname* must be the name of a directory within your current working directory – in other words, a sub-directory one level *below* where you are at that time. To jump any further across the file system requires you to use pathnames.

For example, to move to the directory `witticisms`, you could simply type:

```
cd witticisms
```

your new current working directory would then be displayed as, `/usr/users/guest/witticisms`. However to move back up the directory tree and into the directory `editing-files` you would have to type:

```
cd /usr/users/guest/editing-files
```

The example `/usr/users/guest/editing-files` is a full or **absolute** pathname, because it will get you to the directory `editing-files` wherever you are in the directory structure.

Getting back to your home directory

When you specify a pathname beginning with the '/' character you are telling the system to start searching for the new path from the top of the directory tree. So as long as you type the name of the directories in the pathname correctly and these directories exist, the system will always find the new directory.

Wherever you are in the directory structure, you can get back to your home directory by typing:

```
cd
```

Your home directory will become your current working directory. This is a useful command to have at your fingertips, when you want to return quickly to your home directory.

For example, if you are in the directory `editing-files` and you type `cd`, you are returned to your home directory, namely `/usr/users/guest`.

Abbreviating your CWD pathname

To save typing, you can abbreviate the pathname of your current working directory (CWD) to a '.' character. This can save a lot of typing when, for example, you want to copy a file from a directory in another part of the file system into your current working directory.

For example, the command `cp` (short for *copy*) could be used as follows:

```
cp /usr/users/guest/witticisms/wilde .
```

The above command would copy the file `wilde` from your directory called `witticisms` to your current working directory. For more information on the `cp` command, see the later section *Copying files*.

Substituting your parent directory pathname

Another useful substitution is two full stops '..', to signify the pathname of the **parent directory** of your current working directory. The parent directory is the one above your current working directory. In other words, if you are in the directory `/usr/users/guest/witticisms` your parent directory would be `/usr/users/guest`.

For example, if your current working directory was `/usr/users/guest/witticisms` the `mv` command (short for *move*), could be used as follows:

```
mv allen ..
```

This would move the file `allen` from the current working directory (`/usr/users/guest/witticisms`) to the parent directory, `/usr/users/guest`.

For more information on the `mv` command, see the later section *Renaming and moving files*.

You can also use this abbreviation to switch to other directories – for example, to make the parent directory the current working directory, type:

```
cd ..
```

This command would now make `/usr/users/guest` the current working directory.

From the same position in the file system, the command:

```
cd ../..
```

would now make `/usr/users` the current working directory.

The above pathnames are referred to as **relative** pathnames because they move you around the file system relative to your current position in the file system.

Although these abbreviations are useful, keep in mind that you can always use an absolute pathname in place of a relative one. This ensures that you are copying and moving files to the right place.

The following section introduces the commands used to create and then manipulate files and directories:

- creating and naming files and directories
- displaying and listing files and directories
- renaming and moving files and directories
- copying files and directories
- removing files and directories.

The simplest way to create a new file is by using the command `cat` (short for *catenate*). Type:

```
cat > filename
```

Creating and manipulating files and directories

Creating and naming files

where *filename* is the name you wish to give to the file you are creating. Having decided what you want to call your new file, if you want to put some text into it, just carry on typing, remembering to type ↵ at the end of each line of text.

When you have completed entering the text, press:

```
<CTRL-D>
```

on a line by itself. This indicates to UNIX that the file should end here. The file is now created.

If you list the contents of your directory using the command `ls`, you can confirm that the file has been created.

You can call a file almost anything you want, but avoid using any of the following characters in the name – these characters have special meanings to the shell:

```
\ > < | & ? $ [ ] * ( ) # ! " ' % ~ ^ { } ; @
```

Also, don't put spaces or '/' in the middle of filenames, or start filenames with a '.' (this indicates a **hidden file**).

You should also not attempt to call the file, '.' or '..' as again, files called this have special meanings to the shell.

You will discover all about the uses of these special characters and hidden files in the next chapter, *Using the UNIX shell*.

To avoid any potential confusion, filenames should be composed only of the following set of characters:

```
0-9 a-z A-Z . _ , - +
```

The use of '-' as the first character of a filename clashes with its universal use as the *flag* character for program arguments, and is therefore strongly discouraged. Also do not start filenames with a '+'.

Creating and naming directories

You can create a new directory to keep your files in, with the command `mkdir` (short for *make directory*). Type:

```
mkdir directoryname
```

This creates a new sub-directory, called *directoryname*, within your current working directory.

Note that the conventions already discussed for naming files apply equally to naming directories.

Displaying files

To display an ordinary text file using `cat`, type:

```
cat filename
```

The file will be typed up on the screen. If it is a long file, it will scroll too fast for you to read. Press `<CTRL-S>` to stop it scrolling, and `<CTRL-Q>` to restart.

To display the file one screenful at a time, which is much easier to read, type the command:

```
more filename
```

The first page of the file will be displayed. If there is more information in the file, press the space bar to display subsequent pages, until you reach the end of the file.

If you want to quit from `more` before you reach the end of the file, type `q` at the `--more--` prompt. To read the list of other options available with `more`, type `h`.

Be careful not to try and display the contents of directories, since directories contain unprintable characters. If you try to display a directory using the command `more` you will get the error message:

```
*** directoryname: directory ***
```

If you try to display a directory using the command `cat` you will receive garbled output.

The file system diagram at the start of this chapter distinguishes directories by placing them in boxes, so do not try to `cat` or `more` any of these.

Listing files

To display a list of the files and directories you have created, type:

```
ls
```

The files and directories will be printed out on the screen in lexicographic order. To produce a longer listing of the files and directories, type:

```
ls -l
```

For more information about the output produced from this command, refer to the section *Protecting your files* later on in this chapter.

To list out the files and directories contained within a particular directory, type:

```
ls directoryname
```

The files and directories in the specified directory will be listed out.

Renaming and moving files

One of the idiosyncracies of UNIX is the dual purpose of some of its basic commands. For example, the command `mv` can be used to rename files and also to move files to another directory in the file system.

To change the name of a file using `mv`, type:

```
mv oldfilename newfilename
```

This command renames `oldfilename` as `newfilename`. The `oldfilename` disappears, but its contents live on as `newfilename`.

Where two files already exist, you can use this command to replace one file by another file. For example, the command:

```
mv sun moon
```

would replace the existing file called `moon` by the file `sun`. This effectively deletes the file called `moon`, and you won't be able to get it back, so be careful how you use this command.

If you want to place a number of files into a directory – in order to tidy up the structure of your file system, for example – first create a directory using `mkdir` (see the previous section, *Creating and naming directories*). Then, either move the files one at a time into the directory, using the `mv` command.

Renaming and moving directories

For example:

```
mv filename directoryname
```

or to save yourself some time, use the `mv` command to move the files all at once, by typing:

```
mv filename1 filename2 filename3 directoryname
```

See also the section *Copying a file into a directory*, later on in this chapter.

The `mv` command works for directories just as it does for files. To move or rename a directory, type:

```
mv directory1 directory2
```

If the directory `directory2` does not already exist, then the existing directory, `directory1`, is renamed as `directory2`.

If `directory2` is the name of a directory which already exists, the contents of `directory1` (including all its files and sub-directories) will be effectively moved into `directory2`, and become a sub-directory of `directory2`.

For example:

```
mv planets solarsys
```

moves the directory `planets` into the directory `solarsys` (providing it exists). `planets` thus becomes a sub-directory of `solarsys`.

Be careful when using this command. For example, if you try renaming a directory as a filename that already exists, then the directory is renamed but the file is deleted.

Copying files

To make a copy of a file, type:

```
cp filename1 filename2
```

where `filename1` is the name of the file to be copied, and `filename2` is the name you want to give to the copy you have made. `cp` doesn't affect the contents of the original file at all.

Be careful, however, not to give your copy of the file an existing filename by mistake, as the newly-made copy will overwrite the contents of the existing file, and delete it.

Copying a file into a directory

To make a copy of a file and place it in another directory, type:

```
cp filename directoryname
```

The original file you copied will remain where it was and a separate copy of the same file with the same name, will appear in the directory specified.

Deleting files

You can delete files you no longer need, freeing storage space for new work, with the command `rm` (short for *remove*). Type:

```
rm filename
```

where *filename* is the name of the file you want to delete.

Removing directories

Use the command `rmdir` (short for *remove directory*) to remove directories you no longer need. They must be empty before you can remove them.

Type:

```
rmdir directoryname
```

If the directory is not empty, you will get a message reminding you of this and nothing will be removed. You will have to delete **all** the files that the directory contains (see previous section) before you can delete the directory.

Alternatively, you can delete a directory and all the files and subdirectories it contains, using the command `rm -r`.

For example:

```
rm -r directoryname
```

This command will delete the directory specified, whether or not it contains any files.

It will also delete all the other files and directories beneath it, regardless of whether they are empty.

This is obviously a very dangerous command, so should be used with caution, otherwise you might delete much more than you wanted to.

If you want the system to check each of your removal commands, use `rm` with the `-i` option. When this option is specified, you will be asked to confirm your deletion.

For example:

```
rm -i filename
rm: remove filename?
```

Answering `y` to the above prompt removes the file. Answering anything else stops the file from being removed.

If you have specified more than one file to be removed, `rm` moves on and asks you about removing the second file. For example:

```
rm -i filename1 filename2
rm: remove filename1? n
rm: remove filename2? y
```

In this example, only `filename2` is removed.

Miscellaneous commands

This section describes a few of the more useful miscellaneous UNIX commands, covering:

- sending files to the printer
- protecting your files
- accessing reference documentation.

Printing files

Assuming that you have a printer, you are connected to it, and it is set up and working, type the command `lpr` (short for *line printer*) with the name of the file you want to print. For example:

```
lpr filename
```

The file will be printed out on your printer. After issuing this command, you can examine whether your file has been sent successfully to the printer using the command, `lpq` (short for *line printer queue*).

At your normal shell prompt, type:

```
lpq
```

you will receive the following type of information back:

Rank	Owner	Job	Files	Total Size
active	lcarroll	709	/tmp/chapter5.doc.F438	16238 bytes
1st	lcarroll	710	/tmp/reports.J438	17834 bytes

The list displays a summary of the contents of the printer's **spooling queue** – The printer can only print one job at a time so the spooling queue acts like an *in-tray* for the printer by holding the files to be printed in a temporary spooling area on the disc, until the printer is ready to receive and process the file.

For each job submitted (that is, each time you type `lpr`), `lpq` reports the user's name, current rank in the queue, the names of files comprising the job, the job identifier (a number which may be used for removing a specific job, see below), and the total size of the file in bytes.

In the above example, user `lcarroll` has submitted two jobs to the printer. The first file (`/tmp/chapter5.doc.F438`) is currently being sent to the printer and is printing. The second file (`/tmp/reports.J438`) is next in the queue and will be sent to the printer as soon as the first file has been printed.

Removing an entry from the printer queue

To remove entries from the printer's spooling queue, use the command `lprm` (short for *line printer remove*) followed by the job number of the entry you wish to delete.

This command can only be used to remove jobs that you have placed on the queue – ie jobs that are owned by you. In the previous example, if user `lcarroll` wishes to remove the file called `/tmp/reports.J4388` (job number 710) from the printer queue, he types:

```
lprm 710
```

The system removes the specified job from the temporary spool area on the disc of the workstation called `acorncpd` as shown by the following message:

```
dfA710acorncpd dequeued  
cfA710acorncpd dequeued
```

Protecting your files

Self-explanatory messages should be issued if there is no such job number, the job is not owned by you or the queue is empty. However, UNIX, as you will discover, tends to be very terse and will only respond if your command has been successful.

Unfortunately, the commands and procedures outlined above can quite easily be fraught with problems when you try using them. For more information about printing and the problems that can occur, see the *RISC iX System Administrator's Manual* or consult your System Administrator.

The previous chapter introduced you to the concept of protecting your files and determining different access permissions for different classes of user. This section introduces the command you use to protect your files and directories and the options available with the command.

If you own a file, then you are able to determine who has the right to read that file, to make changes to or write to the file, and to run or execute the file if it is a program.

Determining existing permissions

To determine what permissions are currently in effect on your files and directories, use the `ls` command with the `-l` option to specify a long listing of the contents of the directory. For example:

```
ls -l
```

If you are in the directory `/usr/users/guest/editing-files`, the above command will display the contents of the directory in a form similar to the following:

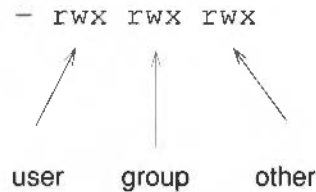
```
-rw-r--r-- 1 guest      274 Oct 20 09:52 example
-rw-r--r-- 1 guest      269 Oct 21 10:46 quotes
```

The permissions for the two files are shown on the extreme left of the display.

The first character, '-' is not relevant to access permissions and is used to determine the type of file. The following nine characters represent three groups of three characters.

The first set of three characters refers to your permissions as owner of the file, the second set to members of the group to which you belong, if any, and the last set refers to the access permissions for all other system users.

For example:



Within each set of characters, the `r`, `w` and `x` indicate the access permissions currently enabled for each of the three groups.

These permissions are defined as follows:

- `r` allows users to read a file or to copy its contents
- `w` allows users to write changes into a file
- `x` permits users to run an executable file
- `-` means that the access permission is not set (so that no access is permitted to the file).

So, for example, the file `example` has read and write permissions set for the owner (`guest`) and only read permission set for group and other members.

Changing existing permissions

To change the access permissions, use `chmod` (short for *change mode*) according to the following format:

```
chmod who+/-permission filename
```

where *who* refers to one, or more, of the following:

- `u` for *user* – the owner of the file
- `g` for *group*
- `o` for *other*
- `a` as a synonym for *all users*.

'+' or '-' is an instruction that grants (+) or denies (-) access permission.

permission is the authorisation to read, write or execute the file.

Accessing reference documentation

For example, to give users in your group read and write access to the file `example`, type:

```
chmod g+rw example
```

To deny read access to the file `example` to users not in your group, type:

```
chmod o-r example
```

To change the file `example` so that every user on your system has write permission to the file, type:

```
chmod ugo+w example
```

The letter `a` can be used as a synonym to refer to *all* users. Therefore, you could also type:

```
chmod a+w example
```

to give write permission to all users of your system.

Manual pages are the standard form of reference documentation used to describe UNIX commands. Each command has its own separate manual page detailing how to use it, what the options are, any likely error messages and finally, any *known* bugs.

Although the manual pages are very thorough in describing each command, you may find them a little daunting in the amount of detail they contain. However, most experienced UNIX users prefer them. As your knowledge of UNIX increases you should also begin to use them to find out more about your system.

The *Berkeley 4.3 UNIX User's Reference Manual* contains all the manual pages for the commands introduced so far in this chapter. A selection of these manual pages along with the manual pages for commands specific to RISC iX are contained in the reference section, *RISC iX manual pages* at the back of this guide.

The manual pages for system administrators are in the *Berkeley 4.3 UNIX System Manager's Manual*. A selection of these manual pages along with the manual pages for commands specific to RISC iX are contained in the *RISC iX System Administrator's Manual*.

The manual pages more pertinent to programmers are in the *Berkeley 4.3 UNIX Programmer's Reference Manual*.

A set of all the above manual pages is installed on your system in the directory, `/usr/man` - you can read them from your screen using the `man` command (short for *manual*). At your normal shell prompt, type:

```
man commandname
```

where *commandname* is the name of the UNIX command you are interested in. After a few moments the manual page describing *commandname* will be displayed on your screen.

For example, suppose you want some information about the command `cat`. Type:

```
man cat
```

the manual page for the command `cat` will be scrolled on to your screen.

The manual page is headed `CAT(1)` meaning that the command name is `cat` and this is the entry in section 1 of the *Berkeley 4.3 UNIX User's Reference Manual*. Then comes the name of the command again, with a very brief explanation of what it does.

The **SYNOPSIS** section is important; it tells you the command format for using the command. The synopsis for `cat` is:

```
cat [-u] [-n] [-s] [-v] filename ...
```

showing that to issue the command you type:

```
cat
```

along with any of the optional arguments `-u`, `-n`, `-s`, `-v` followed by the name of the file you wish to read. In the synopsis, anything enclosed in square brackets is optional and doesn't have to be supplied if you don't want it. Any word not enclosed in brackets has to be included on the command line. In the above example, this means you have to supply at least one filename. The dots mean, in this case, that more than one *filename* can be entered.

Thus, a valid command for `cat` would be:

```
cat -n comedies notes
```

Accessing online help

Other information provided in a manual page will be a description of all the available options, other manual pages of similar commands you could refer to, likely error messages (DIAGNOSTICS) and any *known* bugs in the command.

The remainder of this guide will refer to manual pages that you can consult for more information about a particular topic or command – ie for further details about manual pages, see `man(1)`. This means, the command called `man` which is described in section 1 of the *Berkeley 4.3 UNIX User's Reference Manual*.

Note that if your system contains the X Window System, you can also use the utility `xman(1)`, which is a graphics-based manual page browser. For more information about the X Window System and `Xman`, refer to the chapter *Using the X Window System* later on in this guide.

An on-line system tutorial called `learn` can be used if you want to find out more about using UNIX. At your normal shell prompt, type:

learn

you will see the following information displayed:

These are the available courses -

```
files
editor
vi
morefiles
macros
eqn
C
```

If you want more information about the courses,
or if you have never used 'learn' before,
press RETURN; otherwise type the name of
the course you want, followed by RETURN.

The `files` and `morefiles` courses cover some of the information that has been discussed in this chapter. The remaining `learn` courses will be referenced at appropriate locations throughout the rest of this guide.

Command summary

The following list is a summary of the UNIX commands that have been described in this chapter.

Command	Syntax	Use
cat(1)	cat <i>filename</i>	Display a file; stop scrolling with <CTRL-S>, restart with <CTRL-Q>.
	cat > <i>filename</i>	Create a new file called <i>filename</i> , end with <CTRL-D>.
cd(1)	cd <i>directoryname</i>	Change current working directory.
	cd	Return to home directory.
chmod(1)	chmod <i>arguments filename</i>	Change access permissions on files.
cp(1)	cp <i>filename1 filename2</i>	Copy <i>filename1</i> to <i>filename2</i> .
	cp <i>filename directoryname</i>	Copy <i>filename</i> into a directory.
learn(1)	learn	Computer aided instruction about UNIX.
lpq(1)	lpq	Examine the printer's spooling queue.
lpr(1)	lpr <i>filename</i>	Send the file <i>filename</i> to the printer.
lprm(1)	lprm <i>jobnumber</i>	Remove <i>jobnumber</i> from the printer queue.
ls(1)	ls	List files and directories in the current directory.
	ls <i>directoryname</i>	List files and directories (for that directory).
man(1)	man <i>commandname</i>	Display the manual page for <i>commandname</i> .
mkdir(1)	mkdir <i>directoryname</i>	Create a new directory.
more(1)	more <i>filename</i>	Display the contents of a file (see also <i>cat</i>).

mv(1)	mv <i>filename1 filename2</i>	Move or rename a file; contents (if any) of <i>filename2</i> will be replaced by <i>filename1</i> .
	mv <i>filename directoryname</i>	Move a file into a directory.
	mv <i>directory1 directory2</i>	Make <i>directory1</i> a subdirectory of <i>directory2</i> if it exists. If not, rename <i>directory1</i> as <i>directory2</i> .
pwd(1)	pwd	Display the current working directory.
rmdir(1)	rmdir <i>directoryname</i>	Remove a directory which is empty.
rm(1)	rm <i>filename</i>	Remove a file.

Sources of further information

For more information about the above commands and the options that can be used with them, refer to the relevant manual page for each command.

Possible error messages that you may receive while using these commands are outlined in the reference section *Trouble-shooting*, at the back of this guide.

Using the UNIX shell

Introduction

This chapter assumes that you have logged in to your system as guest.

The UNIX shell has been described so far purely as a command line interpreter; that is, as a systems program which reads a command line that you type, interprets and executes the command and then returns control back to you, prompting for another command or additional input.

This chapter moves on from the use of the shell in this manner and shows you how to use it interactively to improve your efficiency when using the system.

For example, this chapter describes how you can use the shell to:

- modify existing shell variables
- interpret the name of a file or directory you input in an abbreviated way, using a set of characters that are special to the shell
- redirect the flow of input and output
- write a simple shell program.

Ultimately, you can customise your whole UNIX environment to suit your individual needs and preferences.

As in the previous chapter, read through the text, trying some of the examples and, if you think you understand enough, some examples of your own. The only way to become familiar with using the shell is by practice and experimentation.

Available shell types

There are two shells available for use on your system:

- the standard shell (`sh`), sometimes called the **Bourne shell** after its creator – Stephen R. Bourne
- the **C shell** (`csh`), developed by Bill Joy and others at the University of California at Berkeley, unfortunately **never** referred to as the '*Joy shell*'.

Creating a login shell

Both shells have many similar attributes when you use them interactively. However, their paths begin to diverge quite sharply when used for complex shell programming.

This chapter will only deal with simple shell programming, so where possible, it will try to describe the use of the shell as suitable for both shell types. But where differences occur, the shell described will be the Bourne shell.

Why choose the Bourne shell? Well, most UNIX systems support the Bourne shell, so any shell scripts that you write will be portable to other systems. Also, the Bourne shell executes faster, uses less memory and is the default shell that the system uses to execute programs, unless told otherwise.

Following login, the shell is called to read and execute commands typed by you. However, if your home directory contains a file named `.profile`, then this file is executed once by the shell before reading any commands that you type.

`.profile` is called a **hidden file** but is just a normal file that you create that contains commands to set up shell variables or execute commands.

Files like this are called hidden because they are not seen in normal `ls` listings of directories. `ls` ignores files like these, unless it is used with the appropriate option. For example, to view all the hidden files in your home directory, type:

```
ls -ld .*
```

Below is a typical `.profile`:

```
echo ""
echo "Welcome to the Acorn RISC ix workstation"
echo ""
echo "Today's date and time is:"
date

HOME=/usr/users/guest
PATH=:/bin:/usr/bin
export HOME PATH
```

In `cs`, the comparable file is called `.login`, and is also executed following log in. A typical `.login` file is similar in syntax to the `.profile` file. For example:

```
echo ""
echo "Welcome to the Acorn RISC ix workstation"
echo ""
echo "Today's date and time is:"
date

set home=/usr/users/guest
set path=(:/bin:/usr/bin)
```

The settings of the shell variables shown in both example login shells, are described below:

HOME Is the default directory that is used when the `cd` command is issued with no arguments.

PATH Is the search path that is followed when the shell tries to execute a command you have requested. The above path setting specifies the following search path. Firstly:

- `:` the current directory, then
- `/bin` bin sub-directory of `'/'` directory, and
- `/usr/bin` bin sub-directory of `/usr`.

For example, when you issue the `ls` command, your system looks in the directories specified by `PATH`. The first directory it looks in is `/bin` where `ls` lives – so the program is run to carry out the command.

The `export` command used in the Bourne shell ensures that the values you have assigned for each of the variables (`HOME` and `PATH`) are propagated to the commands and processes that you execute. The C shell takes care of this with the `set` command in the `.login` file.

You can also use your login shell to define, amongst other things, the characteristics of your keyboard, and to specify where your electronic mail should be delivered.

There are also several other hidden files that may reside in your home directory, or that can be created by you, to describe other software applications on your system.

These hidden files normally begin with a *dot* so that they do not show up in a normal 'ls' listing and end in 'rc'. For example, you can create a hidden file to describe the features of an X Window System window manager called *uwm*. The appropriate hidden file is called *.uwmrc*.

In short, hidden files can be used to tailor an application on your system to suit your individual preferences.

The hidden files for the applications available on your system, will be introduced in the appropriate chapters of this guide.

Perhaps one of the simplest and yet most powerful uses of the shell is the use of **metacharacters**. These are special characters that the shell understands and that you can use to save yourself typing when you are using the system. A list of the most useful shell metacharacters and their meaning is shown below:

- * match any sequence of characters including an empty sequence.
 - ? match any single character.
 - [...] match any of the characters enclosed by the square brackets. A pair of characters separated by a minus sign will match any character between the pair.
- The metacharacters above are sometimes called **wildcards** as they are commonly used as shortcuts for referring to filenames and directory names.
- & this metacharacter instructs the shell to run the following command in the background – ie not to wait for the command to finish before returning control back to you, allowing you to carry on working.

Metacharacters and their use

- ; allows you to type more than one command on a line.
- \ ignore the meaning of the following metacharacter.

Examples of the use of these metacharacters are provided in the next few sections.

Generating filenames

Filename generation is the term used to describe the process of referring to groups of files using shell metacharacters. For example, the following files appear in the directory `/usr/users/guest/metacharacter-usage`:

```
ching10 ching9 section3.doc synopsis3
ching11 info-ching synopsis-final synopsis4
ching7 section1.doc synopsis1
ching8 section2.doc synopsis2
```

This section will show you how you can use metacharacters to refer to groups of the above files. If you are unfamiliar with using metacharacters, log in as `guest` and change directory to `metacharacter-usage` (using the `cd` command), and try the examples out on your system.

To list the file `section1.doc` you would type:

```
ls -l section1.doc
```

which would print information about the the file `section1.doc`. However, if you also wanted to list out the other two files, `section2.doc` and `section3.doc`, you could type:

```
ls -l section1.doc section2.doc section3.doc
```

To save time, you could use the shell metacharacter `*`, which will match any sequence of characters including an empty sequence. For example, all the above files end in `.doc`. So, to list all the files in the directory that end in `.doc`, you could type:

```
ls -l *.doc
```

This generates, as arguments to the command `ls`, all filenames in the current directory that end in `.doc`. The shell searches through the directory looking for every file that matches the **pattern** that you have specified – in other words, looking for every filename that ends in `.doc`.

The other metacharacters are used in a similar way to refer to different file patterns.

For example, to list out the files `synopsis1`, `synopsis2`, `synopsis3` and `synopsis4`, type:

```
ls -l synopsis?
```

The metacharacter '?', matches any single character in a filename. So the pattern the shell looks for in this case is every filename beginning `synopsis` and ending in any one single character. Therefore, the file `synopsis-final`, although starting with the correct pattern, is not listed because it has more than one character after `synopsis`.

The '?' metacharacter can also be used more than once. For example, to list the files `ching10` and `ching11`, but not `ching7`, `ching8` or `ching9`, type:

```
ls -l ching??
```

You can also specify a range of characters to search for within a filename, using the metacharacters '[...]'. For example, to list the files `ching7` and `ching8` but not `ching9`, `ching10` and `ching11`, you could type:

```
ls -l ching[7-8]
```

This command sets the shell searching for filenames beginning with `ching` followed by a single character within the range 7 to 8. Therefore, it matches the files `ching7` and `ching8`.

The metacharacters can also be used in combination when you need to specify a file pattern that needs to use the attributes of more than one metacharacter. For example, to list out the files `synopsis2`, `synopsis3`, `synopsis4` and `ching7` and `ching8`, you could type:

```
ls -l *[2-8]
```

This matches any filename with any number of characters ending with a number from 2 to 8.

The use of metacharacters has so far only been described in conjunction with the command `ls`. However, you can also use these metacharacters with other UNIX commands. For example, to move all the files related to `ching` to a directory called `info-ching`, use the `mv` command in the following format:

```
mv ch* info-ching
```

This moves the files that match the pattern `'ch*'` into the directory `info-ching`. Therefore, the files `ching7`, `ching8`, `ching9`, `ching10` and `ching11`, in our example directory, are moved into the directory `info-ching`.

You can also use the `rm` command with metacharacters to remove a set of files. But take care when using the command in this way: more files could easily be removed than intended. One way to reduce the chance of error is first to list out the files that match the pattern and then remove them. For example:

```
ls -l syn*
```

which lists out all the files beginning with `syn` including `synopsis-final`, `synopsis1`, `synopsis2`, `synopsis3` and `synopsis4`. If you are satisfied that these are the files you want to remove, just type:

```
rm syn*
```

taking care not to introduce a space between `syn` and `*`, otherwise the shell interprets the command as 'remove the file `syn` followed by all the files in the current directory'. This is a common mistake, even amongst experienced UNIX users.

To be even more cautious, use the `rm` command, with the `-i` option set. This asks you whether you wish to delete the files specified. For example:

```
rm -i syn*  
rm: remove synopsis-final? y  
rm: remove synopsis1? y  
rm: remove synopsis2? y  
rm: remove synopsis3? y  
rm: remove synopsis4? y
```


Running commands in the background

You can now be assured that you have only deleted the files that you wanted to. This is a good habit to get into when you start deleting files as it is very easy to delete files accidentally.

To execute a command, the shell normally creates a new **process** and waits for it to finish before returning control back to you. However, you can add the metacharacter `'&'` alongside the command to instruct the shell to run the command, but not to wait for the command to finish before returning control back to you.

For example, to compile a C program in the background, use the command `cc` (short for *C compiler*) as follows:

```
cc filename &  
2436
```

calls the C compiler (`cc`), to compile the file `filename`. The trailing `&` instructs the shell not to wait for the command to finish. `2436` is the **process id number** that the command is given by the kernel.

After displaying the process id number, the normal shell prompt returns and you can usually carry on with another task, while the command is running silently in the background. However, if an error occurs during the compilation you may receive an error message on your screen.

You can have more than one process running in the background. At your normal shell prompt, just type each command, with the `'&'` metacharacter appended.

As your experience in using the various tools of UNIX grows, you will find that some of the commands you use will take a while to execute, so this command is very useful as it allows you to continue using the system.

To find out how many processes you have running at any one time, use the `ps` command (short for *process status*) with no arguments:

```
ps
```

This displays a list of background jobs running on your system.

Running commands in sequence

For example, if you are compiling the C program *filename* in the background, there would be an entry in the output from *ps* confirming it was running:

```
PID          TT  STAT      TIME          COMMAND
...
602          co  S          0:00          cc filename
...
```

For more information about the *ps* command and the output it generates, refer to the manual page *ps(1)*.

The metacharacter ';' allows you to type a number of commands on a single command line. The format for placing commands on a single line is:

```
commanda ; commandb ; commandc
```

The shell executes the commands in the order they were typed. Therefore, *commanda* is executed first, followed by *commandb* and finally, *commandc*.

Executing commands in sequence like this is useful when you want to type in a series of quick commands without having to press \downarrow after each command. For example, an often-used combination of commands is to change directory, using *cd* and then list the contents of the new directory, using *ls*. This can all be placed on one line as follows:

```
cd /bin ; ls -l
```

Quoting metacharacters

To be able to quote a metacharacter so that its special meaning is ignored by the shell, the metacharacter must be preceded by a '\'.

For example, *echo* is a UNIX command that displays its arguments on the screen. A simple example would be:

```
echo hello there

hello there
```

This may not at first seem like a useful command, but as you use UNIX more, you will begin to appreciate its applications – for example, outputting diagnostic information to the screen when running shell programs. For more information about the *echo* command, refer to the manual page *echo(1)*.

To use `echo` to display the metacharacter '&', you would have to type:

```
echo \&
```

Note, that if you type the character without a '\', the shell interprets your instruction to run the command in the background. The command is read as `echo` with no arguments.

By prefixing '&' with the metacharacter '\', the first '\' is skipped so that the command is read as, `echo` the character '&'. The metacharacter is said to be **quoted** and loses its special meaning to the shell.

Another example:

```
echo \\
```

will echo a single '\'.

To allow long strings to be continued over more than one line, a '\' followed by `\` is ignored. This is sometimes called a **hidden newline**.

The '\' is convenient for quoting single characters. When more than one character needs quoting you can enclose the string between single quotes. For example:

```
echo 'xx' ?&' xx
```

will echo:

```
xx?&xx
```

Note that the quoted string may not contain a single quote. This method of quoting using single quotes is the simplest and is recommended for casual use.

You may think that this section is rather labouring the point about quoting metacharacters – but be warned, if you intend to use the shell for programming purposes, it is vital that you grasp this idea. It will save you a lot of time in the long run.

Redirecting input and output

One of the most important and powerful tools available when you are using the shell is the ability it gives you to redirect the input and output of commands and programs.

The shell has certain defaults set for where it expects to find the input for a command (called **standard input**) and where it puts the output from a command (called **standard output**). For example:



In the default case a command expects to receive its input from the keyboard (standard input), and to send its output to your screen (standard output).

However, using a few shell metacharacters you can redirect each of these inputs and outputs.

Redirecting output

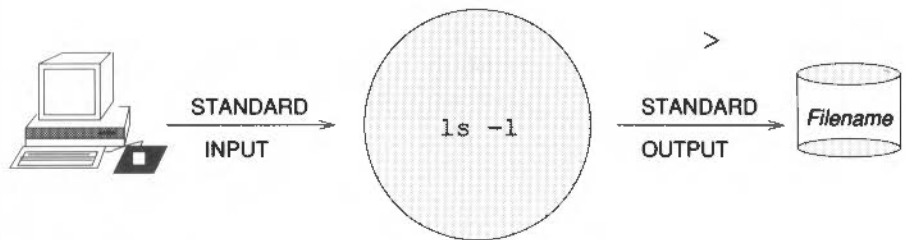
To direct the standard output into a file for further manipulation, editing or printing, use the character '>'.

For example, to redirect the **standard output** from an `ls` command (the information produced on your screen when you issue the command) you could type:

```
ls -l > filename
```

The above command is interpreted by the shell and the output from the `ls` command is redirected to the file `filename` instead of to your screen.

For example:

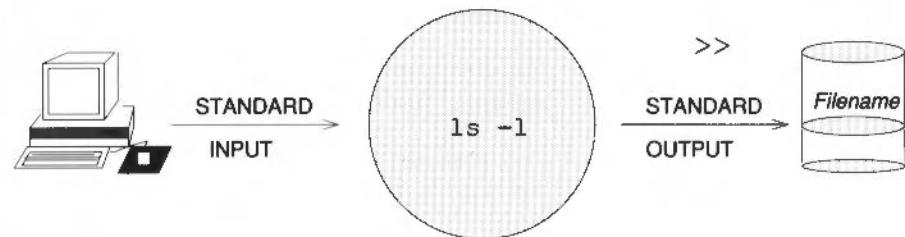


If the file *filename* does not exist then the shell creates it; otherwise the original contents of *filename* are replaced with the output from *ls*.

If you want to append the output to the end of an existing file, you can use the notation '>>', instead of '>'. For example:

```
ls -l >> filename
```

For example:



Again, *filename* is created if it does not already exist.

This is an immensely useful feature of the shell that you can use with many UNIX commands. For example, there is a UNIX command called `spell`, that checks the spelling of the words in a given file and sends all the incorrectly spelled words it finds to the standard output.

If you have a very large, poorly edited file, it would be nice to be able to direct all your misspellings to a file that you could then read at your leisure. This can be done easily. For example:

```
spell -b filename1 > filename2
```

The above command runs the spelling checker program, `spell` on the file `filename1`, with the `-b` option set to specify *British* as opposed to *American* spelling. The offending words produced are sent to the file `filename2` instead of to your screen. Therefore, make sure that you do not use the same file name for redirecting your output to, otherwise you lose the original text file.

Redirecting input

Instead of entering information from your keyboard all the time, you can redirect the text of a file to be the input for a command. For example:

```
mail username < filename
```

accepts input from the file `filename` instead of from the keyboard.

`mail` is a UNIX command that you use to send mail to someone. The command expects one argument with the name of the user to whom you are sending mail. After typing this, press `↵` and type in the mail message from your keyboard and terminate it with `<CTRL-D>`.

The trouble is that once you have started typing the message, there is no easy way of editing the message you are sending.

A nice way around this is to first create a file using your favourite text editor, containing the message you want to send. Then at your normal shell prompt, type:

```
mail username < filename
```

The `'<'` character causes `mail` to read the message text from the file `filename`, instead of from your keyboard.

For example:

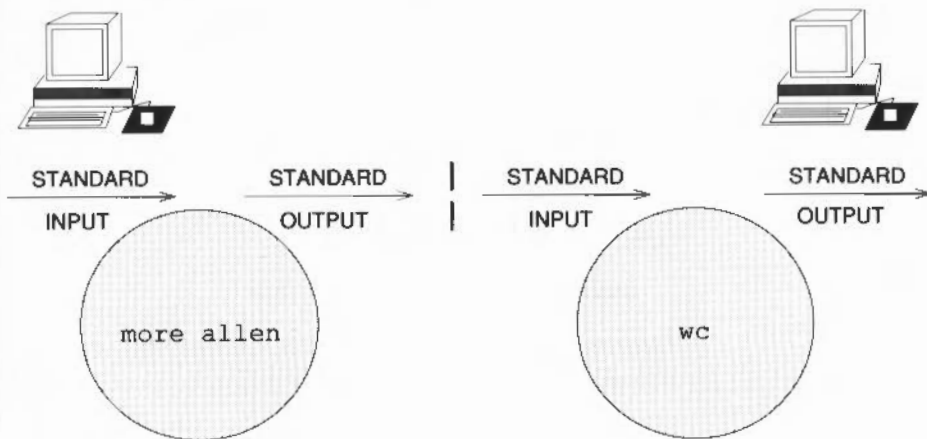


Using pipes

One of the most important and useful features of the UNIX operating system is the ability it gives you to redirect the standard output of one command to be the standard input of another, using the **pipe** operator '`|`' (represented by a broken vertical bar on your keyboard). For example:

```
more allen | wc
```

Two commands connected in this way constitute a pipeline. The file listing output from the `more` command is sent to be used as the input for the command `wc` (short for *word count*) that counts the number of lines, words and characters. For example:



The overall effect is almost the same as:

```
more allen >file; wc <file ; rm file
```

except that no interim file is used. Instead the two processes are connected by a pipe that is created by an operating system call. `wc` waits when there is nothing to read and `more` waits when the pipe is full.

The pipe is particularly useful if you want to direct the output of a command to the printer. For example, to print the manual page for the command `wc`, you could type:

```
man wc | lpr
```

which would send the formatted manual page straight to the printer instead of to your screen.

Pipes are very useful for creating your own tailored commands out of two or more existing UNIX commands. For example, you could use `wc` and `ls` to count the number of files in a directory. For example:

```
ls directoryname | wc -l
```

Using `wc` with the `-l` option instructs `wc` to count only the number of lines in the standard input. As `ls` lists each file on a single line when used with a pipe, this would display the number of files in that directory.

If you want to send output both to your screen and to a file you can use the `tee` command. For example, to display the number of files in the directory and also send this result to a file, you could type:

```
ls directoryname | wc -l | tee file-count
```

The above command displays the number of files in the directory and creates the file `file-count` that also contains this information.

Using filters

Many UNIX commands act in a similar way to `wc`, by reading their standard input if no filename is given, then modifying or using the information in some way and then writing the result to the standard output.

When a command is used in this way it is known as a **filter**. One such filter, `grep`, selects from its input those lines that contain some specified string. For example:

```
ls -l | grep ching
```

prints those lines, if any, of the standard output from `ls -l` that contain the string `ching`.

Another useful filter is `sort`. For example:

```
who | sort
```

The `who` command displays a list of all the people that are using your machine. (Don't be surprised to find other users as well as yourself logged in, if your system is on a network. Remember, because UNIX is a multi-user operating system, there may be other people accessing your machine from another workstation.)

The output from `who` is then sent to `sort` which arranges the list of users logged in into alphabetical order and outputs the result to your screen.

Programming in the shell

As well as being a command line interpreter, the shell is also a powerful programming language. This section describes how to use the shell to write a simple program.

Creating your own commands

You have already seen how to use the shell to concatenate commands so that each command is executed sequentially on one command line.

You can also include such commands in a file and then use the shell to read this file and execute the commands in it.

For example, the sequence of commands:

```
cd /bin ; ls -l
```

which changes directory to `/bin` and then displays a long listing of the contents of that directory, can be placed in a file.

For example:

```
cat > dirlist  
  
cd /bin ; ls -l  
  
<CTRL-D>
```

You can then use the shell to execute the commands contained in the file, `dirlist`. As the file contains shell commands, it is known as a **shell program** or more commonly, a **shell script** that can be invoked by the shell using the `sh` command. For example, to execute the shell script `dirlist`, type:

```
sh dirlist
```

This instructs the shell to take its input from the file `dirlist`, instead of from your keyboard. The `sh` command invokes a shell and the two commands are executed as if they had been typed on the command line.

Notice however, that after the command finishes you are still in the same directory where you issued the command. This is because your original shell has invoked a secondary shell to execute the shell script, `dirlist`. When the shell has finished executing the command, you return to the original shell.

This notion of running multiple shells is an important concept in RISC iX. Shells can be invoked from within other shells which themselves can also invoke other shells.

To make `dirlist` appear like a fully blown RISC iX command similar to `cat` or `pwd`, you need to make the file **executable** by changing the access permissions on the file using `chmod`. By making the file executable, you are telling the shell that it contains commands that can be executed directly by the shell.

For example:

```
chmod u+x dirlist
```

gives execute permissions to the file `dirlist` for the owner of the file.

Following this, the command:

```
dirlist
```

is equivalent to:

```
sh dirlist
```

You have now created your first RISC iX command.

You could improve your shell script to change to and list out the contents of any directory on your system. This can be done by introducing a **substitutable parameter** in your shell script.

For example, using your favourite editor, substitute the directory `/bin` by `$1` in `dirlist`. Your new file should now read:

```
cd $1 ; ls -l
```

`$1` is a substitutable parameter that tells the shell to substitute it by the first argument it finds.

For example, if you type:

```
dirlist /tmp
```

the shell replaces `$1` by the argument `/tmp`, so that the contents of the directory `/tmp` are listed. In effect, the shell executes the command:

```
cd /tmp ; ls -l
```

For more information about using the shell for programming, refer to the manual pages for `sh(1)` and `csh(1)`.

Command summary

The following list summarises the features of the shell that have been discussed in this chapter along with a list of the new commands that have been introduced:

Features of the shell

Shell operator	Meaning
*	Match any character, including none.
?	Match any single character.
[...]	Match any of the set or range of characters enclosed by the square brackets.
\$1	Substitute by first argument found
&	Run a command in the background.
;	Command separator.
\	Quote the next character.
''	Quote a string.
>	Redirect output.
>>	Append output.
<	Redirect input.
	Pipe – redirect standard output to standard input.

Command summary

Command	Syntax	Use
cc(1)	cc <i>filename</i>	Call the C compiler to compile <i>filename</i> .
ps(1)	ps	Print information about the processes running.
echo(1)	echo <i>arguments</i>	Echo arguments.
spell(1)	spell <i>filename</i>	Check the spelling of the contents of <i>filename</i> .
mail(1)	mail <i>username</i>	Send mail to <i>username</i> .
wc(1)	wc	Count lines, words and characters.
tee(1)	 tee <i>filename</i>	Redirect standard output to a file as well as to the screen.
grep(1)	grep <i>pattern</i>	Search for <i>pattern</i> .
who(1)	who	Display list of logged in users.
sh(1)	sh <i>filename</i>	Execute the file <i>filename</i> as a shell script.
sort(1)	sort	Arrange the contents of the standard input.

Sources of further information

For more information about the use of the shell for quoting, use of metacharacters, redirecting input and output and simple programming, refer to the following sources:

- *An Introduction to the UNIX Shell* by S. R. Bourne in the *Berkeley 4.3 UNIX User's Supplementary Documents*.
- *An Introduction to the C shell* by William Joy in the *Berkeley 4.3 UNIX User's Supplementary Documents*.
- `sh(1)` and `csh(1)` manual pages.

For more information about the commands described in this chapter and the options that can be used with them, refer to the relevant manual page for each command.

Possible error messages that you may receive while using these commands are outlined in the reference section, *Trouble-shooting*, at the back of this guide.

Where to go from here?

Well, now that you are familiar with the basics of the UNIX file system and some of the attributes of the shell, you can begin to explore a few of the interactive programs on the system.

Some of the most frequently used interactive programs are those that allow you to exchange and transfer information to other users. The capabilities of your system for these tasks is explained in the chapter entitled *Communicating with other systems and users*.

Another widespread set of programs are text editors. The editors available on your system are detailed in the next chapter.

Later chapters are devoted to describing using your workstation on a network, communicating with other users, using the X Window System and finally, a chapter detailing the more advanced uses of UNIX. Take your pick!

Text editing

Introduction

You will need to become familiar with at least one RISC iX editor for writing programs, creating large files etc. There are many editors available for use on your workstation, this chapter describes two of them – `ed`, a line editor, and `vi` (pronounced *vee-eye*), a screen-based interactive editor.

It is your choice which of these two editors you use. The main considerations are availability, and ease of use.

You will find that `ed` is available on almost any other UNIX system, whereas `vi`, although widely available, is less common. If you are going to use many different UNIX systems, it is therefore a good idea to learn `ed`.

Be warned that although both editors are widely used amongst the UNIX community, neither was designed with the first time user in mind, so many of the commands may appear slightly esoteric and cumbersome. Of the two, most people find `vi` easier to use, as it is a screen-based editor and therefore closer in character to other editors they may be familiar with.

The first half of this chapter describes `ed`, and the second half describes `vi`. (Quick references to most of the commands available are given, and also included in the reference section *Command summaries*, at the back of this guide.)

A final section contains sources of further information for each editor and lists the other editors that are provided with your system.

Introduction to ed

In an attempt to simplify ed, its description is split up into two distinct halves. The first half describes sufficient ed commands to enable a first time user to get started. The second half describes the more advanced features of the editor. Each half ends with a summary of the commands described in the previous sections.

Creating files using ed

To create a file using ed:

- enter ed
- append text
- save the text
- leave ed.

Each of the above steps will be described in this section and you will be shown how to create an example file. Try creating this file on your own system as you read through the section. The file has some deliberate mistakes in it, which will be used later on in this chapter for you to try out some of the more advanced features of ed.

Alternatively, if you do not trust your typing skills, log in to your system as guest and in the directory `/usr/users/guest/editing-files` you will find the example file used, called `example`.

Entering ed

At the normal command line prompt, type:

```
ed
```

Although nothing appears to happen, you have entered ed. Unlike the shell, ed does not have a prompt, but it is waiting for you to input a command.

ed can also be invoked with an existing file that you wish to edit. At the normal command line prompt, type:

```
ed filename
```

where *filename* is the name of the file that you wish to edit.

Appending text

To create the example file, type:

```
a
```

The `a` is a command to `ed`, telling it to append (or add on) any text that follows.

You can enter as many lines of text as you like. If you make a mistake while typing, use the `` key to erase the text. Press `↵` at the end of each line of text you enter. To create a blank line press `↵` twice.

When you have finished entering the text, add a final line that contains just a full stop. This tells `ed` that you have finished, and it is then ready for another command.

```
# ed
a
'Twas brillif, andsss teh slithy sstoves
Did gyre and gimble in the wabe;
All malmsy were the borogoves,
And teh mome raths sssoutgrabe.

"Beware the Jabberwock, my son!
The jaws that bite, the claws that catch!
Beware teh jubjub bird, and shun
The frumious Bandersnatch!"
.
```

Saving the text

To save the example text to a file, type `w` followed by an appropriate filename and then press `↵`. The text that you typed in will automatically be written to this file. The number of characters written is displayed on the next line.

If you are editing an existing filename already given on the command line, you need not specify the filename when you save the file; `ed` knows which file it is and updates it for you.

Leaving ed

To leave `ed` type `q` and then press `↵`. This will take you out of `ed` and back to your normal command line prompt.

```
# ed
a
'Twas brillif, andsss teh slithy sstoves
Did gyre and gimble in the wabe;
All malmsy were the borogoves,
And teh mome raths sssoutgrabe.

"Beware the Jabberwock, my son!
The jaws that bite, the claws that catch!
Beware teh jubjub bird, and shun
The frumious Bandersnatch!"
w example
274
q
#
█
```

If you try to leave `ed` without having saved the changes made to the file, a '?' will be displayed to warn you. If you forgot to save the file, use the `w`

Commands and error messages

command before trying again to quit. If you do not want to save the file, use the `q` command a second time to leave `ed`.

Before you edit the example file you just made, you ought to know a little about giving commands, and how `ed` tells you about errors.

Because `ed` does not have a prompt it is very easy to forget whether you are giving `ed` commands or entering text. If you are ever giving commands and nothing seems to be happening, try typing a full stop on a line of its own. This makes sure that `ed` is ready for a command, and is a useful get-out.

All commands must be on a line of their own, and you must press ↵ after each one.

If you give `ed` an incorrect command, or it could not do what you asked it to, it tells you by displaying a `'?'`. This is the only error message that `ed` gives, and it is left up to you to work out why `ed` gave the message.

Looking at a file

Load the example file you just saved by entering `ed` and using the `e` command:

```
e
e example
```

The file `example` is copied into a buffer, so that you are not changing the file itself. The number of characters in the buffer appears on the screen, but none of the buffer itself. Before you can do anything else, you need to be able to see the text. The `ed` command you need to use is:

```
p
```

Try all the `p` commands below using the example file.

Selected lines

The `p` command prints one or more lines of the text to your screen. You can tell `ed` which lines to start and stop printing:

```
1, 3p          - prints lines 1 to 3 inclusive
2, 4p          - prints lines 2 to 4 inclusive
```

If you wanted to print all of the example you could type:

```
1, 9p          - prints lines 1 to 9 inclusive
```

All lines

For a long buffer you are unlikely to know the number of the last line, and so `ed` uses the character '\$' as an abbreviation for this. You can print all of the buffer by typing:

1, \$p – prints all of the buffer

Single lines

Of course no abbreviation is needed for the start of the buffer – it is always line number 1. So you can print the first line of the buffer by typing:

1, 1p – prints line 1

This can be abbreviated to:

1p – prints line 1

An even simpler abbreviation is to just type the line number:

1 – prints line 1

\$ – prints the last line

Selected lines - using arithmetic

You can also use constructs such as:

\$-3, \$p – prints the last 4 lines

There are other ways to use the `p` command, but first you need to know about the current line.

The current line

Try the following two commands on the example:

1,4p - prints lines 1 to 4 inclusive

p

```
# ed example
274
1,4p
'Twas brillif, andsss teh slithy sstoves
Did gyre and gimble in the wabe;
All malmsy were the borogoves,
And teh mome raths sssoutgrabe.
p
And teh mome raths sssoutgrabe.
.=
4
█
```

The second command (p) printed the fourth line of the buffer. Because you didn't tell ed what line numbers to use, it printed the current line – the last line you did something to. ed uses the character . (called 'dot') as a symbol for the current line.

You can find out the value of dot with the command:

`. =`

You can use dot in commands just as you use line numbers. Try these on the example; remember that after a print command the value of dot is set to the last line printed:

- `4` – print line 4, making it the current one
- `.-2, .+2p` – print from 2 lines before to 2 lines after the current one (ie lines 2 to 6 inclusive).
- `. =` – print the current line number – now line 6
- `., $p` – print from the current line to the end of the buffer (ie line 6 onwards).

Stepping through the text

These two commands are especially useful:

- `+.1p` – print the next line in the buffer
- `-.1p` – print the previous line in the buffer

You can use them to step forwards or backwards through the buffer, printing a line at a time. They are such useful commands that they both have abbreviations:

- `␣` – print the next line in the buffer
- `-␣` – print the previous line in the buffer

Try these on the example. If `ed` shows a '?' it means you are trying to go past the start or end of the buffer.

Editing the buffer

There are a number of ways you can edit the buffer:

- add new text by appending it before or inserting it after the current line
- remove text by deleting lines
- move lines from one part of the buffer to another
- replace text by substituting one string for another.

Adding text

The simplest way of adding text using `ed` is to use one of the following commands:

- a** – append text after the current line
- i** – insert text before the current line

These commands can also be used with line numbers:

- 3a** – append text after line 3
- 3i** – insert text before line 3

Remember to type a full stop on a line by itself to finish adding the text.

Deleting lines

You can delete any unwanted lines by typing:

- d** – delete the current line, and set dot to the next line

This command can also be used with line numbers:

- 4d** – delete line 4
- 1, .d** – delete line 1 to the current line inclusive
- .+3, \$d** – delete from 3 lines after the current one to the end of the buffer.

Moving lines

You can move lines using:

start line, end line m after this line ↵

For example:

- 5, 5m\$** – move line 5 to after the last line
- 1, 4m\$** – move lines 1 to 4 inclusive to after the last line

Replacing text

The substitute command is one of the most used and important `ed` commands. Its simplest form is:

- s/this/that/** – changes the first occurrence only of *this* to *that* on the current line

So to correct the first mistake in the example you could use these commands:

- 1** – make sure line 1 is the current line
- s/if/ig/** – make the change
- p** – and display it

You can use line numbers with `s`, just like with other commands, so you could instead have typed:

- 1s/if/ig/** – make the change to line 1
- p** – and print it

You can in fact also combine these two commands – a `p` can follow almost any command. (Note that otherwise `ed` does not allow multi-command lines.) This is so useful with the `s` command that most of the examples from now on will use it:

- 1s/if/ig/p** – make the change and print it

If you want to delete text, tell `ed` to change the text to nothing:

- 1s/sss//p** – change `sss` in line 1 to nothing, ie delete it

If you want to change all occurrences in a line rather than just the first one, you need to add a `g` to the end of the command:

- 1s/sss//gp** – change all occurrences of `sss` in line 1 to nothing, ie delete them

Trouble-shooting

There is one very important command that will help you to get out of difficulty or to avoid complete catastrophe!

- u** – undo the last change made

If you delete something from a line or a number of lines or even the whole file and then change your mind, you can recover the previous situation by typing `u`. Typing `u` a second time reverses the effect of the 'undo'.

An exercise

Use what you have learned already to correct and add to the text. Your finished text should read as follows when you use 1,\$p to print it all to the screen:

```
1,$p
'Twas brillig, and the slithy toves
Did gyre and gimble in the wabe;
All mimsy were the borogoves,
And the mome raths outgrabe.

"Beware the Jabberwock, my son!
The jaws that bite, the claws that catch!
Beware the Jubjub bird, and shun
The frumious Bandersnatch!"

He took his vorpal sword in hand:
Long time the manxome foe he sought-
So rested he by the Tumtum tree,
And stood awhile in thought.

- Lewis Carroll
  Through the Looking Glass
■
```

Basic command summary

The following commands are sufficient to get started with ed:

Command	Use
i	insert text before the current line
a	append text after the current line
.	finish adding text
d	delete the current line
p	print the current line
↓	print the next line
↑	print the previous line
m	move lines
s	substitute text on the current line
w filename	write out the changes to <i>filename</i>
q	quit. A second q bypasses checking
u	undo the last change made

Context searches

Sometimes you may know which part of the text you want to edit, but not the line it is on. You can replace line numbers in any command by a context search:

/the/p – prints the first line after the current one that contains 'the'

Just as **2p ↓** could be abbreviated to **2 ↓**, so the above can be abbreviated to:

/the/ – prints the first line after the current one that contains 'the'

If your context search did not find the occurrence of **the** that you were looking for, you'll probably have to repeat the search. It's a nuisance having to type the whole command again, but you can shorten it to:

// – repeat the last context search

Context searches are often most useful with the substitute command. For instance, to correct **malmsy** in our example, you could type:

/malmsy/s/malmsy/mimsy/p
– changes the next **malmsy** to **mimsy**

This means: find `malmsy` starting from the next line, change `malmsy` to `mimsy`, and print the result. Again we can save a bit of typing and not repeat `malmsy`:

```
/malmsy/s//mimsy/p
```

– changes the next `malmsy` to `mimsy`

This means: find `malmsy` starting from the next line, change what you found to `mimsy`, and print the result. Just as above, the `//` refers to the last context search.

Note that we have now seen two different uses for `//` – this command shows them both:

```
/sss/s///p
```

– delete the next `sss`

The first two `//` of the group of three slashes immediately after the last `s` is telling `ed` what to replace, and so refers to `sss`, the last context search; the pair of `//` immediately before the `p` tells `ed` what to replace it with, and so represents nothing. Make sure you understand this difference.

When a context search is specified, `ed` starts looking from the line after the current one to the end of the buffer, then moves up to the start of the buffer and searches up to the current line. This wrap-around effect ensures that the whole of the file is searched through irrespective of where you are in the buffer.

Backward searches

You can change the direction of any context search by using `?` instead of `/`:

```
?malmsy?s??mimsy?p
```

– on the first line it finds that contains `malmsy`, changes the first occurrence of `malmsy` to `mimsy`

In the above example, the whole buffer is searched through, but in the opposite direction.

Making global changes

Sometimes you want to make the same changes or corrections throughout a file. You can do this with the commands you have already learnt:

```
1,$s/ teh / the /p
```

– change the first `teh` in each line to `the`

```
1,$s/ teh / the /gp
```

– `g` specifies to change all `tehs` in each line to `the`

Note the spaces before and after `the`, which ensure that a whole word is found and not just three letters contained within a longer word.

In fact, these commands only print the last line that was changed – the value of dot when the command finished. The `g` modifier is easier to use and can print all the changes it makes. If you put a `g` at the start of a line, the command will be repeated over all of the file. So this command that changes one line:

```
/ teh /s// the /gp
```

– find the next line containing `teh`, and change all `tehs` to `the`

could become this one, which changes all lines:

```
g/ teh /s// the /gp
```

– find all lines containing `teh`, and change all `tehs` to `the`

This is the most commonly used form of the global `g` modifier:

```
g/these/s//those/gp
```

This can be shortened to:

```
gs/these/those/gp
```

You can also use the `g` modifier with other commands such as `d` (delete). For example:

```
g/something/d
```

– delete all lines containing *something*

Special characters

If you've been trying the above commands you may have found that they don't always work as you expect. This is because some characters have a special meaning to `ed` when you are telling it what to search for. They are:

```
^ $ . [ * & \
```

This section provides a summary of the special characters.

The character `^`

The character `^` stands for the start of a line, so:

```
/^start of a line/
```

finds:

```
start of a line...
```

but not:

```
This is the start of a line.
```

Likewise:

```
s/^hello/Hello/p
```

– changes `hello` to `Hello`, if it is at the start of a line

The character `$`

Similarly, the character `$` stands for the end of a line, so:

```
/end of a line$/
```

finds:

```
look for the end of a line
```

but not:

```
The end of a line is ...
```

Likewise:

```
s/hello$/Hello/p
```

– change hello to Hello if it is at the end of a line

The character .

The character '.' stands for any character, so:

```
/m.h/
```

finds:

```
mjh, m3h, m%h or mAh
```

but not:

```
mh, mash, m32h or m#*%h
```

The character [

The character '[' is used for a similar purpose. It introduces a **character class** which matches any single character in the class. The class is ended by the character ']':

```
/[0123456789]/
```

finds the next digit.

You can shorten a character class if the characters follow each other in the alphabet, so:

```
/[0-9]/
```

also finds the next digit.

```
s/[a-z]!/gp
```

changes all lower case letters on the current line to !

The character *

The character '*' means 'the greatest possible number of the previous character'. This includes zero, something that you could find confusing:

```
/mj*h/
```

finds:

```
mh, mjh, mjjh, mjjjh, mjjjjh, etc.
```

If instead you want there to be at least one of the character, you can tell ed to find the character, then to find any number more:

```
/mjj*b/
```

This no longer finds mh, but still finds mjh, mjjh, mjjjh, mjjjjh, etc.

The character &

The character '&' is used on the right side of an s (substitute) command to stand for the text that was matched:

```
s/./(&)/p
```

finds all of a line (. * is any number of any character) and puts brackets around it.

It can be used more than once:

```
s/on and/& & & on.../p
```

changes on and to on and on and on and on...

The character \

Finally, the '\ ' character can be used to remove the special meaning from a character:

```
s/\^\. \&/caret dot ampersand/p
```

changes ^. & to caret dot ampersand.

Remember that the backslash itself is a special character:

```
s/\\/#/p
```

changes \ to #.

You can usefully combine the special characters:

```
g/^[0-9]*/s///gp
```

– delete all digits from the start of every line

```
g/^\$/d
```

– delete all blank lines

ed command summary

The following list of `ed` commands have been covered in this chapter. This list is also included in the reference section, *Command summaries*, at the back of this guide.

Command	Use
i	insert text before the current line
a	append text after the current line
.	finish adding text
d	delete the current line
p	print the current line
↓	print the next line
-	print the previous line
m	move lines
s	substitute text on the current line
/pattern/	search forwards for <i>pattern</i>
?pattern?	search backwards for <i>pattern</i>
//	repeat a context search in the forward direction
??	repeat a context search in the backward direction
g	global modifier – make a command affect all lines
e filename	edit <i>filename</i>
w filename	write out the changes to <i>filename</i>
q	quit. A second <code>q</code> bypasses checking
u	undo the last change made

Introduction to vi

In an attempt to simplify `vi`, its description is split up into two distinct halves. The first half describes sufficient `vi` commands to enable a first time user to get started. The second half is devoted to describing the more advanced features of the editor. Each half ends with a summary of the commands described in the previous sections.

Creating files using vi

To create a file using `vi`:

- Enter `vi`
- insert text
- save the text
- leave `vi`.

Each of the above steps will be described in this section and you will be shown how to create an example file. Try creating this file on your own system as you read through the section – the file created will be useful later on in this chapter for you to try out some of the more advanced features of `vi`.

Alternatively, if you do not trust your typing skills, log in to your system as `guest` and in the directory `/usr/users/guest/editing-files` you will find the example file used, called `quotes`.

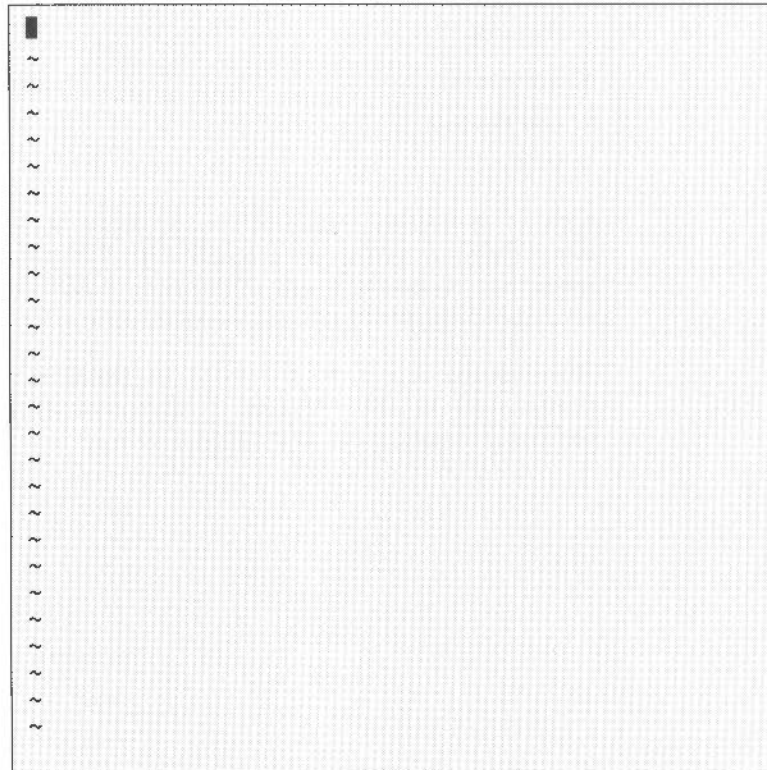
Entering vi

At the normal command line prompt, type:

```
vi
```

After a short delay, `vi` erases the current screen display and creates its own interactive screen. The cursor is on the top line and marks the current position.

The '~' signs represent non-existent lines.



`vi` can also be invoked with an existing text file that you wish to edit. At the normal command line prompt, type:

```
vi filename
```

where *filename* is the name of the file that you wish to edit.

Saving the text

to remember if you ever make a mistake or get confused as to which mode you are in.

To save text to a file, with `vi` in command mode, type a colon (`:`) – this puts `vi` into **colon mode**. Type `w <SPACE>` followed by an appropriate filename and then press `↵`.

The text on your screen will be automatically written to this file. The filename you have specified will appear at the bottom of the screen along with information about the size of the file.

For example, in the diagram below, a new file called `quotes` has been created; it is 9 lines long and contains 269 ASCII characters.

```
"Oh! Pardon me, thou bleeding piece of earth,  
That I am meek and gentle with these grocers"
```

```
"There is a tilde in the affairs of men  
Which, taken at flood, leads on to fortune;"
```

```
"Cowards die many times before their deaths;  
The valiant never taste of garlic but once"
```

```
█
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
"quotes" [New file] 9 lines, 269 characters
```

Leaving vi

To leave vi, type `:` to enter colon mode followed by `q` for **quit**. This will take you out of vi and back to your normal command line prompt.

If you try to leave vi without having saved the changes made to the file, you will receive a warning message.

As indicated by the message below, to leave vi without saving the file, type `q!` (abbreviation for `quit!`). If you forgot to save the file, type `:` to return to command mode, save the file by typing `w` followed by `↵` and then leave vi.

```
"Oh! Pardon me, thou bleeding piece of earth,  
That I am meek and gentle with these grocers"
```

```
"There is a tilde in the affairs of men  
Which, taken at flood, leads on to fortune;"
```

```
"Cowards die many times before their deaths;  
The valiant never taste of garlic but once"
```

```
█
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
No write since last change (:quit! overrides)
```

Moving around within a file

To edit a file of text, you need to be able to locate a particular position in the text and then make some change. `vi` has several commands that enable you to move throughout a file – from character to character, word to word, line to line and by screenfuls of text.

Character to character

`vi` has four commands, one for each direction, that enables you to move to any single character in a file:

h – move left or *west* by one character on the screen

j – move downwards or *south* by one line on the screen

k – move upwards or *north* by one line on the screen

l – move right or *east* by one character on the screen.

The keys are grouped together on the keyboard to facilitate their use for moving the cursor around, although they are hopelessly non-mnemonic!

Two more useful commands for moving from character to character are:

\$ – move to the last character on the current line

0 – (zero) move to the first character on the current line.

Word to word

A quicker way to move along a line than by character to character, is by jumping from word to word. There are various commands available to do this:

w – move forward to the beginning of the next word

b – move back to the beginning of the previous word

e – move forward to the end of the current word.

`w`, `b`, and `e` will stop at punctuation in the text, so they may not always take you to your desired location. The capital letter versions `W`, `B` and `E` treat any non-blank character as part of the word and so override punctuation.

All of the above commands may be prefixed with a number to move any number of words at a time. For example, `5w` moves forwards five words from the current position.

Line to line

`vi` also has commands to take you to specific lines on the screen:

H – move to the **H**ome or top line on the screen

M – move to the **M**iddle line on the screen

L – move to the **L**ast line on the screen.

`H` and `L` can also be prefixed with a number to give you a greater range of labelled lines. For example, `3H` will take you to the third line on the screen. `3L` will take you to the third line from the bottom of the screen.

A count with the `M` command is meaningless and has the same effect as `M` by itself.

Screenfuls

The basic idea of `vi` is that what you see on the screen is part of a buffer that contains the file to be edited. The screen acts as a window to this buffer. Therefore, you need a set of scrolling and paging commands that give you window control and allow you to move through the buffer when you are editing large files that cannot be contained on one screen.

The scrolling commands are:

`<CTRL-D>` – scroll **D**own by half a screenful

`<CTRL-U>` – scroll **U**p by half a screenful.

The paging commands are:

`<CTRL-F>` – page **F**orward by a screenful

`<CTRL-B>` – page **B**ackward by a screenful.

Thankfully these commands are more pertinent to the functions they perform and therefore easier to remember. Scrolling is usually preferable to paging since it is a slightly smoother movement and leaves more context from the previous window.

When the end of the buffer is reached, `vi` will display non-existent lines as a `'~'`. Two more useful commands for window control are:

`<CTRL-E>` – to **E**xpose one more line at the bottom of the screen, ie scroll down one line.

<CTRL-Y> - to Yank another line onto the top of the screen, ie scroll up one line.

Editing a file

There are a number of ways you can edit a file:

- add new text by appending it after or inserting it before the current cursor position or the current line
- replace text by substituting one string for another
- remove text by deleting all or part of an object
- move text from one part of a file to another.

Adding text

The simplest way of adding text using `vi` is to use one of the following commands:

- a** - append text after the current cursor position
- i** - insert text before the current cursor position
- o** - open a gap in the file to append text after the current line.

These commands can be capitalised to give them slightly different meanings:

- A** - Append text at the end of the current line
- I** - Insert text at the beginning of the current line
- O** - Open a gap in the file to insert text before the current line.

All of the above commands are accessed with `vi` in command mode; while using them you are in insert mode. For each command you exit from insert mode by pressing <ESC>, when you have finished making the changes.

The example below shows how the `o` command is used, starting with `vi` in command mode and the cursor positioned at the top of the file.

```
"Oh! Pardon me, thou bleeding piece of earth,  
That I am meek and gentle with these grocers"  
- Antony; Julius Caesar III-█
```

```
"There is a tilde in the affairs of men  
Which, taken at flood, leads on to fortune;"
```

```
"Cowards die many times before their deaths;  
The valiant never taste of garlic but once"
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
:"quotes" [New file] 9 lines, 269 characters
```

The sequence of instructions used were:

- move to the end of the line ending `grocers"` by typing `$`
- type `o` followed by the text `- Antony; Julius Caesar III-i`
- press `<ESC>` to exit from insert mode
- type `:` to put `vi` into colon mode, followed by `w ↵` to save the changes.

Replacing text

Use the same sequence of commands to add the speaker and title of the play to the other two quotes:

- - Brutus; Julius Caesar IV-iii
- - Caesar; Julius Caesar II-ii

There are three basic commands that allow you to change or replace text in a file:

s – to substitute a string of characters for the character under the cursor

cw – to change a word

cc – to change an entire line.

Just as with the *adding text* commands, you access these commands with `vi` in command mode; while using them you are in insert mode. For each command you exit from insert mode by pressing `<ESC>` when you have finished making the changes.

To substitute a string of characters for the character under the cursor, position the cursor on the character you wish to replace and type **s**. `vi` will go into insert mode and identify the letter up to which the text will be changed, by replacing it with a `$` sign.

The `cw` command works in the same way, except the substitution symbol appears on the last character of the word you are changing. The word is changed from the cursor position onwards, so if you issue the `cw` command in the middle of a word the first part of the word remains unchanged. For example:

```
"Oh! Pardon me, thou bleeding piece of earth,
That I am meek and gentle with these bu■cer$"
- Antony; Julius Caesar III-i

"There is a tilde in the affairs of men
Which, taken at the flood, leads on to fortune;"
- Brutus; Julius Caesar IV-iii

"Cowards die many times before their deaths;
The valiant never taste of garlic but once"
- Caesar; Julius Caesar II-ii

~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~

:"quotes" 12 lines, 364 characters
```

The `cc` command is similar to `s` and `cw`. To use the command, position the cursor anywhere on the line you wish to alter and type `cc`. The line will disappear and you can then type the new line, terminated by `<ESC>`.

Deleting text

Use the replace commands just discussed to change:

- `tilde` to `tide` in the second quote
- `garlic` to `death` in the third quote.

Text may be deleted using various commands. In contrast to adding and replacing text commands, text is deleted with `vi` in command mode.

The basic commands used are:

- `x`** – to delete the current character
- `dw`** – to delete the current word
- `dd`** – to delete the current line.

To use the commands, just position the cursor over the letter, word or line that you wish to delete and type the appropriate command.

All the above commands can be prefixed with a number to give you a repetition count. For example, `4x` will delete four characters, starting with the current character.

Moving text

Text may be moved from one place to another using a delete command followed by the `p` command:

- `p`** – puts back the deleted text after the current cursor position or current line, if you deleted a whole line.
- `P`** – Puts back the deleted text before the current cursor position or current line, if you deleted a whole line.

Again, this command is used with `vi` in command mode. The delete command saves the deleted text in a buffer and allows you to take as many copies of it as you like.

The example below shows how to use the `p` command.

```
"Cowards die many times before their deaths;
The valiant never taste of death but once"
- Caesar; Julius Caesar II-ii

"Oh! Pardon me, thou bleeding piece of earth,
That I am meek and gentle with these butchers"
- Antony; Julius Caesar III-i

"There is a tide in the affairs of men
Which, taken at the flood, leads on to fortune;"
- Brutus; Julius Caesar IV-iii

~
~
~
~
~
~
~
~
~
~
~
~

:"quotes" 12 lines, 364 characters
```

The sequence of instructions used to make the last quote the first quote, were:

- with `vi` in command mode, move the cursor to the beginning of the line commencing, "Cowards die ...
- type `3dd` and then `H` to return the cursor to the top line
- type `P`. The three lines you just deleted should appear at the top of the file.

Trouble-shooting

Instead of deleting and moving the text, you can make a copy of the text and move it. The `Y` command will *yank* (copy) the current line into a buffer. You can then use the `p` command in the same way as before to copy the text to another part of the file. For more information regarding the use of buffers in `vi` see the later section entitled *Editing using multiple buffers*.

There are two very important commands that will help you to get out of difficulty or to avoid complete catastrophe!

- `u` – undo the last change made

- `U` – Undo the last set of changes made to the current line.

Both commands are issued with `vi` in command mode. So if you delete something from a line or a number of lines or even the whole file and then change your mind, you can recover the previous situation by typing `u`. Typing `u` twice in a row undoes the last `u`.

The `u` command lets you reverse only a single change. However, if you have really made a mess of editing the current line after a number of commands, and have **not** moved the cursor off the line, you can recover the line in its original state by using the `U` command.

Basic command summary

The following commands are sufficient to get started with `vi`:

Command	Use
h	move left by one character on the screen
j	move down by one line on the screen
k	move up by one line on the screen
l	move right by one character on the screen
<CTRL-D>	scroll down by half a screenful
<CTRL-U>	scroll up by half a screenful
<CTRL-F>	page forward by a screenful
<CTRL-B>	page backward by a screenful
i	insert text before the current cursor position
o	open file to append text after the current line
x	delete the current character
dd	delete the current line
Y	yank lines into a buffer
P	put back the deleted text after the current cursor position
:w filename	write out the changes to <i>filename</i>
:q	quit. :q! bypasses checking
u	undo the last change made

More advanced features

Locating high-level objects

The requests described so far can be learned with just a few sessions on your system. The second half of this chapter introduces a few more advanced editing concepts plus some short cuts for the more experienced user.

The positioning commands described earlier, can be used to locate any character in a file. However, `vi` has extra commands that allow you to locate more complex, high-level objects such as a sentence or paragraph. For example:

-) – move the cursor to the end of the current sentence
- (– move the cursor to the start of the current sentence
- } – move the cursor to the end of the current paragraph
- { – move the cursor to the start of the current paragraph.

A sentence is defined as an object that ends with a dot (.), exclamation mark (!) or question mark (?) followed by newline or two spaces. A paragraph is defined as beginning and ending with a blank line.

The commands are used with `vi` in command mode and can also be prefixed with a number to instruct the cursor to move by that number of sentences or paragraphs.

Location by context

It is also possible to search for a particular word or string of text by specifying a string pattern. With `vi` in command mode, type:

/

The cursor will jump to the bottom of the screen. You can then type in the string pattern you are searching for, followed by `↵`. Following this, the cursor will move forwards through the file searching for the specified string.

There is also a command to instruct `vi` to search backwards for a string pattern. Again, with `vi` in command mode, type:

?

The cursor will jump to the bottom of the screen. You can then type in the string pattern you are searching for followed by `↵`. The cursor will then move backwards through the file searching for the specified string pattern.

When a context search is specified, `vi` starts looking from the line after the current one to the end of the file, then moves up to the start of the file and searches up to the current line. This wrap-around effect ensures that the whole of the file is searched through irrespective of where you are in the file.

In both cases, if the string cannot be found, an error message will be issued to state that no such string pattern can be found.

Once a string pattern is found, the command `n` can be used to repeat the context search in either direction:

n – repeats a context search in the same direction

N – repeats a context search in the opposite direction.

The above commands can be used continuously to find all occurrences of the string pattern in the file.

Repeating the last command

The last text change that you entered, with `vi` in command mode, can be repeated by pressing `.'`.

This is very handy when used with a search command. For example, suppose you are editing a file that contains references to 'great Britain' instead of 'Great Britain' but also contains instances of the word 'great'.

One way you could correct this problem is search for 'great'. The first time you found 'great' in great Britain, you would change the 'g' to 'G' using `~`. The next occurrence you found, you could simply press the `.'` key and `vi` would repeat the last editing command you made, ie change the 'g' to 'G'. Whenever you find 'great' on its own, just type `n` to search for the next instance of the word.

Joining two lines

Any two continuous lines in a file can be joined together using the `J` command. To use the command, with `vi` in command mode, place the cursor anywhere on the current line and press `J`. The line below the current line will be joined on to the end of the current line.

Notice that `vi` automatically places a space between the last word on the first line and the first word on the second line.

Splitting a line

To split a line, put `vi` into insert mode at the point in the line where you want to split the text and press `↵`. The remaining text will jump to a new line.

Editing using multiple buffers

In earlier sections you were introduced to the concept of using a buffer in `vi` to move or yank text from one part of a file to another. This buffer is a hidden buffer that `vi` uses to preserve the last text object deleted, changed or yanked – and you cannot reference it directly. However, `vi` does have two sets of buffers that can be referenced:

Nine numerical buffers (1-9) are used to store the last nine deleted blocks of text. However, note that in order to be stored in this way, the block must be at least one complete line (so that when you delete it, the file has fewer lines of text). When a new block is deleted, it is stored in buffer 1 after the contents of each existing buffer are moved to the next highest numbered buffer. Any text previously stored in buffer 9 is discarded.

26 alphabetical buffers (a-z) can be referenced directly in a delete or yank command. Unlike the numerical buffers, the contents of the alphabetical buffers are not lost when you change files, so they are ideal for moving text from one file to another.

Numerical buffers

With `vi` in command mode, the contents of buffer 1 (ie the last bit of text that you deleted) can be examined by the command:

```
"1P
```

The `"` notifies `vi` that you are about to quote the name of a buffer and `P` places its contents before the current line. The remaining buffers can be examined in a similar way. When searching through these buffers looking for a particular section of deleted text, be careful to use `u` (the undo command) rather than just re-deleting the unwanted text as this will then be put in a new buffer and you may lose the previous deletion you were looking for.

Alphabetical buffers

The command `Y` which you met in an earlier section, will yank the current line into the unnamed buffer. However, if preceded by a buffer name, for example `t`, the command `"tY` will place the current line into buffer `t`.

You can subsequently refer to this buffer and include it anywhere in the file using the command `"tP` to put the text after the cursor, or `"tP` to put the text above the cursor. If you have deleted a whole line, then the command `"tP` puts the text after the current line and `"tP` puts the text before the current line.

Remember, you can also give *Y* a count of lines to yank and thus duplicate several lines. These lines can then be transferred to another part of the same file.

Alternatively, you can read in a new file and transfer the contents of the buffer to this file. With *vi* in command mode, type:

```
:e filename
```

The new file called *filename* will be read in, yet the contents of the alphabetic buffers will be saved. So you can use the *p* or *P* commands to place text in this new file.

If you wish to append text to these buffers without destroying their current contents, you can do so using the same command format as previously described but instead capitalising the alphabetic name of the buffers, *A-Z*. For example, to add two more lines to buffer *t* with *vi* in command mode, type:

```
"T2Y
```

The two lines following the current cursor position are added to buffer *t*.

The line editor

Just like *ed*, *vi* also has some line editing capabilities. The line editor associated with *vi* is called *ex*. The commands that *ex* provides are very similar to those used by *ed*, and you will find that this helps you if you wish to learn both *vi* and *ed*.

ex is invoked from within *vi* by typing a colon, ie putting *vi* into colon mode. A few commands have already been described with *vi* in colon mode so you should already be familiar with some *ex* commands. For example, *w* to save a file, *q* to quit *vi* etc.

There are many other commands in the *ex* editor that can be called from *vi*. Some of the commands will be discussed here but for a complete description of *ex*, refer to the *Ex Reference Manual – Version 3.7* by William Joy and Mark Horton in the *Berkeley 4.3 UNIX User's Supplementary Documents*.

Making global changes

One of the most powerful commands in *ex* is the global command. It is also one of the most dangerous, so it is advisable to practise with this command on a test file before doing any serious editing with it.

The command format for making global changes from `ex` is:

```
g/pattern/command-list
```

A simple example should explain the fundamentals of this command.

Suppose you have typed in several pages of text about Shakespearian tragedies but have misspelt 'tragedies' as 'tragedys':

- The first action is to scan the file to find every line that matches the pattern. Then `command-list` is executed for each such line. Thus the command,

```
:g/tragedys
```

instructs `ex` to do a global search for the first instance of the characters 'tragedys' on each line in the file.

- The command-list is as follows:

```
:s/tragedys/tragedies/
```

This is the substitute command that will change the first occurrence of 'tragedys' in the current line with 'tragedies'. To change every occurrence of 'tragedys', add a `g` at the end:

```
:s/tragedys/tragedies/g ↵
```

- The complete command to substitute 'tragedies' for every instance of 'tragedys' is therefore:

```
:g/tragedys/s/tragedys/tragedies/g
```

This can be abbreviated to:

```
:g/tragedys/s//tragedies/g
```

as the substitute command remembers the word 'tragedys' after the global command `g`.

Executing shell commands from within vi

There are two ways you can execute a shell command from within `vi`.

- You can execute one command only, with `vi` in command mode by typing:

```
:!cmd
```

The shell command `cmd` will be executed and then you will be instructed to press `↵` to go back to `vi`.

- Alternatively, you can temporarily start a new shell, perform some shell commands and then return to `vi`. Type:

```
:sh
```

You will receive a normal shell prompt, from which you can perform some shell commands (you could even begin to edit another file in `vi`).

After you have executed the shell commands, press:

```
<CTRL-D>
```

or whatever means you use to log out.

You will then return to the exact line and file you were editing before you left `vi`.

Editing multiple files

In `vi` it is possible to edit a number of files one after the other. At the normal command line prompt, type:

```
vi file1 file2
```

`file1` will be displayed for editing.

When this file has been edited to your satisfaction, type:

```
:w
```

followed by,

```
:n
```

From the commands above, the changes to `file1` are saved and the next file in the list (`file2`) is read into the buffer.

Setting options

If the first file is not saved, an error message is printed and no action is taken. `:n!` can be used to override this and move on to the next file in the list to be edited.

Also, if you try to quit `vi` without having edited all the files in the list, `vi` will display a warning message and refuse to quit. However, if you issue the command a second time, `vi` will be persuaded that you are serious and allow you to exit.

`vi` has a number of options for controlling its behaviour. Some of the options have a value; others are either *on* or *off*.

A list of the most useful options available, together with their abbreviations and default values, is given in the following table:

Name	Default	Description
<code>autoindent(ai)</code>	<code>noai</code>	Supply program indentation automatically.
<code>autowrite(aw)</code>	<code>noaw</code>	Automatic write when <code>:n</code> or <code>!</code> is issued
<code>ignorecase(ic)</code>	<code>noic</code>	Ignore upper/lower case when searching
<code>list</code>	<code>nolist</code>	Print tabs as <code>^I</code> ; end of lines marked with <code>\$</code> .
<code>number(nu)</code>	<code>nonu</code>	Display lines prefixed with line numbers.

To list the options which are currently set, with `vi` in command mode, type:

```
:set all
```

Certain options are set initially by default but can be changed to suit your own needs. With `vi` in command mode an option value can be altered by a command of the form:

```
:set optionname=value
```

Other options may be turned *on* or *off* respectively by commands of the form:

```
:set optionname
```

```
:set nooptionname
```


At any time during an editing session you can get a list of all the options which you have changed. With `vi` in command mode type:

```
:set
```

`set` can be abbreviated to `se` and multiple options can be placed on one line. For example:

```
:se ai aw nu
```

Options set by the `set` command will only last for as long as you are in the editor. To have some options set up whenever you use the editor, use the shell variable `EXINIT`.

For example, suppose you want to give shiftwidth (`sw`) a value of 15 and set the autoindent (`ai`) option so that you get an automatic indent for your programs, ie:

```
se sw=15 ai
```

How you do this depends on the shell you are using.

If you are using the Bourne shell, put the following lines in the file `.profile` in your home directory:

```
EXINIT='se sw=15 ai'  
  
export EXINIT
```

If you are using the C shell, put the following line in the file `.login` in your home directory:

```
setenv EXINIT 'se sw=15 ai'
```

After you reread your `.profile` or `.login` file by logging out and logging in again, these commands will then be run every time you start up `vi`.

Macro commands

`vi` can also map a common sequence of commands to an unused character on the keyboard. For example:

```
:map K eas <CTRL-V> <ESC>
```

defines `K` to be equivalent to `eas` followed by `<ESC>`, where `<CTRL-V>` escapes the `<ESC>` character when typing in the above mapping. Otherwise, the first `<ESC>` would return `vi` to command mode, rather than become part of the map definition.

This means that if the cursor is at the beginning of a word and you type `K`, the word will have an `s` added to it.

The following characters have no associated meaning in `vi` and can therefore be used as macro commands:

```
K V g q v *
```

Macro commands can also be set in the shell variable `EXINIT` and grouped together along with the options described in the previous section. For example, if you are using the Bourne shell you would put the following in your `.profile` file:

```
EXINIT = 'se sw=15 ai | map K eas <CTRL-V> <ESC>'  
export EXINIT
```

and if you are using the C shell you would put the following in your `.login` file:

```
setenv EXINIT 'se sw=15 ai | map K eas <CTRL-V> <ESC>'
```

Similarly to setting options, after you reread your `.profile` or `.login` file by logging out and logging in again, these commands will then be run every time you start up `vi`.

For more information about login shells, see the chapter *Using the UNIX shell*.

vi command summary

The following list of `vi` commands have been covered in this chapter. This list is also included in the reference section *Command summaries*, at the back of this guide.

Command	Use
h	move left by one character on the screen
j	move down by one line on the screen
k	move up by one line on the screen
l	move right by one character on the screen
\$	move to the last character on the current line
0	move to the first character on the current line
w	move forward to the beginning of the next word
b	move back to the beginning of the previous word
e	move forward to the end of the current word
W	move forward to the beginning of the next word, ignoring punctuation
B	move back to the beginning of the previous word, ignoring punctuation
E	move forward to the end of the current word, ignoring punctuation
H	move to the home or top line on the screen
M	move to the middle line on the screen
L	move to the last line on the screen
<CTRL-D>	scroll down by half a screenful
<CTRL-U>	scroll up by half a screenful
<CTRL-F>	page forward by a screenful
<CTRL-B>	page backward by a screenful
<CTRL-E>	expose one more line at the bottom of the screen
<CTRL-Y>	yank another line onto the top of the screen
)	move the cursor to the end of the current sentence
(move the cursor to the start of the current sentence
}	move the cursor to the end of the current paragraph
{	move the cursor to the start of the current paragraph
/pattern	search forwards for <i>pattern</i>
?pattern	search backwards for <i>pattern</i>
n	repeat a context search in the same direction
N	repeat a context search in the opposite direction

a	append text after the current cursor position
i	insert text before the current cursor position
o	open a file to append text after the current line
A	append text at the end of the current line
O	open a file to insert text before the current line
s	substitute a string of characters for the cursor character
cw	change a word
cc	change an entire line
x	delete the current character
dd	delete the current line
dw	delete the current word
Y	yank lines into a buffer
p	put back the deleted text after the current cursor position or current line
P	put back the deleted text before the current cursor position or current line
u	undo the last change made
U	undo the last set of changes made to the current line
.	repeat the last buffer change command
J	join together the current line with the line below

ex command summary

The following list of `ex` commands have been covered in this chapter. This list is also included in the reference section *Command summaries*, at the back of this guide.

Command	Use
<code>:w filename</code>	write out the changes to <code>filename</code>
<code>:q</code>	quit. <code>:q!</code> bypasses checking
<code>:e</code>	edit a new file. <code>:e!</code> bypasses checking
<code>:g</code>	globally search for a string
<code>:s</code>	substitute one string for another
<code>!:cmd</code>	execute the shell command, <code>cmd</code> and return to <code>vi</code>
<code>:sh</code>	execute a shell
<code>:n</code>	edit the next file in an argument list. <code>:n!</code> bypasses checking
<code>:set</code>	print or set options
<code>:map</code>	define a macro command

Sources of further information

- *A Tutorial Introduction to the UNIX Text Editor* by Brian W. Kernighan in the *Berkeley 4.3 UNIX User's Supplementary Documents*.
- *Advanced editing on UNIX* by Brian W. Kernighan in the *Berkeley 4.3 UNIX User's Supplementary Documents*.
- *An Introduction to Display Editing with vi* by William Joy & Mark Horton in the *Berkeley 4.3 UNIX User's Supplementary Documents*.
- *The Ex Reference Manual – Version 3.7* by William Joy & Mark Horton in the *Berkeley 4.3 UNIX User's Supplementary Documents*.
- `ed(1)`, `vi(1)` and `ex(1)` manual pages.

Possible error messages that you may receive while using `ed` and `vi` are outlined in the reference section *Trouble-shooting*, at the back of this guide.

Other editors provided

The following editors may also be available on your system:

- `edit(1)`, a simplified line-based editor based on `ex` – for more information, refer to *Edit: A Tutorial* by Ricki Blau and James Joyce in the *Berkeley 4.3 UNIX User's Supplementary Documents*.

The utility `learn` has a useful system tutorial on `edit` (the course called `editor`).

- `uemacs` (short for `micro-emacs`), a cut-down version of `emacs`, the screen-based extensible text editor. `uemacs` lives in `/usr/new/uemacs` and has a useful online help facility.
- `xedit`, the screen-based interactive editor that runs from the X Window System. For information on how to start using `xedit`, refer to the chapter *Using the X Window System* later on in this guide and also to the manual page for `xedit(1)`.

Networking and NFS

Introduction

This chapter is split into two sections. The first section defines some of the terms and concepts of networking. The second section describes the networking software that is available on your workstation.

What is a network?

A **network** is a group of computers connected together so that they can transmit information between each other. For example, there is a whole host of networking commands you can use to transfer files and send messages to other users on the network.

Networks also allow you to share resources. For example, a printer may be connected to a workstation on your network. If you wish to use this printer from your **local** workstation, you can use the network to send your file to this printer via the **remote** workstation that is connected to it.

Note that **local** refers to the workstation you initially log in to and **remote** refers to other workstations and machines on the network.

Types of network

There are many different types of networks. The two types discussed in this chapter are:

- **Local Area Network (LAN)**
- **Wide Area Networks (WAN)**

The characteristics of these networks are discussed in the next few sections.

Local area network

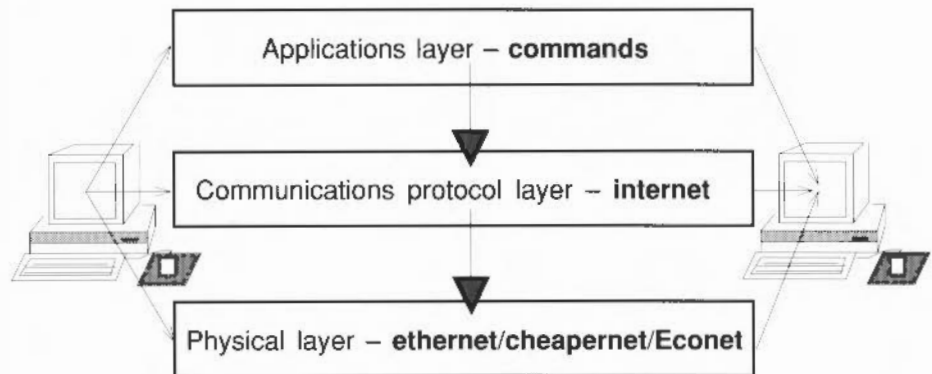
A local area network usually refers to a number of workstations, mainframes and peripherals connected together within a single premises. Your workstation may be connected to other workstations by an **ethernet** cable or **Econet** cable.

Ethernet is the physical cable that attaches to the ethernet port of your workstation and provides a high speed communication link between all the other machines connected on your network. There is also a thin wire version of ethernet, called **cheapernet**.

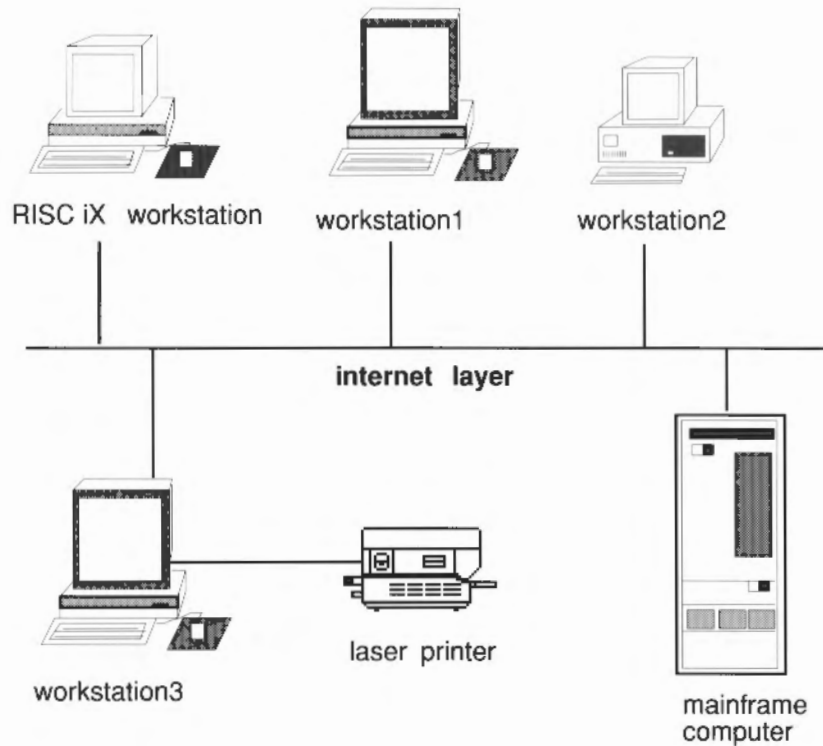
Econet is another similar cable that attaches to the econet port of your workstation and provides a similar communications link to an ethernet cable.

Whichever cable is used to connect two machines, they also need to use the same 'language' to talk to each other, so that you can use commands to copy files etc. In networking this is referred to as a communications protocol. The protocols supported by ethernet and Econet on your RISC iX workstation are the **internet** communications protocols.

This idea of increasingly sophisticated levels of communication can be represented as a series of layers. For example:



A typical local area network, could be shown as:

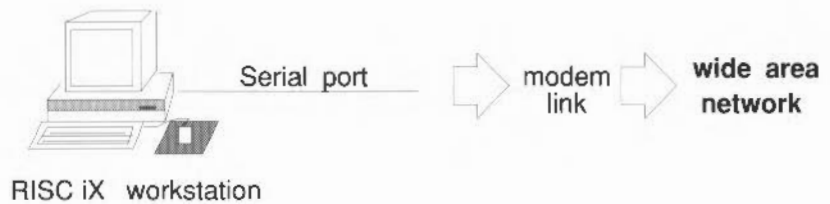


The machines communicate using the internet protocols layer over a combination of ethernet, cheapernet and Econet cables.

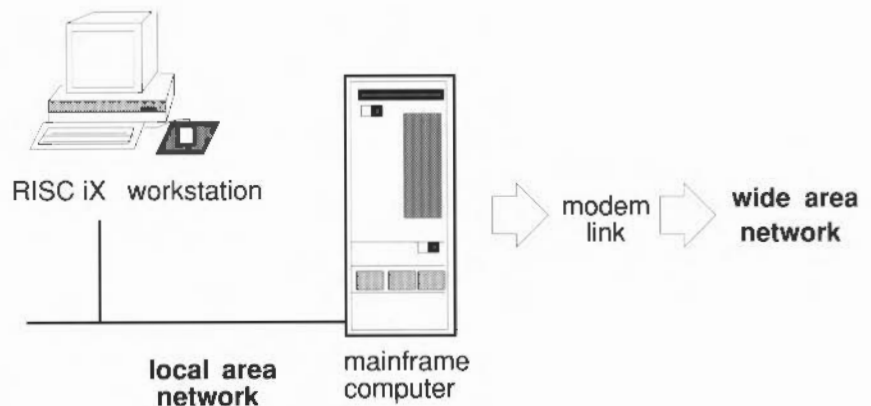
With a set-up like this, you can use the networking commands on your workstation to communicate with, and access information on, every other piece of hardware on the network.

Wide area network

As well as being connected to a local area network, your workstation can also be connected to other machines over longer distances via a **Wide Area Network**. (WAN). In this case, the connection is normally made from the serial port of your workstation to a modem and then to the outside world via telephone cables. For example:



Alternatively, you could have access to both types of network by being connected locally, via ethernet, to a large mainframe computer which is also connected to a modem:



In this example, the mainframe acts as a **gateway** between the two networks, by being able to transfer the information from one type of network and communication line to another type of network and communication line.

As with a local area network, the physical link must be supported by an appropriate communications protocol for the machines to be able to talk to each other. Due to the variety of machines that use these links, there is a wide variety of protocols.

To circumvent this problem a series of programs exists that support a range of communications protocols. One of the most popular of these programs is **UUCP** (*UNIX to UNIX copy program*) which permits communication between all types of UNIX systems, typically over serial lines.

It is likely that your System Administrator has already set up UUCP on your workstation to enable you to communicate over wide area networks and local area networks as well. For more information about UUCP and the other programs available, refer to the next chapter *Communicating with other systems and users*.

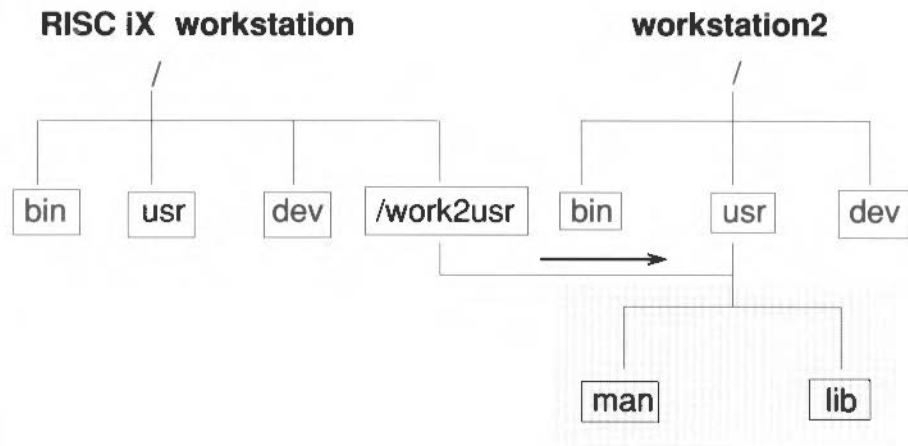
The second part of this chapter describes how to use the software available on your workstation for networking in a local area network environment. It assumes that your workstation has been installed on a local area network. If it has not, skip the rest of this chapter and move on to the next chapter *Communicating with other systems and users*.

NFS – the Network File System

The industry standard Sun Network File System (NFS) is available on your workstation and uses internet protocols along with some other advanced communications protocols to provide a facility for sharing files in a local area network.

It allows you to perform many operations over the network without you even realising you are doing so. For example, you can access a file on a remote workstation and treat it as if it were on your local workstation.

This is accomplished by mounting a remote file system, or just a particular directory of the file system, then reading or writing files in place. In the diagram at the start of this chapter you could, for example, access a file on *workstation2* from your workstation, provided the directory where the file resides has been mounted on your workstation.

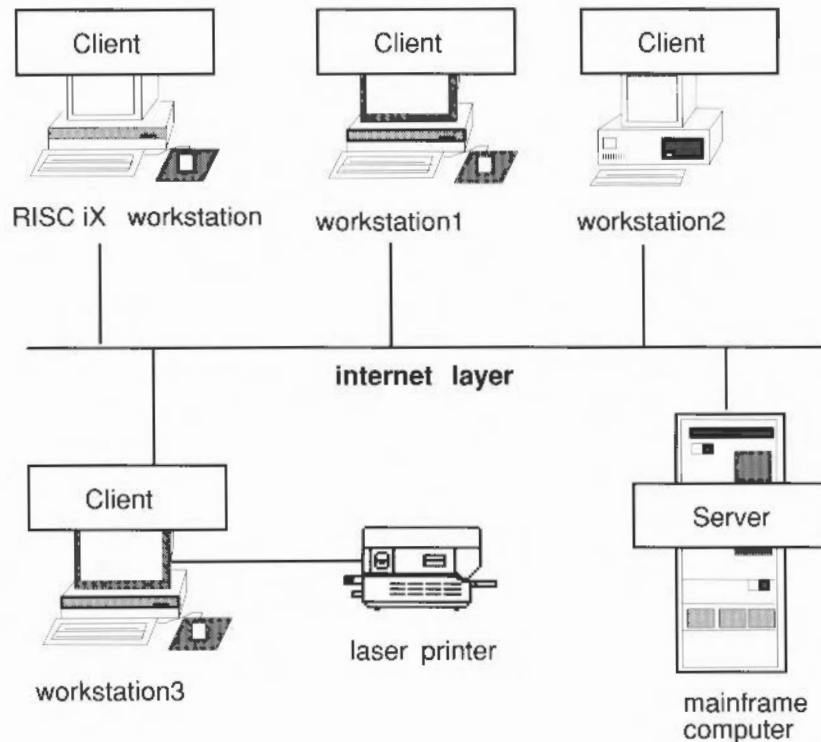


In the above diagram, if the directory `/usr` on *workstation2* has been mounted on your workstation as `/work2usr`, then the shaded directory tree for `/usr/man` and `/usr/lib` on *workstation2* would appear below `/work2usr/man` and `/work2usr/lib` respectively, on your workstation. The shaded area 'belongs' to *workstation2* but appears to be part of your filing system.

You can then refer to the contents of this directory as you would any normal directory on your workstation. The fact that the information resides on the hard disc of another workstation on the network is of no consequence. To you as a user, there seems to be no difference between reading and writing a file contained on your own hard disc and reading or writing a file on a disc connected to another workstation.

So unless you know which directories are mounted on your workstation you are unaware of any network activity – the network is said to be **transparent** and you have **transparent** access to the files.

As NFS makes it so easy to access the resources of any other system on a network, it is normal for certain systems to act as central **servers** that provide resources for the other systems on the network called **clients**:



On a typical network, a server would be a mainframe computer with a lot of disc storage and a large amount of memory. The sample network used in this chapter shows a typical set-up.

One advantage of this client-server relationship is that all the user files can be stored on the server, thereby facilitating backups and generally making the onerous tasks of system administration that much easier.

Also, only one copy of a particularly large directory needs to be kept. For example, the set of manual pages `/usr/man` could be kept on the server and removed from the clients, thus freeing disc space.

An addition to NFS, called **yellow pages**, may also be running on your workstation. As it eases even further the duties of the System Administrator, this is more than likely to be the case!

Yellow pages is a distributed, network database containing key information about the characteristics of your network, such as the password file and the files describing the machines that can be accessed over the network. The database is normally stored and updated on a server machine (called the yellow pages **master**). The remaining machines in the network (yellow pages **slaves**) are then periodically updated with the files from this database.

The benefits of this set-up are substantial. An application can access data served by yellow pages, independent of the relative locations of the client and server. The job of the System Administrator is also eased as only one set of files needs to be updated, when a change in the profile of the network is required. For more information about setting up the yellow pages database on your network, refer to the *RISC iX System Administrator's Manual*.

The task of mounting file systems and directories, creating servers, setting up yellow pages on the network is looked after by your System Administrator. So before you use your workstation, you should contact your System Administrator for a profile of your network, including which workstations you have access to and what is mounted on your workstation.

The rest of this chapter describes the commands you can use to access workstations on your local network whose files are **not** mounted on your workstation.

The following list summarises the networking commands covered in this chapter:

Command	Description
rlogin(1C)	<i>remote log in</i> – log in to another workstation from your workstation.
rcp(1C)	<i>remote copy</i> – copy files between two workstations.
rsh(1C)	<i>remote shell</i> – execute a command on a remote workstation.
rup(1C)	<i>remote uptime</i> – find out how much a remote workstation is being used.
rusers(1C)	<i>remote users</i> – find out who the users of a remote workstation are.
rwall(1C)	<i>remote write to all</i> – send a message to all the users of a remote workstation.

The above commands are referred to as the **Berkeley networking commands**. They assume that the workstations are UNIX systems running Berkeley 4.3BSD and are using internet protocols over a suitable physical layer such as ethernet. A summary of the use of the commands is listed at the end of this chapter.

For information on accessing workstations that do not use internet protocols, refer to the next chapter, *Communicating with other systems and users*.

Setting up your network environment

Before using the above commands, you need to have certain files and access permissions set up on your workstation.

Some of the above commands require you to be able to log in to at least two workstations on the network (*rlogin*, *rsh* and *rcp*). For example, you need a login id on your workstation and permission to access the account of another user on a remote workstation. Your System Administrator usually sets this up for you, so if you are in any doubt, ask first.

You can decide which users on which remote workstations have the right to log in to your workstation and whether they should have to give a password in order to do so or not. In the same way, they can determine whether you should be able to log in to *their* workstations.

The files that control the levels of access on your network are the system files `/etc/hosts`, `/etc/hosts.equiv` and `/etc/passwd` and the hidden file `.rhosts` in your home directory. The files `/etc/hosts`, `/etc/hosts.equiv` and `/etc/passwd` are controlled by your System Administrator, so normally the only file that you can alter is the file `.rhosts` in your home directory.

You can communicate with the workstations whose names and internet addresses (the address used to locate your workstation on the network) are listed in the file `/etc/hosts`, so you should read the contents of this file to find out which workstations you have permission to log in to.

In the file `/etc/hosts`, each line contains a number referring to the internet address of the workstation followed by the hostname of the workstation. Any other names included alongside the hostname are discretionary **nicknames** assigned to each workstation by your System Administrator.

To find out the name of your workstation, at your normal command prompt type:

hostname

The name displayed is the name by which other users on other workstations refer to your workstation. For more information, refer to the manual page for `hostname(1)`.

You can let users of specified workstations log in using your user id without having to type a password.

The simplest case is where the two workstations have similar `/etc/passwd` files (ie all users have the same login name and userid on both workstations). and where the remote workstation name occurs in `/etc/hosts.equiv`. file of your workstation. This is usually set up by your System Administrator.

Having logged in to your local workstation, you are allowed to log in to the remote workstation (or use `rcp` or `rsh`), without having to give a password. This is only possible in a very open environment, or one where workstations are very secure.

Access without a
password

If the workstation you are trying to log in to (via `rlogin`, `rcp` or `rsh`) does not have your workstation name in its `hosts.equiv` file, then access is controlled by the file `.rhosts` in the home directory of the user you are trying to log in as on the remote workstation.

If this `.rhosts` file contains a line with your workstation name and user name, then access is granted without asking for a password. In the same way you can decide who has access to log in as you by entering their username and workstation name in your `.rhosts` file in your home directory.

For example, to add a new **trusted user**, ie someone you trust not to delete all your files, edit your `.rhosts` file to include the following line:

```
workstation [ username ]
```

where `workstation` is the name of the remote workstation used by `username` to log in to your workstation. The named user will then be able to log in to your workstation using your id, from the workstation specified, without having to type a password.

For example, here is a typical `.rhosts` file, containing two trusted users:

```
locke      einstein
acorncpd   newton
```

this allows users `einstein` and `newton` to log in to your workstation with your id, from the remote workstations `locke` and `acorncpd` respectively.

Note that the `username` argument is optional and if omitted, only lets you log in from the remote workstation.

`.rhosts` is one of many hidden files that you can create in your home directory. For more information about other hidden files, refer to the earlier chapter, *Using the UNIX shell*.

If neither the `.rhosts` file nor the `/etc/hosts.equiv` file has been set up to allow you remote access from your workstation, then you will not be able to use `rsh` or `rcp` commands, since a password is required before you can obtain access and these commands never prompt for a password.

Password access to
your workstation

**Logging in to other
workstations using
rlogin**

Possible errors using
rlogin

Therefore, you should ensure that your workstation is set up correctly before you try using any of the following commands as they are so dependent on correct access permissions.

If the above authentication process fails, then `rlogin` command will prompt you for a password. It is read without echo, just like a normal login procedure.

With your workstation set up correctly, the remaining sections describe how to use the networking commands available.

This section outlines how to access other workstations from your workstation using `rlogin`.

To log in to a remote workstation as yourself, using `rlogin`, type:

```
rlogin workstation
```

You may be prompted for a password – if so, type it in. The rest of the login procedure is very much the same as normal. When the shell prompt appears, it means you are successfully connected to the remote workstation *workstation* and you can execute commands and run programs as you normally would on your own workstation.

For example, you could also use the `rlogin` command again to log in to yet another remote workstation from the remote workstation you are already connected to.

If your workstation doesn't recognise the remote workstation you are trying to log in to, you may get the message:

```
workstation unknown host
```

This may be because:

- you have made a mistake typing the name of the remote workstation – try typing the command again.
- the name and address of the workstation is not listed in your `/etc/hosts` file – see your System Administrator.

If your workstation doesn't recognise the login id you are using, you will get the error message:

```
Login incorrect
```

Logging in with a specified username

This means that the workstation does not contain a valid login id for you – see your System Administrator.

For details about other errors, refer to the reference section *Trouble-shooting*, at the back of this guide.

Where you have a different username on the remote workstation where you want to log in, type:

```
rlogin workstation -l username
```

username in this case is the username you use on the remote workstation.

Also, if you log in as someone else who has an entry for you in his *.rhosts* file you can log in as him, again without having to specify a password.

It is quite possible that a *guest* login id has been set up on each workstation on your network by your System Administrator. This is normally not password protected and allows users to log in to the specified workstation, albeit with restricted access. For example:

```
rlogin workstation -l guest
```

Your workstation contains a *guest* home directory (*/usr/users/guest*) containing sample text files for first-time UNIX users that are used in the earlier chapters of this guide.

Logging out of a remote session

Log out of a session on a remote workstation in the normal way, by typing:

```
exit
```

or whatever means you normally use to log out.

If you can't log out for some reason (for example, because the workstation hangs while you are using it), you can **abort** the session by typing, at the beginning of a new line:

```
~.
```

A tilde (~) appearing as the first character of a line is an escape signal and directs *rlogin* to perform some special action. The dot means 'break the connection'. After this command, you will be returned to your original workstation.

Sending commands to remote workstations

Requesting network information using `rup` and `rusers`

Possible errors using `rup` and `rusers`

If you have used multiple `rlogin`'s to **leap-frog** from one workstation to another, then typing '~.' brings you back to the workstation you first logged in to.

There are several commands you can execute on remote workstations, without being logged in to them. These commands enable you to

- request network information – using `rup` and `rusers`
- send messages – using `rwall`
- execute remote commands – using `rsh`.

Using `rup`

To find out how much a particular workstation is being used, type:

```
rup workstation
```

This command gives a status similar to the local command `uptime(1)`. The information for the workstation will be displayed in the following form:

```
workstation up 1 day, 2.45, load average: 0.05, 0.00, 1.20
```

Using the command `rup` without specifying a *workstation* will display the status of all the workstations connected to your local network.

Using `rusers`

To find out who the users of a remote workstation are, type:

```
rusers workstation
```

This command gives a status similar to the local command `users(1)`. If you omit *workstation*, the users of all the workstations on your local network will be listed.

Be careful when using `rup` and `rusers` with no *workstation* specified. If any workstation connected to the network has these commands disabled, then the command will take a long time before it gives up interrogating that workstation. To escape from this predicament, type <CTRL-C>.

Sending messages using `rwall`

You can send simple messages to remote workstations with the command `rwall`, by entering them in this form:

```
rwall workstation  
here is a sample  
message comprising two lines  
<CTRL-D>
```

The message will be transmitted to the users of the other workstation, together with the time it was sent and who sent it. For example, if you receive the following message, it indicates that user `einstein` sent you a message from the workstation `locke`:

```
Broadcast message at 12.15 ...  
broadcast message from locke!einstein: here is a sample  
message comprising two lines
```

Possible errors using `rwall`

Like `rup` and `rusers`, this command may also be disabled on the remote workstation you are directing your message to. If it is, then the message will not be received.

You may also find when you type this command that you get the error message:

```
rwall: Command not found
```

If this is the case, you may need to change your **search path** – the command `rwall` lives in the directory `/usr/etc`. To change your search path, refer to the earlier chapter *Using the UNIX shell*.

Executing remote commands using `rsh`

You can call up a shell and execute a command from a remote workstation using the command `rsh`. The syntax for this command is:

```
rsh workstation command [ arguments ]
```

The `rsh` command doesn't log you in to the remote workstation – it communicates directly with a **daemon** on the remote workstation, which is a process that runs in the background and executes the command on the remote workstation.

Beware of trying to use shell metacharacters with *command*, as the shell expansion will take place on your local workstation before the remote workstation – so metacharacters have to be **escaped** if they are to be interpreted remotely. To be sure, type the command explicitly.

If you omit *command*, instead of executing a single command, *rsh* tries to log you in on the remote workstation as though you had typed *rlogin*.

Using *rsh* enables you to use commands or facilities that are available on a remote workstation but not on yours, such as a piece of hardware like a printer.

However, you must ensure that the right access permissions are set up for you to be able to use these facilities.

Copying files using *rcp*

rcp is used to copy files and directories *to* and *from* remote workstations running UNIX, for which you have login access.

You should be aware that there is more than one command you can use to copy files over a network. For details of the utilities you can use to copy files that do not support internet protocols, see the next chapter, *Communicating with other systems and users*.

General syntax

The general syntax for copying a file **from** a workstation **to** another workstation, using *rcp* is very similar to the *cp* command, discussed in the chapter *Using UNIX*. For example

```
rcp <from> <to>
```

where *<from>* can refer to your local workstation and *<to>* can refer to a remote workstation on the network. In this case the command is used as follows:

```
rcp sourcefile workstation:destdir
```

sourcefile should be the name of the file on your workstation.

destdir should be the full pathname of the directory on the remote workstation where you want the copy of the file to be put.

You could also use *rcp* the other way round, where *<from>* can refer to a remote workstation and *<to>* can refer to your local workstation.

Copying from a remote workstation using rcp

In this case the command is used as follows:

```
rcp workstation:sourcefile destdir
```

For example, to copy the file `/tmp/chapter5.doc` from the workstation `acorncpd` to the directory `/tmp` on your workstation, you would type:

```
rcp acorncpd:/tmp/chapter5.doc /tmp
```

If you want to change the name of the file when it arrives in the directory on your workstation, specify a filename instead of a directoryname as the second argument.

For example, to rename `chapter5.doc` as `chap5` you would type:

```
rcp acorncpd:/tmp/chapter5.doc /tmp/chap5
```

Note that `rcp` requires you to have login access and the correct access permissions, for the file on the remote workstation.

Copying to a remote workstation using rcp

For example, to send a copy of the file `quotes` to the `/tmp` directory of the remote workstation `acorncpd`, you could type:

```
rcp /usr/users/guest/editing-files/quotes acorncpd:/tmp
```

A copy of the file `quotes` would then appear in the `/tmp` directory of the remote workstation `acorncpd`, provided that you had the appropriate read and write permissions for the pathname specified.

Copying a directory

The syntax for copying whole directories from one workstation to another is very similar to that used for copying a file with the `cp` command. For example:

```
rcp -r workstation:sourcedir destdir
```

To transfer one of your directories and all its sub-directories to another workstation from your own workstation, type:

```
rcp -r sourcedir workstation:destdir
```

Networking command summary

The following list summarises the major networking commands covered in this chapter:

Command	Syntax	Use
rcp	rcp <i>sourcefile workstation:destfile</i>	Transfer a copy of your file to a remote workstation and call it <i>destfile</i> .
	rcp <i>workstation:sourcefile destfile</i>	Copy a remote file and transfer the copy to <i>destfile</i> on your workstation.
	rcp -r <i>sourcedir workstation:destdir</i>	Copy a directory and sub-directories from your workstation to a remote workstation, and call it <i>destdir</i> . Can also be used in the other direction.
rlogin	rlogin <i>workstation</i> [-l <i>user</i>]	Log in to the remote <i>workstation</i> (optionally, using a different username).
	~.	Abort an rlogin connection.
rsh	rsh <i>workstation command</i>	Execute <i>command</i> on workstation specified.
rup	rup [<i>workstation</i>]	Display system status information for all workstations on the network, (optionally, on workstations specified only).

<code>rusers</code>	<code>rusers</code> [<i>workstation</i>]	Display all network users logged in (optionally, users on specified workstation only).
<code>rwall</code>	<code>rwall</code> <i>workstation</i>	Send the message which appears on the following lines to users of the workstation specified.

Key:

<i>destfile</i>	the destination filename (the file to which the copy is to be sent).
<i>destdir</i>	the destination directory (the directory to which the copy is to be sent).
<i>workstation</i>	the name of the remote workstation.
<i>sourcedir</i>	the name of the directory which is to be copied.
<i>sourcefile</i>	the name of the file which is to be copied.

Sources of further information

For further information about the commands discussed in this chapter, refer to the appropriate manual page entry for the command.

Error messages that you may receive while using these commands are outlined in the reference section *Trouble-shooting*, at the back of this guide.

Communicating with other systems and users

Introduction

The previous chapter, *Networking and NFS*, described UNIX utilities for accessing workstations connected together using internet protocols on a local area network.

This chapter introduces the utilities you can use to communicate with machines that do not fall into this category and also introduces the more general UNIX commands that you can use to communicate or just exchange information with other systems and users.

The topics covered in this chapter include:

- **Accessing remote machines** – logging in to machines on your local network that do not support the Berkeley networking commands described in the previous chapter.
- **Serial line communications** – communicating with other machines over a serial line.
- **Electronic mail** – sending information using the UNIX electronic mail program.
- **Interactive communication** – communicating with other users who are logged in on your local network.
- **Floppy disc utilities** – transferring information using floppy discs for use with other types of workstations – including UNIX workstations, Acorn *Archimedes* RISC OS workstations and MS-DOS based workstations.

Accessing remote machines

The following commands are discussed in this section:

- `ftp` and `tftp` – (short for **file transfer protocol** and **trivial file transfer protocol** respectively) used to copy and transfer files to and from remote machines that do not support the Berkeley networking command `rcp`.
- `telnet` – used to access machines that do not support the Berkeley networking command `rlogin`.

Using ftp and tftp

The above commands are primarily used to access machines that do not support the Berkeley networking commands discussed in the previous chapter. However, these commands use the internet communications protocol, so you can still use them between Berkeley 4.3BSD workstations. In some cases, you may choose to use `ftp` in favour of `rcp` due to the greater range of facilities it offers.

`ftp` is a very elaborate program that provides a suite of options and commands for transferring files.

`tftp` is a simplified version of `ftp`. Although easier to use, it can only be used to copy publicly-readable files and also displays very esoteric error messages when things go wrong. It should really be used as a last resort for copying files.

Using the file transfer protocol – ftp

To copy a file from your local workstation to a remote machine using `ftp`, follow these steps:

- At your normal shell prompt, type:

```
ftp
```

- After a few seconds the '`ftp>`' prompt is displayed. At this prompt type:

```
open machinename
```

where *machinename* is the name of the remote machine.

- Wait until a connection has been established. This is indicated by the following prompt being displayed:

```
Name (machinename:username):
```

- If you are successfully connected, type in your *username* and *password* for the remote machine.

If you do not have a user id for this machine, try logging in using the `guest` login id. If this fails, contact your system administrator for an appropriate login id.

- If you successfully log in to the remote machine, the '`ftp>`' prompt will appear again – the connection has been established.

- To transfer a file, type:

get *sourcefile destfile*

sourcefile is the name of the file on the remote machine that you wish to copy to your workstation.

destfile is the full pathname of the file on your workstation where you want the file to be copied to.

- When the transfer has been completed and the 'ftp>' prompt reappears, type `quit` to end the connection.

To **transfer** a file from your workstation to a remote machine using `ftp`, follow this procedure:

- Type **ftp** and wait for the 'ftp>' prompt to appear.
- Open the connection and log in, as described above.
- When you see the 'ftp>' prompt, type:

put *sourcefile destfile*

sourcefile is the full pathname of the file on your workstation.

destfile is the full pathname of the file on the remote machine where you want the copy to go to. As with `rcp`, you must have read and write permissions for the pathname specified.

- When you are notified that the transfer is complete and the 'ftp>' prompt re-appears, type `quit` to terminate the connection.

Getting help for ftp

Note that during any time, with the 'ftp>' prompt displayed, you can type:

help

to display a full list of the commands that you can issue from the 'ftp>' prompt.

Using the trivial file transfer protocol `tftp`

`tftp` is similar to `ftp`, but can only be used to copy publicly-readable files. This is the procedure to follow to **copy** such files (the procedure and command syntax is similar to `ftp`):

- Type **`tftp`** and wait for the '`tftp>`' prompt to appear.
- At the '`tftp>`' prompt, type:

```
connect machinename
```

where *machinename* is the name of the remote machine.

- When a connection has been established, the '`tftp>`' prompt will appear again. At this prompt, type:

```
get sourcefile destfile
```

sourcefile is the full pathname of the file on the remote machine that you wish to copy to your workstation.

destfile is the full pathname of the file on your workstation – ie where you want the file to be copied to.

- When the transfer has been completed, type `quit` to terminate the connection to the remote machine.

To use `tftp` to **transfer** a file to a remote machine, a file with the same name must already exist on the remote machine and you must have read and write permission to copy your file to it.

This is the procedure to follow:

- Type **`tftp`** and wait for the '`tftp>`' prompt to appear.
- At the '`tftp>`' prompt, type:

```
connect machinename
```

where *machinename* is the name of the remote machine.

- When a connection has been established (the '`tftp>`' prompt will appear), type:

```
put sourcefile destfile
```

sourcefile is the name of the file on your workstation.

destfile is the full pathname of the file on the remote machine where you want the copy to go to. Again, as with *rcp*, you must have read and write permissions for the pathname specified.

- When you are notified that the transfer is complete and the *tftp* prompt reappears, type *quit* to terminate the connection to the remote machine.

Getting help for *tftp*

Note that during any time, with the '*tftp>*' prompt displayed, you can type:

```
? ↵
```

to display a full list of the commands that you can issue from the '*tftp>*' prompt.

For more information, refer to the manual page entries for *ftp(1C)* and *tftp(1C)*.

Using telnet

You can use *telnet* to log in to machines that do not support the Berkeley *rlogin* command.

To use *telnet*, type:

```
telnet machinename
```

If the connection is successful, you will be told that you have been connected, and given an escape character (you will use this when you need to abort a *telnet* connection). For example:

```
telnet acorncpd  
Trying 89.0.0.2 ...  
Connected to acorncpd.  
Escape character is '^]'.  
4.3 BSD UNIX (acorncpd)  
login:
```

Serial line communications

At the 'login:' prompt, try to log in as normal. If successful, you can execute commands as if you were using your own workstation.

To abort a `telnet` connection during any time, type the 'Escape character' on a new line; which would be `<CTRL-]` in the above example.

After typing this, you will see the '`telnet>`' prompt. At this prompt, type:

```
quit
```

During any time, with the '`telnet>`' prompt displayed, you can type:

```
help
```

to display a full list of the commands that you can issue.

If you fail to connect to the remote machine, leave `telnet` by typing:

```
quit
```

For more information, refer to the manual page entry for `telnet(1C)`.

You can connect with machines that are not on your local network but which are accessible via a directly connected serial line or via telephone lines, by using any of the following commands:

- `tip` (short for *terminal interface processor*)
- `uucp` (short for *UNIX to UNIX copy program*)
- `kermit` (general purpose copy program)

In order to use any of the above commands, your workstation will first have to be set up to communicate with the outside world.

All the information is transmitted down the serial line at the back of your workstation (called `/dev/serial`), so before using `tip`, check that your serial port is connected.

If you are connecting your machine via telephone lines, you will also require a **modem** (short for **modulator-demodulator**) to convert the output from your workstation into a form which can be transmitted down the telephone lines. Another modem at the other end translates the signals back again into a form which is readable by the machine there.

It is beyond the scope of this guide to explain in detail how to configure your network to communicate with remote machines in this way. However, in normal operation over telephone lines, you may get involved in **baud rates**. The baud rate is the speed at which data is transmitted down a line. This is usually measured in bits per second, and the most common speeds are 1200 and 9600 baud (1200 and 9600 bits per second).

Obviously, both machines must be set to transmit and receive data at the same speed, or – like one juggler throwing clubs to another faster than he can catch them – they won't be able to communicate properly. Some machines adjust baud rates automatically, but if you see rubbish on your screen once a connection has been made, you may need to check the transmit and receive baud rates of the two machines.

As well as having your hardware configured correctly, you will also need to have the appropriate files set up on your workstation that describe the characteristics of the remote machine – for example, `/etc/phones` and `/etc/remote`.

For information about setting up these files, refer to the manual page entry for `tip(1C)`. Refer to the manual page for `stty(1)`, for information on how to set up the baud rate on your system.

For more information about setting up `kermit` and `uucp` on your RISC iX workstation, refer to the *RISC iX System Administrator's Manual*.

Electronic mail

Your system provides a number of ways of communicating electronically with other users. This section tells you about sending and receiving electronic mail messages (using `mail`).

Electronic mail combines the immediacy of a telephone conversation with the permanence of a letter: you can get straight through to another user, and you will also have a written record of your communication.

The mail program described in this chapter is the `/usr/ucb/mail` version, which is the one most often used on Berkeley 4.3 BSD systems. If you are not sure if you are using this version of electronic mail, check with your system administrator and also check that you are set up to receive electronic mail.

When someone sends you a mail message, the message is collected by the mail system and stored in your **system mailbox** file. When you next log in, you are notified if there is any mail for you in your system mailbox file.

When you choose to read your mail, `mail` opens your system mailbox and displays all the mail messages that have been sent to you, with the name of the sender and the date it was sent attached to each one. You can then choose to read, delete, reply to or save these messages.

The following sections describe how to perform each of the above actions.

Sending a mail message

To send an electronic mail message within your local network (ie, not across a gateway or via a dial-up network), use the `mail` program, the main electronic mail facility on your system.

Follow this procedure:

- Type:

```
mail username@machinename
```

username is the name of the person you are sending the message to and *machinename* is the name of the machine where the user receives their mail. You won't need to add the `@machinename` if you are sending mail to someone who uses your workstation.

- Type in the message you want to send.
- End the message with `<CTRL-D>`.

Typing in a message
using vi

For example:

```
mail romeo@verona
wherefore art thou?
<CTRL-D>
```

The screen will show EOT, to acknowledge the *End-Of-Text*. The mail message typed will then be sent to user `romeo` on the machine `verona`.

To send the same message to more than one user, list their login names, separated by a space, on the command line. For example:

```
mail romeo@verona einstein newton
wherefore art thou?
<CTRL-D>
```

The above message is then sent to user `romeo` on the machine `verona` and also users `einstein` and `newton` on the local workstation.

You can use the text editor `vi` (described in an earlier chapter, *Text editing*) to type in longer messages. To use `vi` from within `mail`, type `~v` at the beginning of a line. For example:

```
mail romeo@verona
~v
```

The tilde character (`~`) temporarily escapes from `mail` and the following character (`v`) is interpreted as a command – in this case a command to start up the text editor `vi`. Other **tilde escape** commands available like this will be discussed later in this section.

However, note that over a network the tilde character is sometimes interpreted as an escape character, so if you are remotely logged in (using `rlogin`) you will have to type at least two tildes to get the host machine to interpret one tilde. If you are logged in over more than one machine, you will have to type a tilde for each machine.

When you have composed your message, type :

```
<ESC> :wq
```

to return to mail. A temporary file is created containing your text and mail asks you if you would like to continue to add any more text:

```
"/tmp/Re778" 23 lines, 1096 characters
```

```
(continue)
```

After this prompt, add any more text, then type <CTRL-D> on a new line to send the message. The vi file you created plus any text you typed after '(continue)' is sent as a complete mail message.

To abort a final message

To abandon a message before it is sent (ie, before you type <CTRL-D>), type your interrupt character (usually <CTRL-C>). You will be asked to confirm that you want to abort. A second interrupt character will confirm it. For example:

```
mail romeo@verona
```

```
Here is an example of an aborted mail message<CTRL-C>
```

```
(Interrupt -- one more to kill letter)
```

```
<CTRL-C>
```

Sending mail to an unknown user

If the user is unknown on the network, the *mailer-daemon* (the mail delivery sub-system) will send your message back to you some time later and also notify the *Postmaster* (either you or your system administrator) that it failed to deliver it.

Security in the mail system

You should not really use any sort of electronic mail if you are sending private or sensitive information. It could quite easily be intercepted and read by someone else.

Reading your mail

mail allows you to:

- see a list of your mail messages
- view individual messages
- save messages to different files
- delete files.

To see a list of your messages, type:

mail

mail does not display the whole contents of your system mailbox, but instead summarises the contents of each message in a **message header** and lists them out in the following form:

```
Mail version 5.2 6/21/85. Type ? for help.
"/usr/spool/mail/newton": 2 messages 2 new

>N 1 mother Mon Mar 22 15:17 4/120 laundry
  N 2 einstein Tue Mar 23 11:37 12/347 relativity
&
```

From the above example, it appears there are two messages. Message number 1 is from your mother about laundry and message number 2 is from einstein about relativity.

The '&' is the standard mail prompt from where you can issue a large number of mail commands. Some of these commands will be discussed in the next few sections.

The upper-case N at the start of each line signifies that both messages are new messages that have not yet been read.

Fortunately you can skip through mail messages without having to read them in the order they were sent. So, in the above example, to read the message from einstein first, at the mail prompt '&', type:

2

The message will be displayed, in the following form:

```
Message 2:
From einstein Tue May 10 12:30:47 1988
Date: Tue, 10 May 88 12:30:41 GMT
From: einstein (Albert Einstein)
To: newton
Subject: relativity
```

Things should be made as simple as possible, but no simpler.

&

To redisplay the above message (the current mail message) again: at the mail prompt, type:

p

To read the next message in the list, type:

↓

The subsequent message will be displayed. If there are no more messages to be displayed, as in the example, you will receive the message:

&

At EOF

&

To go back and read the the first message, type :

1

There is a useful help facility that lists the commands available from within mail. To access it, at the mail prompt type:

help

A long list will be displayed of the mail commands and their syntax.

You can also get a full list of the tilde escape commands available when you are composing a mail message, by typing '~?'. For example:

here is some text

~?

The following ~ escapes are defined:

~~ Quote a single tilde
~b users Add users to "blind" cc list
...

Asking mail for help

If you have no mail

The list is not included in your mail message and you can carry on typing the rest of your message. Some of the tilde escapes listed will be discussed later on in this section.

If there is no mail for you, after typing:

```
mail
```

you will get the message:

```
No mail for username
```

If this occurs, ring up some friends on your local network and ask them to send you some mail, so you can try out some of the features of mail. Failing that, send yourself some!

Sorting your mail

You can sort your mail messages, save the ones you want to keep into different files and delete the ones you don't want.

To **save** a mail message, or a series of mail messages, to a file, type:

```
save messagenumber1 messagenumber2 ... filename
```

You can abbreviate *save* to *s* and you can also specify a range of messages to be saved in one file. For example:

```
s 2-9 mail-file
```

saves the mail messages and mail headers from 2 to 9 in the file *mail-file*.

In either case, mail will respond with the name of the file created, its status and size. For example:

```
"mail-file" [New file] 476/13319
```

If you don't specify any *messagenumbers*, only the current message will be saved.

Once saved, the easiest way to **read** a mail message is to use the *cat* command. For example, to read the messages contained in the file *mail-file*, at the normal shell prompt type:

```
more mail-file
```

Alternatively, you can display mail messages you saved to a file and read them as you would your normal system mailbox file. For example:

```
mail -f mail-file
```

This command is useful for browsing through the file and deleting or reading individually selected messages.

To **delete** a mail message, enter the `mail` program and at the standard `mail` prompt display a list of your messages by typing:

```
h
```

which is short for *headers*. If you have more than a screenful of messages, you can also type:

```
h messagenumber
```

which will display all the mail message headers from *messagenumber* onwards. For example, if only 16 mail messages are displayed and you have 20 mail messages as indicated at the top of the message header list, type:

```
h 17
```

to show the remaining message headers in your system mailbox – ie 17 to 20.

To **delete** any of the messages you have been sent, type:

```
d messagenumber
```

The message will be deleted.

If you mistakenly delete a message, type `u` (short for *undelete*). This will retrieve the last mail message you deleted.

Enter the `mail` program, and display a list of your messages, using the `h` command. To reply to any of the messages listed, type:

```
R messagenumber
```

If you don't include a message number, the current message will be assumed. `R` will send your reply only to the sender of the message.

Replying to mail

To reply to the sender along with any other recipients, type:

reply *messagenumber*

r may be used as an abbreviation for reply.

mail will automatically address the message to the sender of the message and use the same subject, if any, with 'Re:' placed before it. All you have to do is type the text, followed by:

<CTRL-D>

on a line by itself to end the message.

For example:

& **P**

Message 2:

From einstein Tue May 10 12:30:47 1988

Date: Tue, 10 May 88 12:30:41 GMT

From: einstein (Albert Einstein)

To: newton

Subject: relativity

Things should be made as simple as possible, but no simpler.

& **reply 2**

To: einstein

Subject: Re: relativity

Thanks for your message - but what does it mean?

<CTRL-D> EOT

&

If you want to **add** a copy of the message to which you are replying (as an aide-mémoire to the recipient, for example), or a copy of any other relevant message, you can do this by typing, at any point in the message:

~m *messagenumber*

mail doesn't display the message to be inserted, but confirms that it is doing so with the message:

```
Interpolating: messagenumber
```

```
(continue)
```

Again, if you are logged in remotely, watch out for tildes being interpreted incorrectly by the host machine.

After interpolating the message, continue typing your mail message, ending the message with a <CTRL-D> in the normal way.

You can **insert** a standard text file into a mail message in just the same way as you insert another mail message. Start writing the message and when you get to the point where you want to insert the file, type:

```
~r filename
```

Use the absolute pathname of the file to be inserted, to be sure that mail looks in the right place in the file system for the file. If it finds the file, mail will echo the name of the file, along with the number of lines and characters it contains, followed by EOT. For example:

```
here is some text before the inserted file
```

```
~r file1
```

```
"file1" 12/303
```

```
and here is some text after the inserted file
```

```
<CTRL-D> EOT
```

The file specified will be included in your mail message at the point specified. As shown above, after you have inserted the file you can continue with your message, and complete it with a <CTRL-D> in the normal way.

Quitting the mail
program

To quit the mail program, at the mail prompt type:

```
quit
```

(quit may be abbreviated to q).

When you leave `mail`, any messages which have been read, but not saved or deleted, are placed in a file called `mbox` in your home directory. A confirmatory message will be displayed. For example:

```
& quit
Saved 2 messages in mbox
$
```

The `mbox` file is similar to your system mailbox file but is a private mail in-tray file located in your home directory.

Unread messages will remain in your mailbox and will be displayed again the next time you enter the `mail` program, prefixed by upper-case `U` – this signifies **unread** mail.

To leave `mail` without saving the unread messages in the file `mbox`, type `exit`. The messages are left in your system mailbox and displayed again when you enter `mail`. Your private mail in-tray `mbox` remains untouched.

Looking at your mail
from outside the mail
program

To get a quick summary of the contents of your system mailbox from outside the `mail` program, use the `from` command. This displays the 'from' lines in your system mailbox. For example, at your normal shell prompt, type:

```
from
```

A list of the senders and the date sent of the mail messages contained in your mailbox is displayed on the screen.

For more information, refer to the manual page for `from(1)`.

Customising mail

`mail` has a number of options that you can include in a hidden file called `.mailrc` in your home directory to customise how you use the `mail` program.

For example, to set up `mail` so that it prompts you for a subject of each message you send, you could enter in your `.mailrc` file:

```
set ask=Yes
```

If this line is included, whenever you send a message to anyone, you have the option of assigning a subject header to the message.

For example:

```
mail romeo@verona  
Subject: a plea to your heart  
wherefore art thou?  
<CTRL-D>
```

This is especially useful when you come to browse through a long list of mail messages. The subject of each message is displayed alongside the sender of the message and the date it was sent.

If you do not want to add a subject to the message, just type ↵ at the 'Subject:' prompt.

You could also set up `mail` to prompt you for additional carbon copy recipients of the message you are sending. For example, you could enter the following line in your `.mailrc` file:

```
set askcc=Yes
```

Whenever you send a message to anyone you have the option of adding a carbon copy list. For example:

```
mail romeo@verona  
Subject: a plea to your heart  
wherefore art thou?  
<CTRL-D>Cc: newton
```

Wide area network mail

You can also send electronic mail to other users over wide area networks provided that all the necessary network connections have been established. Different networks will have their own procedures for sending mail.

For more information about sending mail over a wide area network, contact your system administrator.

For more information about the mail commands described and the customising options for `mail`, see the manual page entry for `mail(1)`.

Interactive communication

Checking who's logged in

There are a number of utilities on your system for communicating interactively with other users on your local network. For example:

- talk
- write
- wall

To see who's logged in before you use any of the above utilities, you can use one of three commands, `users`, `who` and `w`.

`users`

To display a list of the current users of your workstation, type `users`. For example:

`users`

```
jon dave jim
```

`who`

You can also use the command, `who`. For example:

`who`

The `who` command shows more information than `users`. For example:

```
jon          tty0          May 10 10:01
dave         tty1          May 10 10:28
jim          console       May 10 09:58
```

`w`

The `w` command displays even more system information than `who`, by also displaying the programs that each user is running, when they logged in etc.

For more information, refer to the manual pages for `users(1)`, `who(1)` and `w(1)`.

Using talk

talk provides you with the facility to exchange messages interactively with other users. To start talking interactively to other users, type:

```
talk username@machinename
```

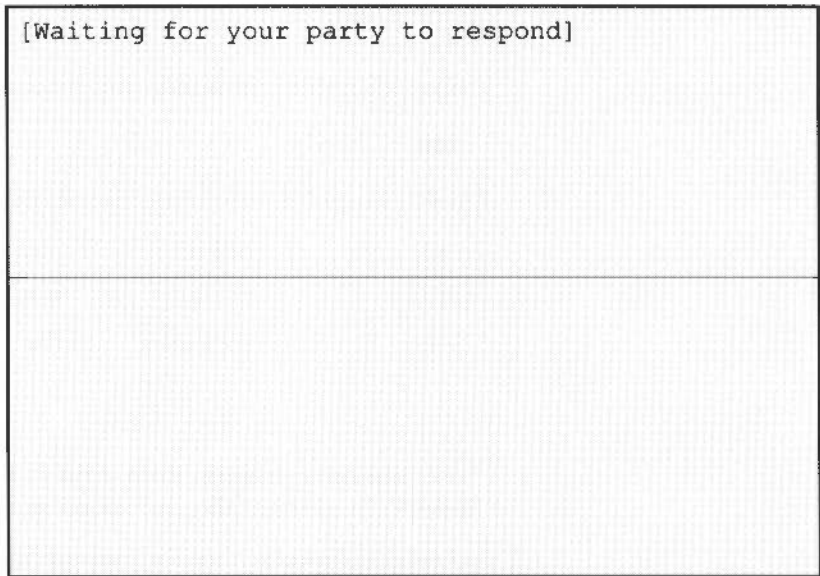
The *@machinename* argument is not needed when both parties are logged in on the same machine.

Your system will attempt to make a connection with the other user's machine (until this is done, the message [No connection yet] will be displayed).

Once a connection has been established, your machine will 'ring' the remote machine by flashing the message:

```
Message from Talk_Daemon@yourmachinename at time ...
```

During this time the message [Waiting for your party to respond] appears on your screen, which is now split in two:



```
[Waiting for your party to respond]
```

Replying to a talk call

If you get the message:

```
Message from Talk_Daemon@remotemachinename at time ...
```

you respond by typing:

```
talk user@remotemachinename
```

When the two users have agreed to talk, `talk`'s interactive screen is displayed on both machines. Users can now exchange messages independently and even at the same time, without clashing.

Terminating a talk session

When you have finished your talk session, or if you don't get through to the other user, exit from `talk` by typing:

```
<CTRL-C>
```

For more information on `talk`, see the manual page entry for `talk(1)`.

Using write

To send a message to a user on another terminal of the same system as yourself, you can use the program `write`.

At you normal shell prompt, type:

```
write username [ttyname]
```

```
Here is the text of the message
```

The message will be sent when you press `↵` at the end of the line. To write to a user who is logged in more than once, use the `ttyname` argument to indicate the appropriate terminal name.

Replying to a written message

Messages sent with `write` appear on your screen as follows:

```
Message from machinename!username on ttyprn at time...
```

```
Message follows here
```

To reply to the message, just type:

```
write username
```

```
Here is my reply
```

```
↵
```

After typing the ↵ key, your message is sent back to the other user. All further text that you type in is sent to the your partner's terminal, until you both type <CTRL-D> to end the session.

However, since you are now simultaneously typing and receiving messages, you will end up with garbage on your screen unless you work out some sort of scheduling procedure with your partner.

You might try the following convention protocol: when you first write a message, wait for your partner to write back before you start to send. Each person should end each message with a distinctive signal, -o- (for 'over') is standard - so that your partner knows when to begin a reply. To end your conversation, type -oo- (for 'over and out') before finishing the conversation with <CTRL-D>.

For more information on `write`, see the manual page entry for `write(1)`.

To stop people talking and writing to you

You can prevent `write` and `talk` messages from appearing on your screen, if you really want to, by adding the command `'mesg n'` on a line by itself in your `.profile` file (or `.login` file if you are using the C shell).

This will prevent every user, except `root`, from interrupting you with either `talk` or `write`.

To make yourself available to `talk` and `write` again, change `'mesg n'` to `'mesg y'`.

Note that your new `.profile` or `.login` file must be read again by your login shell for the changes to take effect. The easiest way to do this is to log out and then log in again.

Using wall

You can write simple messages addressed to everyone logged in to your workstation with the command `wall`, entering them in this form:

```
wall  
  
Here is a message to all users  
of this RISC iX workstation  
  
<CTRL-D>
```

The message will be transmitted to all other users of your workstation, together with the time it was sent.

The `wall` command is useful for important messages which all users need to know, such as when the machine is going down for maintenance. Use this command sparingly, as most users find unimportant interruptions distracting!

For more information on `wall`, see the manual page entry for `wall(1)`.

System messages

You may see a system message when you first log in, welcoming you to the system or telling you about backup times or when the system will be shut down for maintenance, so you should always read and take note of these messages.

The text of the messages is contained in the system file `/etc/motd` (short for *message-of-the-day*) and is usually owned and edited by `root`. If you forget to read the message when you first log in, you can read the message again by typing:

```
more /etc/motd
```


Floppy disc utilities

Transferring files between two UNIX workstations

Storing information on floppy discs is useful for transferring information to other UNIX workstations that you cannot access directly using the communications utilities described previously.

This section describes how to copy information to floppy discs for use by:

- other UNIX workstations
- MS-DOS based workstations
- Acorn *Archimedes* RISC OS workstations.

The following procedure describes how to transfer files between two UNIX workstations. This includes RISC iX workstations and any other UNIX workstation that supports `tar` (the *tape archiver* program) and also has a 3.5" floppy disc drive.

You can transfer textual and binary files between two RISC iX workstations, but you can only transfer textual files between a RISC iX workstation and another UNIX workstation.

In order to be able to transfer files between two workstations, you need to be able to log in as `root` on both workstations.

First, you need to format the disc. Insert a blank unformatted floppy disc into your disc drive (having made sure that the write-protect slide tab is covering the hole), and type:

```
ffd
```

```
Commencing format of /dev/rfdf1024
```

```
Commencing read check
```

```
Format completed satisfactorily
```

`ffd` (short for *format floppy disc*) writes track layout information onto the disc: this is needed if you are going to store files on the disc.

`/dev/rfdf1024` is the device name corresponding to your floppy disc drive.

Once the disc is formatted, a quick read check is performed to check that the disc is not corrupt. If this check is successful, the confirmatory message, 'Format completed satisfactorily' is displayed.

With the floppy disc formatted, you can now copy files onto the floppy disc.

For example, if you have two files (*file1* and *file2*) that you wish to transfer to another UNIX workstation, type:

```
tar cvf /dev/fdf1024 file1 file2  
  
a file1 1 blocks  
a file2 1 blocks
```

The command `tar` can be used for saving and restoring files between workstations that may not use the same file formats. `tar` produces one large file in a standard format containing the files specified, which is written onto your floppy disc and can then be transferred to the remote workstation.

You can check that the files have been copied successfully, by typing:

```
tar tvf /dev/fdf1024  
  
rw-r--r-- 0/0  291 Nov 21 16:19 1988 file1  
rw-r--r-- 0/0   23 Nov 21 16:19 1988 file2
```

To copy the archived files from the floppy disc to the remote workstation, log in to the remote workstation as `root`, change directory to where you want to put the files, insert the floppy disc containing the two files and type:

```
tar xvf /dev/fdname
```

where *fdname* is the floppy disc device name of the remote workstation. For example, if you were copying the files on to another RISC iX workstation you would type:

```
tar xvf /dev/rfdf1024  
  
x file1, 291 bytes, 1 tape blocks  
x file2, 54 bytes, 1 tape blocks
```

As indicated by the system messages above, this command extracts the two files *file1* and *file2* from the floppy disc and copies them into your current directory on the remote RISC iX workstation.

For more information, refer to the following manual pages – `tar(1)` and `ffd(8)`.

Transferring MS-DOS files to your workstation

MS-DOS version 3.2 floppy discs may be accessed using the following five floppy disc utilities:

- `msdosls` – lists the contents of an MS-DOS floppy disc in a format similar to `ls` and with similar options.
- `msdoscat` – copies the contents of a file or files on an MS-DOS floppy disc to the standard output similar to `cat`.
- `msdoscp` – copies a file or files on an MS-DOS floppy disc to a similarly-named file or files under UNIX. Directory trees on the floppy disc may be searched recursively.
- `wrmsdos` – copies or replaces files on an MS-DOS floppy disc. Options are available to format and/or initialise the MS-DOS structure on the floppy disc. Directories may be created, and recursively copied.
- `msdosrm` – removes files and directories from an MS-DOS floppy disc.

The above utilities are suitable for copying text files only.

In MS-DOS, text files are held differently from under UNIX, in that each line is terminated by a carriage-return as well as a line feed, and the file is terminated by `<CTRL-Z>`.

The MS-DOS utilities therefore convert files to and from this format when they are copied. Conversion may be suppressed by describing the file(s) as binary by supplying the option `-b`. If this option is omitted, but the files *look* as though they are binary, a warning message is output.

Also note that IBM PS/2 models 50 upwards use quad density (1.44MByte) formats, which although supported by the floppy disc utilities are not available under RISC iX.

Copying MS-DOS files to RISC iX

If you have a formatted MS-DOS floppy disc that contains information you wish to transfer to your RISC iX workstation, insert the floppy disc and type:

```
msdosls
```

to check that the disc is readable.

To copy two text files `file1` and `file2` to the `/tmp` directory on your workstation, type:

```
msdoscp -V file1 file2 /tmp
```

```
Created file /tmp/file1: 46 bytes
```

```
Created file /tmp/file2: 54 bytes
```

As indicated by the system messages above (generated by specifying the `-V` option), the two files specified are copied from the floppy disc to `/tmp` on your workstation.

To check that the file was successfully copied, list the contents of the current directory, using the `ls` command.

For more information, refer to the following manual pages - `msdoscat(1)`, `msdoscp(1)`, `msdosls(1)`, `msdosrm(1)`, `wradfs(1)` and `wrmsdos(1)`.

Transferring ADFS files to your workstation

ADFS^a floppy discs, as used by RISC OS, may be accessed using similar specification routines to MS-DOS floppy discs:

- `adfsals` - lists the contents of an ADFS floppy disc in a format similar to the UNIX command `ls`, and using similar options.
- `adfsacat` - copies the contents of a file or files on an ADFS floppy disc to the standard output similar to the UNIX command `cat`.
- `adfsacp` - copies a file or files on an ADFS floppy disc to a similarly-named file or files under UNIX. Directory trees on the floppy disc may be searched recursively. Files are renamed to take account of the fact that `'/'` is used as a filename separator under UNIX, where `'.'` is used under ADFS.
- `wradfs` - copies or replaces files on an ADFS floppy disc. Options are available to format and/or initialise the ADFS structure on the floppy disc. Directories may be created, and recursively copied.
- `adfsarm` - removes files and directories from an ADFS floppy disc.

Note that the above utilities can only be used to copy text files with floppy discs that have been formatted using RISC OS to the old ADFS 800*1K disc format (D format).

If you have a formatted RISC OS floppy disc that contains information you wish to transfer to your RISC iX workstation, insert the floppy disc and type:

```
adfsls
```

to check that the disc is readable.

To copy two text files `file1` and `file2`, to the `/tmp` directory on your workstation, type:

```
adfscp -V file1 file2 /tmp
```

```
Created file /tmp/file1: 46 bytes
```

```
Created file /tmp/file2: 54 bytes
```

As indicated by the system messages above (generated by specifying the `-V` option), the two files specified are copied from the floppy disc to `/tmp` on your workstation.

To check that the file was successfully copied, list the contents of the current directory, using the `ls` command.

For more information, refer to the following manual pages; `adfscat(1)`, `adfscp(1)`, `adfsls(1)`, `adfsrm(1)` and `wradfs(1)`.

In addition, you can also use floppy discs for performing backups of your workstation and also as mountable file systems. For more information, refer to the *RISC iX System Administrator's Manual*.

Command summary

The following list summarises the major commands covered in this chapter:

	Command	Use
ftp	ftp	Start file transfer program, using the following subset of commands:
	open <i>machinename</i>	Establish a connection to the machine specified.
	get <i>sourcefile destfile</i>	Copy the remote file to your machine and call it <i>destfile</i> .
	put <i>sourcefile destfile</i>	Transfer the file from your machine to a remote machine and call it <i>destfile</i> .
	help	Display a list of commands you can use at the 'ftp>' prompt
	quit	Quit ftp.
tftp	tftp	File transfer program, including the commands:
	connect <i>machinename</i>	Connect to the machine specified.
	get <i>sourcefile destfile</i>	Copy file on machine to which you are connected, transfer it to your own machine and call it <i>destfile</i> .
	put <i>sourcefile destfile</i>	Transfer a copy of a file from your machine to the existing <i>destfile</i> on the remote machine.
	?	Display a list of commands you can use at the 'tftp>' prompt.
	quit	Quit tftp.
telnet	telnet <i>machine</i>	File transfer program, including the commands:
	quit	Quit telnet program.
	help	Display a list of the commands you can use at the 'telnet>' prompt.

mail

Command	Use
mail <i>username</i>	Start the mail program.
Commands given after the mail prompt (&)	
exit	Exit the mail program without saving unread mail.
headers	Display numbered mail message list.
<i>messagenumber</i>	Display message with specified number.
print [<i>messagenumber</i>]	Display current message (message with specified number).
quit	Exit mail program.
reply [<i>messagenumber</i>]	Reply to current message (to specified message).
save [<i>messagenumber</i>] <i>filename</i>	Save current mail message (message with specified number) into specified file.
? or help	Help information.
tilde escape commands when sending mail	
~m [<i>messagenumber</i>]	Insert a copy of the current mail message (message with specified number).
~r <i>filename</i>	Insert a copy of the specified file.
~v	Enter vi text editor within mail.
~?	Display a summary of tilde escape commands.
Commands to check who is logged in	
users	Display usernames of all currently logged-in users.
who	Display usernames and system information of all users currently logged in (less detail than w).
w	Display information about all users currently logged in to your system.

	Command	Use
from	from	Display list of messages in your system mailbox, by sender.
talk	talk <i>username</i> [@ <i>machinename</i>]	Start interactive communication session with specified user (on specified machine on local network).
write	write <i>username</i> [<i>ttyname</i>]	Send message to a user on a specified terminal. Type text of message beginning on the following line and end message with <CTRL-D> on a line by itself.
wall	wall	Send broadcast message to all users of machine. Type text of message beginning on the following line and end message with <CTRL-D> on a line by itself.
Floppy disc commands	ffd	Format floppy disc.
	tar cvf /dev/fdf1024 <i>filename</i>	Copy <i>filename</i> to a formatted floppy disc.
	tar tvf /dev/fdf1024	List the contents of <i>tar</i> files on a floppy disc.
	tar xvf /dev/rfdf1024	Extract all the contents of a floppy disc and copy the contents to the current directory.
	msdosls	List the contents of an MS-DOS floppy disc.

Command	Use
msdoscp -v filename /tmp	Copy <i>filename</i> from an MS-DOS floppy disc to /tmp.
adfs1s	List the contents of an ADFS floppy disc.
adfscp -v filename /tmp	Copy <i>filename</i> from an ADFS floppy disc to /tmp.

Sources of further information

For further information about mail, refer to the *Mail Reference Manual* by Kurt Shoens in the *Berkeley 4.3BSD Unix User's Supplementary Documents*, which contains a complete description of mail.

For more information about the other commands discussed in this chapter, refer to the appropriate manual page entry for each command.

Error messages that you may receive while using these commands are outlined in the reference section *Trouble-shooting*, at the back of this guide.

Using the X Window System

Introduction

This chapter describes the **X Window System** – the industry standard windowing environment available on your workstation, which runs under a wide variety of operating systems, including RISC iX. The X Window System supplied with your RISC iX workstation is the latest release of Version 11.

Although not essential, a windowing system like this is often a very effective way for you to use all the features of your workstation, as the X Window System allows you to have multiple graphical and textual applications programs (referred to as **client programs**) running simultaneously. For example, at any one time you can be using a desktop, editing one or more files and sending electronic mail all within separate windows on one **display**.

In the X Window System (X for short), a display comprises a mouse and a keyboard plus one or more workstations. This means that if your workstation is on a network, you can be running X client programs on your host workstation and also be running other client programs on a remote machine somewhere else on the network, but showing up on your screen. Like NFS, X is **network transparent**: the requests from the host workstation, including keyboard and mouse commands, are relayed across the network to the remote workstation.

The program that acts as the intermediary between the client programs that are running in any one display and the workstation that is actually running X, is called the **X server**. The name of the X server on your RISC iX workstation is `Xarm` (short for *X Acorn RISC machine server*).

The X programming library, called `Xlib`, contains a range of C language programming tools that you can use to write client programs based on X. As X can also run under a variety of operating systems and workstations, it is said to be **machine independent**. This means that any well written client program you write using `Xlib` may be portable to many other manufacturers' workstations that are also capable of running X.

Clients provided

A set of client programs (clients for short) that have already been written using `Xlib` are available on your workstation. One of the most important clients is a **window manager** that you can use to control the size and location of the windows you create on a display.

Many such window managers have been written for X and you are supplied with several on your system. These are `wm` (a simple but primitive demo window manager), `twm` (a fairly advanced window manager) and `awm` (a window manager very similar to `twm`).

The window manager described in this chapter is `uwm` (short for *universal window manager*) which is probably the most widely used of the ones that are currently available at this time. (As X is an emerging standard, this situation may change).

Another important client is the **terminal emulator** `xterm`. As X is suitable only for bitmapped displays, you also need terminal windows to run existing text-based programs that have been written for use on standard ASCII terminals. `xterm` provides you with this facility. It emulates a DEC VT102 type of terminal.

As you can bring up more than one `xterm` window, you can have many such text-based programs running at any one time together with bitmapped, graphics-based programs such as a desktop.

Some of the clients provided with X are listed below, together with a brief description of their use. Many of the clients are for demonstration purposes, some have other uses:

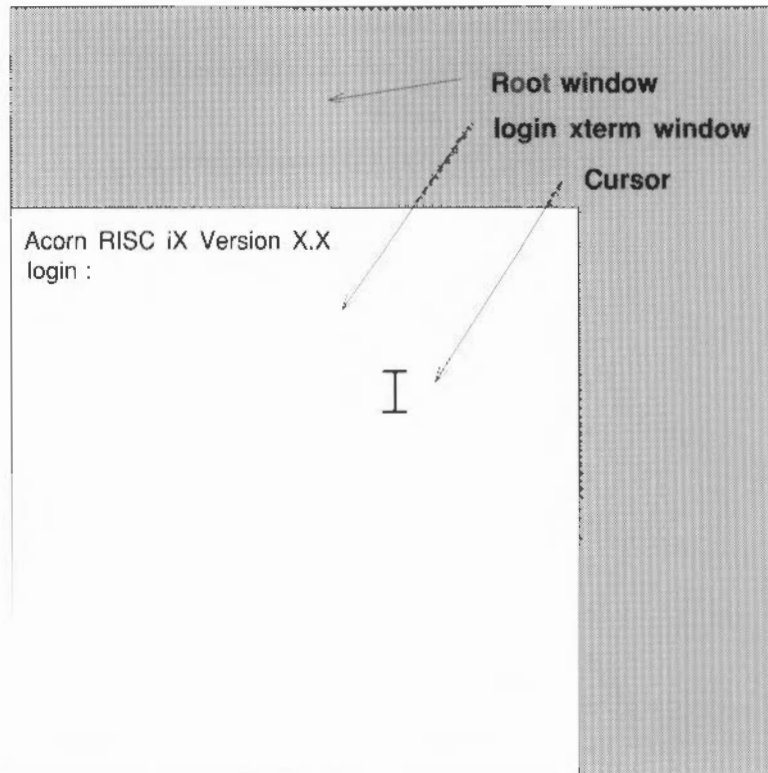
- `awm` – Ardent window manager
- `bitmap` – bitmap editor
- `ico` – animate an icosahedron or other polyhedra
- `mkruler` – make a function key ruler for output to a PostScript printer. Acorn specific.
- `muncher` – draw interesting patterns
- `plaid` – paint some plaid-like patterns
- `puzzle` – 15-puzzle game
- `showsfnf` – font examination tool

- `twm` – Tom's window manager
- `uwm` – a 'universal' window manager
- `xbiff` – US style mailbox icon, to notify user of new electronic mail
- `xcalc` – a calculator
- `xcalendar` – calendar and mini-diary
- `xclock` – a clock
- `xdpr` – print a snapshot of an X window
- `xedit` – a simple screen-based text editor
- `xfd` – font displayer
- `xhost` – server access control program
- `xlsfonts` – server font list displayer
- `xman` – a manual page browser
- `xmodmap` – keyboard configuration tool
- `xmore` – file browser
- `xperfmon` – performance monitor
- `xpr` – print an X window dump
- `xprkbd` – keyboard configuration tool
- `xprop` – display window and font properties
- `xset` – user preference utility
- `xsetroot` – root window parameter setting utility
- `xwd` – dump an image of an X window to a file
- `xwininfo` – window information utility
- `xwud` – show previously dumped window images.

For a full list of the clients provided on your workstation, look in the directory `/usr/bin/X11`. For more information about some of the clients listed above see the reference section *RISC iX manual pages*, at the back of this guide. The `uwm` window manager and the terminal emulator `xterm` are discussed in the rest of this chapter.

Starting X

If your workstation has already been configured to start up in X, you should see something similar to the following display, which indicates that X is running:



The dark background is referred to as the **root window** and the small window that appears in the bottom lefthand corner of your screen is a terminal emulator window called the **login xterm window**, which is produced by the client `xterm`. The cursor is the object shaped like an 'I'.

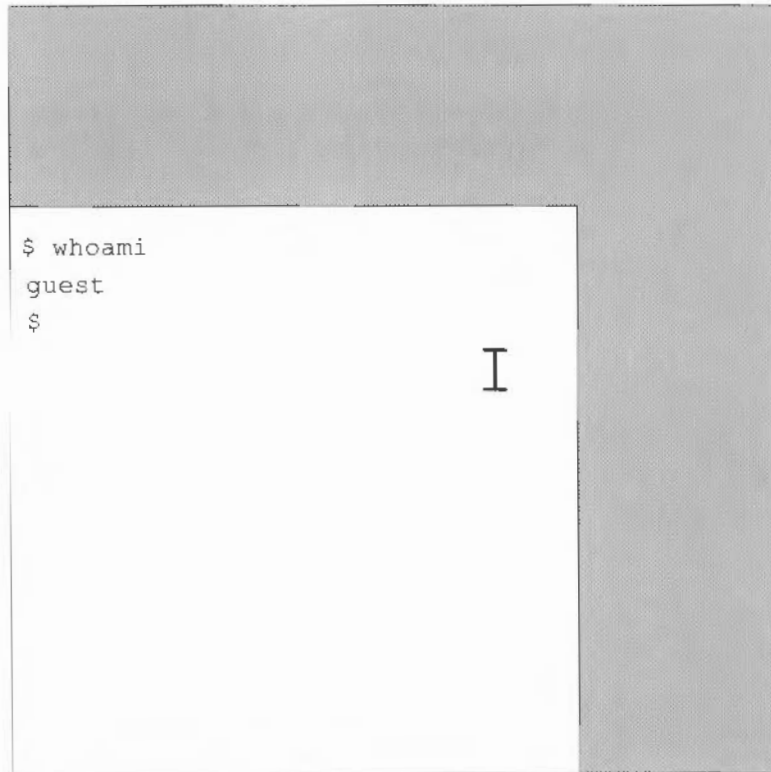
Notice that by moving the mouse between the root window and the `xterm` window the **cursor** changes shape. Also notice that you cannot type into the `xterm` window unless the cursor is positioned in that window. This is known as **focusing**. The section later on in this chapter, describing the window manager, shows how you can change the focus.

To start using the features of X, position the cursor in the `xterm` window and at the 'login:' prompt, log in as normal.

If your workstation has not been configured to start up in X, you can start X running on your workstation from the command line by typing:

xinit

This command starts X by initialising the program `xarm`, the X server for your workstation and produces the following display:



An `xterm` window is automatically created for the user who started X. In the above example, the user is `guest`.

Starting the window manager

For more information about `Xarm(1)` and `xterm(1)` and the options that you can use with them, refer to the manual page for `Xarm(1)` in the reference section *RISC iX manual pages*, at the back of this guide.

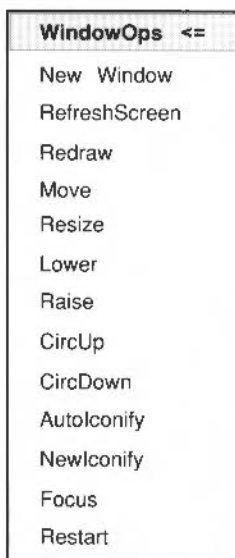
In order to be able to control and manipulate the applications that you have running in separate windows on your screen, you need to use a window manager. For example, resizing windows, closing windows to an icon and moving windows around on the screen can all be done using a window manager.

To start `uwm` running on your workstation in background mode, type:

```
uwm & ↵
```

in the terminal emulator window. Note the use of the ‘&’ metacharacter to specify that the command is to be run in background mode.

After a few seconds a *beep* will sound from your workstation. This confirms that `uwm` is now running. You can further verify that `uwm` is running by moving the mouse anywhere on the root window and clicking and holding down the middle mouse button. This should show the following menu, close to the cursor position:



Start-up files

uwm, unlike most window managers that you may have met outside of the X environment, displays as little information as possible on the screen. For example, there are no title bars with icons for closing and resizing windows. Most of the window operations are performed using menus.

uwm is a fully configurable window manager. When you issue the command to start uwm it searches in various directories in the file system for a start-up file.

The file should contain a list of settings to describe certain attributes of the window manager. For example, the functions of the mouse buttons, menu entries etc.

When uwm is invoked it follows a predefined search path to locate any start-up files, looking firstly in `/usr/lib/X11/system.uwmrc`, and then in your home directory (`$HOME`) for your personalised start-up file `.uwmrc`. If none of these two files are found, then it uses its own built-in default file.

If any of the two start-up files are found then each file is read and the settings defined are incorporated into the characteristics of the window manager. If there are any conflicting settings, then the setting in your file is used.

This means that, within certain constraints, you have the ability to customise the window manager to suit your personal preferences.

The following section assumes that none of the two start-up files are found and so only describes the characteristics of the built-in default file. For information on setting up your own personalised file, refer to the section *Running clients and applications from uwm*, later in this chapter.

The WindowOps menu

This menu, as indicated by the banner at the top, is called the **WindowOps** menu (short for *window operations*). It provides you with all the facilities for controlling the windows on your screen:

- **New Window** – create a new xterm window.
- **Refresh Screen** – redraw the entire contents of the screen.
- **Redraw** – redraw an individual window.
- **Move** – move windows and icons around.
- **Resize** – resize a window.

Selecting options from the WindowOps menu

Using 'New Window'

- `Lower` – place a window behind another.
- `Raise` – place a window in front of another.
- `CircUp` – circulate the stacking of a group of windows by raising the bottom window to the top (and lower the other windows by one level).
- `CircDown` – circulate the stacking of a group of windows by moving the top one to the bottom (and raise the other windows by one level).
- `AutoIconify` – iconify or de-iconify a window or icon and leave it in its present location on the screen.
- `NewIconify` – iconify or de-iconify a window or icon and move it to a new location.
- `Focus` – select window for keyboard input.
- `Restart` – restart the window manager (usually used after altering `$HOME/.uwmmrc`).

To choose any of the above options:

- Move the mouse anywhere on the root window and click the middle mouse button to bring up the `WindowOps` menu.
- With the mouse button still held down, select the option you want from this menu by moving the pointer down the menu until the option is highlighted; then release the mouse button.

The option `New Window` is used to create a new `xterm` window:

- Bring up the `WindowOps` menu and select the `New Window` option.
- After a few seconds a message appears in the top lefthand corner of the screen – `xterm 0x0`. This represents the new window; your mouse pointer also changes to a right-angle pointer that allows you to position the size and location of the new window.
- **To make a default size window**, move this angle pointer to where you want the top lefthand corner of your window to be and click the left mouse button. A default-size window (80x24) appears on your screen.
- **To make your own size window**, move the angle pointer to where you want the top lefthand corner of your window to appear and click and hold down the middle mouse button. The cursor fixes to the position selected and a new angle pointer appears.

With the middle mouse button still held down, move this angle pointer to the desired position for the bottom right-hand corner of the window. Notice the window size changing in the upper left corner of your screen. When you are satisfied with the position of this pointer, release the middle mouse button. The window is drawn to the size specified.

- **To make a window of maximum height**, move the angle pointer to where you want the top lefthand corner of your window to be and click the right mouse button. A default width, but maximum height window is drawn.

The above procedure is also used for loading many of the X clients, so take time to become familiar with all the above ways you can manipulate the size of the new window.

Using 'Refresh Screen'

The option `Refresh Screen` is used to redraw the entire contents of the screen; this is useful when a system message or client disrupts the screen.

- Bring up the `WindowOps` menu and select the `Refresh Screen` option.
- The screen is completely refreshed.

Using 'Redraw'

The option `Redraw` is used to redraw an individual window; this is useful when a system message or client disrupts one particular window.

- Bring up the `WindowOps` menu and select the `Redraw` option.
- The cursor changes to a circle: position it over the window to be redrawn and click a mouse button
- The window is completely refreshed.

Using 'Move'

The option `Move` is used to move windows and icons around.

- Bring up the `WindowOps` menu and select the `Move` option.
- The pointer changes to a circle, which enables you to specify the object (window or icon) on which you want to act – in this case to move the selected object around the screen.
- Select the window or icon you want to move by positioning the circle over the object and clicking and holding down the middle mouse button.
- The pointer now changes to a cross shape and an outline of the selected object appears.
- With the middle mouse button still held down, move the object outline to its new position.
- Release the middle mouse button; the object moves to the new position.

Using 'Resize'

The option `Resize` is used to resize a window:

- Bring up the `WindowOps` menu and select the `Resize` option.
- The pointer changes to a circle, position the circle near the corner or edge of the window you want to resize and click and hold down the middle mouse button.
- The pointer changes to a cross shape and an outline of the window appears.
- With the middle mouse button still held down, move the outline to its new size. Note the read-out opposite the cross, which displays the new size in pixels for graphics-based clients (or in characters for `xterm` clients).
- Release the middle mouse button; the window resizes.

Be careful when resizing windows that you are currently using for screen-based editing. Some editors or screen-based applications may not know about the new size and may be unable to re-draw the screen correctly.

Using 'Lower' and 'Raise'

Windows appearing on the screen typically overlap each other and are controlled in a stacking order by the window manager – similar to the way pieces of paper are pinned to a notice-board. To change the order of the stack so that one window is below or above another window, use the options `Lower` and `Raise` respectively.

To use `Lower`:

- Bring up the `WindowOps` menu and select the `Lower` option.
- The pointer changes to a circle, position the circle over the window you want to lower and click the middle mouse button.
- The window is placed behind all other windows, except the root window; ie placed at the bottom of the stack.

To use `Raise`:

- Bring up the `WindowOps` menu and select the `Raise` option.
- The pointer changes to a circle, position the circle over the window you want to raise and click the middle mouse button.
- The window is placed in front of all other windows; ie raised to the top of the stack.

Using 'CircUp'

In a series of overlapping windows, the bottom window in a stack can be quickly placed at the top by using `CircUp`:

- Bring up the `WindowOps` menu and select the `CircUp` option.
- The window at the bottom of the stack is placed in front of all other windows, which are lowered by one level.

Using 'CircDown'

In a series of overlapping windows, the top window in a stack can be quickly placed at the bottom by using `CircDown`:

- Bring up the `WindowOps` menu and select the `CircDown` option.
- The window at the top of the stack is placed behind all the other windows, which are raised by one level.

Using 'AutoIconify'

To create more space on your screen, you can iconify windows that you are not currently using, by using `AutoIconify`:

- Bring up the `WindowOps` menu and select the `AutoIconify` option.
- The pointer changes to a circle, position the circle over the window you want to iconify and click the middle mouse button.
- The window is iconified to where it was originally iconified or if it was never iconified before, to the current location of the pointer.

To de-iconify a window, follow the same procedure as above. The window will be displayed in its original location.

Using 'NewIconify'

`NewIconify` is similar to `AutoIconify` but also allows you to select where on the screen you want the icon to appear:

- Bring up the `WindowOps` menu and select the `NewIconify` option.
- The pointer changes to a circle, position the circle over the window you want to iconify and click the middle mouse button.
- The pointer changes to a cross shape and an outline of the window icon appears.
- With the middle mouse button still held down, move the outline to its new location.
- Release the middle mouse button; the window is iconified to the location specified.

To de-iconify and relocate a window, follow the same procedure as above. The window will be displayed in its specified location.

Using Focus

Normally, keyboard input goes to whichever window the mouse pointer is in – this is called **follow pointer** mode. Using `Focus`, you can select a window to input from the keyboard regardless of the position of the mouse pointer:

- Bring up the `WindowOps` menu and select the `Focus` option.
- The pointer changes to a circle. Position the circle over the window you want to focus on and click the middle mouse button. This window is now chosen as the **focus window**, as shown by a black border appearing around it.

To change the focus window back to follow pointer mode (the default), select the `Focus` option again and click the middle mouse button anywhere on the root window.

Using Restart

You can use the option `Restart`, to restart the window manager if it is not behaving properly. For example, after editing your `.uwmmrc` file.

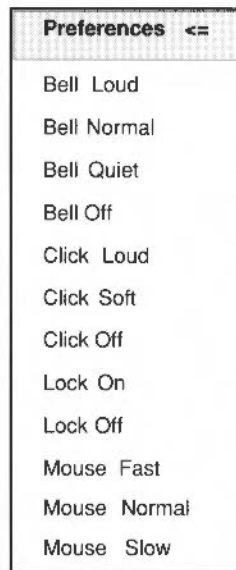
To use `Restart`:

- Bring up the `WindowOps` menu and select the `Restart` option.
- The window manager is restarted.

The Preferences menu

The Preferences menu is also available in uwm: you can use it to alter some of the characteristics of the window manager. For example, you can control the volume of the bell on your system and also the speed of the mouse.

To display the Preferences menu, hold down the <meta> key (on your RISC iX workstation, this is the <Alt> key) and <Shift> key and press the middle mouse button. This reveals the WindowOps menu. Slowly slide the arrow across the WindowOps banner at the top of the menu, until it reaches the end. The Preferences menu will then appear. You can now let go of the <Alt> and <Shift> keys:



To choose an option from this menu, bring up the menu in the way just described and while holding down the middle mouse button, choose the required option. Release the mouse button – the new setting will be used from now on by the window manager.

Keyboard short-cuts

Some of the window management actions described above can also be performed using combinations of the keyboard and mouse. The following table summarises these short-cuts:

Function	Location of pointer	Keyboard/Mouse shortcut
Move	window or icon	<Alt> right and drag
Resize	window	<Alt> middle and drag
Raise	window or icon	<Alt> right click
Lower	window or icon	<Alt> left click
CircUp	root	<Alt> right click
CircDown	root	<Alt> left click
AutoIconify	window or icon	<Alt> left down
NewIconify	window or icon	<Alt> left and drag
De-iconify and move	icon	<Alt> left and drag
WindowOps menu	anywhere	<Alt-Shift> middle down

For example, to bring up the WindowOps menu from anywhere on the screen, press <Alt> and <Shift> simultaneously and hold down the middle mouse button.

To place a window or icon at the bottom of the stack with the pointer either on the icon or window, press <Alt> and click the left mouse button.

To move a window or icon with the pointer either on the icon or window, press <Alt> and the right mouse button simultaneously and drag the object to its new location. Then release both buttons.

Using xterm to load X clients

Some of the most useful X clients can be started from an `xterm` window by just simply typing their name in an `xterm` window. For example, to start `Xman` the manual page browser client, just type the name of the client in one of the `xterm` windows that you have created, followed by an ampersand to run the command in background mode:

```
xman &
```

After a few seconds a message appears in the top lefthand corner of the screen – `xman 0x0` – this represents the window for `xman`. Your mouse pointer also changes to a right-angle pointer that allows you to position the size and location of this new window.

Move this pointer to where you want the top-lefthand corner of your window to be and click the left mouse button. The `xman` client is now ready for use.

For more information, refer to the manual page for `xman` in the reference section *RISC iX manual pages* at the back of this guide.

Using xterm to run non-X applications

As `xterm` is really just a terminal within a window, you can use it as you would any normal terminal to run standard UNIX programs. For example, to run the screen editor `vi` just type the name of the program. For example:

```
vi
```

This loads `vi` into the window where you issued the command.

As well as being able to run the above types of application, `xterm` windows can also provide you with extra window control facilities. For example, there are facilities for:

- cutting and pasting text between windows
- adding scroll-bars
- setting up the terminal – by choosing options from two menus that can be displayed in any `xterm` window.

For more information on any of the above facilities, refer to the manual page for `xterm(1)` in the reference section *RISC iX manual pages*, at the back of this guide.

Running clients and applications from uwm

All of these applications can be included in the start-up file of your window manager, to allow you to select them from a menu.

As a starting point for creating your own customised version of uwm, copy the default start-up file `/usr/lib/X11/default.uwmrc` to your home directory, using the command:

```
cp /usr/lib/X11/default.uwmrc $HOME/.uwmrc ↵
```

and change the access permissions on the file so that you can write to it, using the command `chmod(1)`. For example:

```
chmod u+w .uwmrc
```

As you become more familiar with the window manager, you will want to alter some of the settings and add new entries in this file to suit your preferences.

The uwm start-up file consists of three sections:

- a variables section
- mouse buttons section
- a menus section.

The variables section comes first in the file, followed by the mouse buttons section and finally the menus section.

The menus section contains definitions of the functions that are performed when a menu entry is selected.

Running clients

For example, to include `xman` (the manual page browser) in the start-up file, edit the menus section of the `.uwmrc` in your home directory to include the line:

```
Manual browser:      !"xman &"
```

Save the file and restart the window manager by selecting `Restart` from the `WindowOps` menu. A new entry called `Manual browser` will appear on the `WindowOps` menu when you bring up this menu again.

Select this new entry in the normal way and the client `xman` will be loaded.

Running applications

A typical non-X application that you may like to run from within your window manager is the screen editor, `vi`.

To run `vi`, enter the following line in your start-up file:

```
vi:  !"xterm -e vi &"
```

Restart the window manager as previously described. A new entry called `vi` will appear on the `WindowOps` menu when you bring up this menu again.

Select this new entry in the normal way. A window will appear, which you can size in the normal way. Once the window is sized, the editor `vi` is loaded.

There are a suite of options you can use with `xterm` to select the size of the window, name of the window and title of the window when iconified. For more information, refer to the manual page for `xterm(1)`.

For a full description of the options available for the window manager and the terminal emulator, refer to the manual page entry for `uwm(1)` and `xterm(1)` respectively.

Exiting from X

If you started X using the command `xinit`, to exit from X type your normal log out command in the login `xterm` window. For example

```
exit
```

Following this, the program `Xarm` and all running clients will be stopped, the display will blank and you will receive the following message:

```
waiting for server to terminate
```

Your normal login prompt will then be displayed. You can then carry on using your workstation.

If your workstation is configured to start up in X, just log out in the normal way. The standard login prompt will be displayed ready for another user to log in to the workstation.

Sources of further information

For more information about X and writing applications for X refer to the *X Window System Manual Set* produced by *IXI Limited*, which comes in four volumes:

- *Volume 1: C Language Interface*
- *Volume 2: Reference Manual*
- *Volume 3: Standard Supplement*
- *Volume 4: Server & Porting Guide.*

Also refer to the *Definitive Guide to the X Window System*, produced by *O'Reilly & Associates Inc.*, which similarly comes in four volumes:

- *Volume 1: Xlib Programming Manual*
- *Volume 2: Xlib Reference Manual*
- *Volume 3: X Window System User's Guide*
- *Volume 4: X Toolkit Programmer's Guide.*

Further uses of RISC iX

Introduction

This chapter introduces some of the more advanced features of the RISC iX operating system. A full explanation is beyond the scope of this guide.

Each feature is briefly described and useful sources of further information are referenced for each feature.

The advanced features are categorised as follows:

- general programming utilities
- text preparation
- data manipulation
- miscellaneous utilities.

General programming utilities

This section lists some of the utilities RISC iX provides to help you develop programs. For further details refer to the relevant manual page by looking in the *Berkeley 4.3 UNIX User's Reference Manual* or by using the `man` command. Additional references for further information are also given below.

Command	Use
<code>cb(1)</code>	'C beautifier' – lay out a C program, adding spacing and indentation as appropriate.
<code>lint(1V)</code>	Verify a C program, performing stricter type-checking than the compiler, and look for features that are likely to be bugs, non-portable or wasteful.
<code>cc(1)</code>	RISC iX C compiler. To find out the basics about C, try the C course in <code>learn(1)</code> . For more advanced information, refer to the <i>RISC iX Programmer's Reference Manual</i> .
<code>ld(1)</code>	Link editor.
<code>make(1)</code>	Maintain a program group, using a list giving the inter-dependence of its constituent files.
<code>adb(1)</code>	Debugger for general purpose use.
<code>dbx(1)</code>	Source level symbolic debugger.
<code>as(1)</code>	RISC iX ARM assembler. See the <i>RISC iX Programmer's Reference Manual</i> .
<code>yacc(1)</code>	Parser generator. Can be used in conjunction with <code>lex</code> .
<code>lex(1)</code>	Lexical analyser. Can be used in conjunction with <code>yacc</code> .

Text preparation

This section lists some of the utilities RISC iX provides to help you prepare and format text and documents. For further details refer to the relevant manual page by looking in the *Berkeley 4.3 UNIX User's Reference Manual* or by using the `man` command. Additional references for further information are also given below.

Command	Use
<code>diction(1)</code>	Look for poor or verbose sentences in a document.
<code>explain(1)</code>	Use with the output from <code>diction</code> as an interactive thesaurus.
<code>spell(1)</code>	Find spelling errors in a document.
<code>style(1)</code>	Give indications of the style and readability of a document.
<code>wc(1)</code>	Count lines, words and characters in a document.
<code>nroff(1)</code>	Format a document for output to a printer. For more information, try the <code>macros</code> course in <code>learn(1)</code> .
<code>troff(1)</code>	Format a document for output to a photo-typesetter.
<code>psroff(1)</code>	Convert a <code>troff</code> document for PostScript output.
<code>deroff(1)</code>	Remove <code>nroff</code> and <code>troff</code> formatting commands from a document.
<code>refer(1)</code>	Find and format references for footnotes or endnotes. Used as a preprocessor for <code>nroff</code> or <code>troff</code> .
<code>tbl(1)</code>	Format tables. Used as a preprocessor for <code>nroff</code> or <code>troff</code> .
<code>eqn(1)</code>	Format mathematical equations. Used as a preprocessor for <code>troff</code> . For more information, try the <code>eqn</code> course in <code>learn(1)</code> .

<code>neqn(1)</code>	Format mathematical equations. Used as a preprocessor for <code>nroff</code> .
<code>checkeq(1)</code>	Check the validity of equations prior to formatting.
<code>col(1)</code>	Filter reverse line feeds and other codes output by <code>nroff</code> , typically when it is used with <code>tbl</code> .
<code>colcrt(1)</code>	Filter <code>nroff</code> output for display to a terminal.

Data manipulation

This section lists some of the utilities RISC iX provides to help you manipulate data. For further details refer to the relevant manual page by looking in the *Berkeley 4.3 UNIX User's Reference Manual* or by using the `man` command. Additional references for further information are also given below.

Command	Use
<code>head(1)</code>	Output lines from the head (start) of a file.
<code>tail(1)</code>	Output lines from the tail (end) of a file.
<code>split(1)</code>	Split a file into separate parts.
<code>expand(1)</code>	Expand tabs to spaces.
<code>unexpand(1)</code>	Convert spaces to tabs.
<code>pr(1)</code>	Print text, optionally multi-columned or with a header.
<code>rev(1)</code>	Reverse the order of lines of text
<code>sort(1)</code>	Sort files.
<code>look(1)</code>	Find lines in a sorted file starting with a given string.
<code>grep(1)</code>	Find lines in a file containing a given regular expression.
<code>awk(1)</code>	Find lines of text matching a pattern and perform a corresponding action. Many pattern-action pairs can be specified and programs written.
<code>sed(1)</code>	Stream editor, using commands similar to those used by <code>ed</code> .
<code>tr(1)</code>	Translate all occurrences of a given string of characters to another given string.

Miscellaneous utilities

This section lists some of the more useful of the many other utilities RISC iX provides. For further details refer to the relevant manual page by looking in the *Berkeley 4.3 UNIX User's Reference Manual* or by using the `man` command. Additional references for further information are also given below.

Command	Use
<code>find(1)</code>	Find files.
<code>cmp(1)</code>	Compare two files and find first differing byte.
<code>diff(1)</code>	List differences between two text files.
<code>diff3(1)</code>	List differences between three text files.
<code>comm(1)</code>	List lines common to two files and show which of the two files the remaining lines are in.
<code>uniq(1)</code>	Remove (or report on) repeated lines in a file. The lines must also be adjacent.
<code>od(1)</code>	Octal, decimal, hex or ASCII dump.
<code>sum(1)</code>	Calculate a checksum for a file and its size in blocks.
<code>at(1)</code>	Execute commands at a later time.
<code>calendar(1)</code>	Display lines from a calendar file which contain today's or tomorrow's date.
<code>cal(1)</code>	Display the calendar for a given month.
<code>bc(1)</code>	Do sums interactively – useful both as a simple calculator and an arbitrary-precision arithmetic language.

Bibliography

This appendix lists the titles of all the manuals referenced throughout this guide, as pointers to further sources of information about the features of your system.

1. *R140 Operations Guide* – (Part No. 0483,710)
2. *RISC iX System Administrator's Manual* – (Part No. 0483,747)
3. *RISC iX Programmer's Reference Manual* – (Part No. 0483,748)
4. Berkeley 4.3 BSD System Manual Set, which comprises seven volumes:
 - *User's Reference Manual (URM)*
 - *User's Supplementary Documents (USD)*
 - *Programmer's Reference Manual (PRM)*
 - *Programmer's Supplementary Documents, Volume 1 (PS1)*
 - *Programmer's Supplementary Documents, Volume 2 (PS2)*
 - *System Manager's Manual (SMM)*
 - *UNIX User's Manual Master Index*
5. *X Window System Manual Set* produced by *IXI Limited*, which comprises four volumes:
 - *Volume 1: C Language Interface*
 - *Volume 2: Reference Manual*
 - *Volume 3: Standard Supplement*
 - *Volume 4: Server & Porting Guide*

6. *The Definitive Guide to the X Window System*, produced by O'Reilly & Associates Inc., which comprises four volumes:
 - *Volume 1: Xlib Programming Manual*
 - *Volume 2: Xlib Reference Manual*
 - *Volume 3: X Window System User's Guide*
 - *Volume 4: X Toolkit Programmer's Guide*
7. *Fundamentals of Operating Systems* – A M Lister, Macmillan, (1981)

Reference Section A: Trouble-shooting

Introduction

This reference section provides you with helpful information for you to refer to whenever you get into difficulty using your system. It is structured so that you can quickly access the help information on any given topic.

- command errors – gives guidelines for typing RISC iX commands.
- editing errors – the most likely error messages that you may receive when using the editors described in this guide.
- networking problems – this section describes the typical problems that can occur if your machine is on a network.

RISC iX commands

This section gives a few simple hints on how to tackle an error message you received when typing a RISC iX command. The vast majority of errors in RISC iX are the user's mistake, not the system's, and you must learn to recognise what has caused problems. The examples assume you are trying to use the command:

```
ls -x
```

Typing errors

The most common form of mistake is a simple typing error. For example:

```
la -x
```

you typed `la` instead of `ls` so RISC iX looks for the command `la`, but cannot find it.

So, always **check your typing**. If you made a mistake, retype the line.

Syntax and punctuation

Another very common mistake is poor syntax or punctuation. For example:

```
ls-x
```

a space is missing so RISC iX looks for but cannot find the command `ls-x`

```
ls - x
```

there are too many spaces, so RISC iX executes the command `ls` on the file `x`, with no options (nothing directly follows the `-`).

Next, you should **check your punctuation**. Refer to the appropriate manual page of the command if necessary. You may find the on-line version shows punctuation rather more clearly than the printed version, because the typesetting process can make any spaces appear very small.

Special characters

Another problem that commonly occurs is the incorrect use of characters that have a special meaning to the shell when used individually or in combination with other characters. These are:

*	matches any character(s), including none
?	matches any single character

[...]	matches any of the characters enclosed by the square brackets
&	executes commands in the background
;	sequentially executes several commands typed in on one line, each separated by ;
\	turns off the meaning of special characters in the shell
''	used to quote a string of characters, some of which may be special
>	redirect output
>>	append output
<	redirect input
	redirect standard output to standard input
(...)	used for command grouping.

Look at your command and **check for special characters**. If you used any in your command, you can see how they were interpreted by the command:

```
echo yourcommand
```

You may then need to either quote the special character to stop it being interpreted, or type the command more specifically – ie without using any shell abbreviations.

If you can still see nothing wrong with what you typed, but the command is still not doing what you think it should, your next step is to read the appropriate section of this guide where the command is discussed again.

If you still are having problems, remember that the manual pages for RISC iX commands are available on-line using the `man` command. Carefully read **all** the pages for the command you are trying to use.

You may find that the command doesn't do what you thought it did, or its syntax differs from what you used. Find out how the command really works. If it doesn't do what you want, start looking through the manual pages for the commands that are closely related – these are listed under the heading **SEE ALSO**.

The manuals

Asking for help

As you get more experienced at using RISC iX you will find that you sort out your own problems more quickly and easily. But if you're still stuck now, there's probably not a lot more you can do alone. You will need to **ask for help** – try either a more experienced RISC iX user or your system administrator.

Getting information

If they can only help you later, they will need to know exactly what the problem was so they can reproduce it. You can help them immensely if you use these commands. Either note down what is displayed, or redirect the output to files:

<code>ls -al</code>	list all files in long format, including permissions.
<code>pwd</code>	print your current working directory.
<code>set</code>	show your environment.
<code>whoami</code>	show who you are currently logged on as.
<code>history</code>	list your most recent commands (only if you are in a C shell).
<code>ps</code>	display a list of the processes that you are running.

You may also find that the output from one of these commands helps you to see what the problem is.

Editing errors

The line editor – ed

This section describes the problems that can occur while you are using the editors `ed` and `vi`:

Error messages

There is only one error message that you can receive from `ed`:

```
?
```

It is left up to you to decide what you have done wrong. But here are a list of a few of the most likely actions that could have caused the error:

- you tried to quit from `ed` without saving the changes you made to the file you were editing. Either save the changes and then quit or issue the quit command (`q`) again. This time `ed` will let you quit without saving the changes.
- you issued a command that `ed` does not understand. Check the syntax of the command and look in the guide to see if you are using the command correctly.
- you requested `ed` to search for a string in a file and `ed` did not find it. Check the syntax of the search command.

You may also receive the error:

```
?filename
```

This means that you invoked `ed` with the file `filename`, but `ed` could not find this file. Quit from the editor and check that the file exists and that you spelled it correctly.

Common mistakes

You type a valid `ed` command, but nothing happens – check that `ed` is not in input mode by typing:

```
. ↵
```

then retype the command again.

Avoiding complete catastrophe

Remember, if you do something in `ed` that ruins your file, you can retrieve the situation by typing `u`, with `ed` in command mode. This undoes the last command you issued.

Error messages

Most of the error messages that you receive from `vi` are self-explanatory; for example:

<code>pattern not found</code>	pattern to search for not found.
<code>no more files to edit</code>	tried to read a new file into the buffer without specifying any more files.

In some cases the error message is displayed along with a suggested remedy:

<code>No write since last change (:quit! overrides)</code>	tried to leave <code>vi</code> without saving the changes.
--	--

<code>No write since last change (:next! overrides)</code>	tried to read in a new file without saving the changes to the existing file.
--	--

<code>File exists - use "w! newfile" to overwrite</code>	tried to overwrite an existing file.
--	--------------------------------------

The most common error message that you receive is a *beep*: this tells you that you are doing something incorrectly. Type `<ESC>` to put `vi` into command mode and try the operation again.

Common mistakes

You type a valid `vi` command, but nothing happens – check that `vi` is not in insert mode by typing `<ESC>`, then try the command again.

Avoiding complete catastrophe

Remember, if you do something in `vi` that ruins your file, you can retrieve the situation by typing `u`, with `vi` in command mode. This undoes the last command you issued.

Networking problems

This section describes the common errors that you will encounter when using the network.

Error message	Meaning
Connection refused	Remote workstation functioning, but its daemons are not ready to complete the connection .
Connection timed out	Either your workstation or the remote workstation is down, off or hung: there may be problems with ethernet, or the workstation is just heavily loaded.
File not found	The file cannot be located on the remote workstation.
Host name for your address unknown	Remote workstation has no record of your workstation's name in its <code>/etc/hosts</code> file.
Login incorrect	Your username needs to be added to the <code>/etc/passwd</code> file on the remote workstation, or you typed in your password incorrectly.
Network is unreachable	A gateway or other network connection is not working.
...No such file or directory	The file or directory on the remote workstation has been wrongly described, does not exist, or you do not have the necessary access rights to it.
...not found	The directory on the remote workstation has been wrongly described, or does not exist.

Permission denied	There is no record of your workstation name on the <code>/etc/hosts.equiv</code> or <code>.rhosts</code> file. Try logging in with a password.
RPC: Port map failure	Daemon not functioning correctly.
RPC: Timed out	The remote workstation is not running or is very heavily loaded. Although this is a very common error, it can often be ignored.
RPC: Unknown host	The name you have given for the remote workstation, <i>rhost</i> , is not known on the network. Make sure you entered the correct <i>rhost</i> name.
unknown host	The name of the remote workstation should be added to your <code>/etc/hosts</code> file.

For a more complete description of errors that you can encounter on the network, refer to the *RISC iX System Administrator's Manual*.

Reference Section B: Command summaries

Introduction

This reference section contains a summary of the commands that can be used from your RISC iX workstation, according to the following categories:

- User commands
- UNIX shell
- UNIX editors:
 - the line editor – `ed`
 - the screen editor – `vi`
 - `ex`
- mail commands
- floppy disc commands.

User commands

This section contains a list of the commands that you can use on your RISC iX workstation. The commands are listed in alphabetical order along with a brief sentence describing their function.

The commands in bold type are new commands specific to RISC iX. Commands in italics are specific to the X Window System.

Command	Description
<i>adb</i>	Debugger for general purpose use.
adfscat	Write files from an ADFS format floppy disc to standard output..
adfscp	Copy files from an ADFS format floppy disc.
adfsls	List files on an ADFS format floppy disc.
adfsrm	Remove files from an ADFS format floppy disc.
<i>alert</i>	Display an alert box.
<i>apropos</i>	Locate commands by keyword look-up.
<i>as</i>	RISC iX assembler for the ARM.
<i>at</i>	Execute commands at a later time.
<i>atq</i>	Print the queue of jobs waiting to be run.
<i>atrm</i>	Remove jobs spooled by <i>at</i> .
<i>awk</i>	Pattern scanning and processing language.
<i>awm</i>	<i>Ardent</i> window manager.
<i>bc</i>	Arbitrary-precision arithmetic language.
<i>bitmap</i>	Bitmap editor.
<i>cal</i>	Display the calendar for a given month.
<i>calendar</i>	Display lines from a calendar file.
<i>cat</i>	Catenate and print

<code>cb</code>	'C beautifier' – lay out a C program.
<code>cc</code>	RISC iX C compiler.
<code>cd</code>	Change working directory.
<code>checkeq</code>	Check the validity of equations.
<code>chgrp</code>	Change file group.
<code>chmod</code>	Change file mode.
<code>clear</code>	Clear terminal screen.
<code>cmp</code>	Compare two files.
<code>col</code>	Filter reverse line feeds.
<code>colcrt</code>	Filter <code>nroff</code> output.
<code>comm</code>	List lines common to two files.
<code>compress</code>	Compress and expand data.
<code>cp</code>	Copy files.
<code>csh</code>	A shell (command interpreter) with C-like syntax.
<code>date</code>	Print and set the date.
<code>dbx</code>	Source-level symbolic debugger.
<code>dc</code>	Desk calculator.
<code>deroff</code>	Remove <code>nroff</code> and <code>troff</code> formatting commands.
<code>df</code>	Summarise the free space on a disc.
<code>diction</code>	Look for poor or verbose sentences .
<code>diff</code>	List differences between two text files.
<code>diff3</code>	List differences between three text files.
<code>du</code>	Summarise disc usage.
<code>echo</code>	Echo arguments

ed	Line-based editor.
eqn	Format mathematical equations for <code>troff</code> .
expand	Expand tabs to spaces.
explain	Interactive thesaurus for <code>diction</code> .
fc	Font compiler.
file	Determine file type.
find	Find files.
ffd	Format floppy discs.
flpop	Specify parameters on variable floppy disc controller devices.
grep	Find lines in a file containing a given regular expression.
gs	Get string.
head	Output lines from the head of a file.
hostid	Set or print identifier of current host system.
hostname	Set or print name of current system.
ico	Animate an icosahedron or other polyhedra.
inituser	Set up a new user.
install	Install binary files.
kill	Terminate a process with <i>extreme</i> prejudice.
ld	Link editor.
learn	Computer aided instruction about RISC iX.
lex	Lexical analyser.
lint	Verify a C program.
ln	Make links to files.
login	Sign on

<code>look</code>	Find lines in a sorted file starting with a given string.
<code>lpq</code>	Spool queue examination program.
<code>lpr</code>	Send job to printer.
<code>lprm</code>	Remove jobs from the line printer spool queue.
<code>ls</code>	List contents of directory.
<code>mail</code>	Send and receive mail.
<code>make</code>	Maintain a program group.
<code>man</code>	Find manual information by keywords; print out the manual.
<code>mkdir</code>	Make a directory.
<code>mkruler</code>	Make a function key ruler for output to a PostScript printer.
<code>more</code>	File perusal filter for crt viewing.
<code>msdoscat</code>	Write files from MS-DOS format floppy disc to standard output.
<code>msdoscp</code>	Copy files from an MS-DOS format floppy disc.
<code>msdosls</code>	List files from an MS-DOS format floppy disc.
<code>msdosrm</code>	Remove files from an MS-DOS format floppy disc.
<code>muncher</code>	Draw interesting patterns.
<code>mv</code>	Move or rename files.
<code>neqn</code>	Format mathematical equations for <code>nroff</code> .
<code>nice</code>	Run a command at low priority.
<code>nroff</code>	Format a document for output to a printer.
<code>od</code>	Octal, decimal, hex or ASCII dump.
<code>passwd</code>	Change password file information

<i>plaid</i>	Paint some plaid-like patterns.
<i>pr</i>	Print text, optionally multi-columned or with a header.
<i>printenv</i>	Print out the environment.
<i>ps</i>	Display process status.
<i>psroff</i>	Convert a <i>troff</i> document for PostScript output.
<i>puzzle</i>	15-puzzle game.
<i>pwd</i>	Display working directory name.
<i>rccp</i>	Remote file copy.
<i>reborder</i>	Put borders around an X window.
<i>refer</i>	Find and format references.
<i>rev</i>	Reverse the order of lines of text.
<i>rlogin</i>	Remote login.
<i>rm</i>	Remove (unlink) files or directories.
<i>rmdir</i>	Remove (unlink) directories.
<i>rsh</i>	Remote shell.
<i>script</i>	Make typescript of terminal session.
<i>sed</i>	Stream editor.
<i>sh</i>	Command interpreter.
<i>showsnf</i>	Font examination tool.
<i>sleep</i>	Suspend execution for an interval.
<i>sort</i>	Sort or merge files.
<i>spell</i>	Find spelling errors.
<i>split</i>	Split a file into separate parts.
<i>stty</i>	Set terminal options

<code>style</code>	Give indications of the style and readability of a document.
<code>su</code>	Substitute user id temporarily.
<code>sum</code>	Calculate a checksum for a file, and its size in blocks.
<code>tail</code>	Output lines from the tail (end) of a file.
<code>tbl</code>	Format tables.
<code>tar</code>	Tape archiver.
<code>tee</code>	Pipe fitting.
<code>tftp</code>	Trivial file transfer program.
<code>time</code>	Time a command.
<code>touch</code>	Update date last modified of a file.
<code>tr</code>	Translate characters.
<code>troff</code>	Format a document for output to a photo-typesetter.
<code>tset</code>	Terminal dependent initialisation.
<code>twm</code>	<i>Tom's</i> window manager.
<code>uemacs</code>	Display-based programmable editor.
<code>unexpand</code>	Convert spaces to tabs.
<code>uniq</code>	Remove (or report on) repeated lines in a file.
<code>unit</code>	Conversion program.
<code>uucp</code>	UNIX to UNIX copy.
<code>uux</code>	UNIX to UNIX command execution.
<code>vi</code>	Screen-oriented (visual) display editor based on <code>ex</code> .
<code>wc</code>	Count lines, words and characters.
<code>whatis</code>	Describe what a command is.
<code>whereis</code>	Locate source, binary and or manual for a program

<i>which</i>	Locate a program file including aliases and paths.
<i>whoami</i>	Print effective current user id.
<i>wm</i>	A simple window manager.
wradfs	Write files onto an ADFS format floppy disc.
wrmsdos	Write files onto an MS-DOS format floppy disc.
<i>xbiff</i>	Mailbox flag.
<i>xcalc</i>	A calculator.
<i>xcalendar</i>	Calendar and mini-diary.
<i>xclock</i>	A clock.
<i>xdpr</i>	Print a snapshot of an X window.
<i>xedit</i>	A simple screen-based editor.
<i>xfd</i>	Font displayer.
<i>xhost</i>	Server access control program.
<i>xload</i>	Load average display.
<i>xmore</i>	File browser.
<i>xperfmon</i>	Performance monitor.
<i>xpr</i>	Print an X window dump.
<i>xprkbd</i>	Keyboard configuration tool.
<i>xprop</i>	Display window and font properties.
<i>xrdb</i>	X server resource database utility.
<i>xrefresh</i>	Refresh all or part of an X screen.
<i>xset</i>	User preference utility.
<i>xsetroot</i>	Root window parameter setting utility
<i>xwd</i>	Dump an image of an X window to a file.

<i>xwininfo</i>	Window information utility.
<i>xwud</i>	Show previously dumped window images.
<i>yorn</i>	Alert box.
<i>yacc</i>	Parser generator.

The UNIX shell

This section contains a quick reference to the features of the UNIX shell that have been discussed in this guide.

Special characters

*	Matches any character(s), including none.
?	Matches any single character.
[. . .]	Matches any of the characters enclosed by the square brackets.
&	Executes commands in the background.
;	Sequentially executes several commands typed in on one line, each separated by ;.
\	Turns off the meaning of special characters in the shell.
''	Quote a string.

Redirecting input and output

>	Redirect output.
>>	Append output.
<	Redirect input.
	Redirect standard output to standard input.

Shell variables

HOME	The name of your home directory; also the default directory that is used when the <code>cd</code> command is issued with no arguments.
PATH	The search path that is followed when the shell tries to execute a command that has been issued.

The line editor, ed

The following ed commands are described in the chapter *Text editing*. Further commands are available: see

- *A Tutorial Introduction to the UNIX Text Editor* by Brian W Kernighan in the *Berkeley 4.3 UNIX User's Supplementary Documents Manual*.
- *Advanced editing on UNIX* by Brian W Kernighan in the *Berkeley 4.3 UNIX User's Supplementary Documents Manual*.
- `ed(1)` manual page.

Command	Use
<code>i</code>	Insert text before the current line.
<code>a</code>	Append text after the current line.
<code>.</code>	Finish adding text.
<code>d</code>	Delete the current line.
<code>p</code>	Print the current line.
<code>↓</code>	Print the next line.
<code>-</code>	Print the previous line.
<code>m</code>	Move lines.
<code>s</code>	Substitute text on the current line.
<code>/pattern/</code>	Search forwards for <i>pattern</i> .
<code>?pattern?</code>	Search backwards for <i>pattern</i> .
<code>//</code>	Repeat a context search in the forward direction.
<code>??</code>	Repeat a context search in the backward direction.
<code>g</code>	Global modifier: make a command affect all lines.
<code>e filename</code>	Edit <i>filename</i> .
<code>w filename</code>	Write out the changes to <i>filename</i> .
<code>q</code>	Quit; a second <code>q</code> bypasses checking.
<code>u</code>	Undo the last change made.

The screen editor, vi

The following vi commands are described in the chapter *Text editing*. Further commands are available: see

- *An Introduction to Display Editing with vi* by William Joy & Mark Horton in the *Berkeley 4.3 UNIX User's Supplementary Documents Manual*.
- vi(1) manual page.

Command	Use
h	Move left by one character on the screen.
j	Move down by one line on the screen.
k	Move up by one line on the screen.
l	Move right by one character on the screen.
\$	Move to the last character on the current line.
0	Move to the first character on the current line.
w	Move forward to the beginning of the next word.
b	Move back to the beginning of the previous word.
e	Move forward to the end of the current word.
W	Move forward to beginning of next word, ignoring punctuation.
B	Move back to beginning of previous word, ignoring punctuation.
E	Move forward to end of current word, ignoring punctuation.
H	Move to the home or top line on the screen.
M	Move to the middle line on the screen.
L	Move to the last line on the screen.
<CTRL-D>	Scroll down by half a screenful.
<CTRL-U>	Scroll up by half a screenful.
<CTRL-F>	Page forward by a screenful.
<CTRL-B>	Page backward by a screenful.
<CTRL-E>	Expose one more line at the bottom of the screen.
<CTRL-Y>	Yank another line onto the top of the screen.
)	Move the cursor to the end of the current sentence.
(Move the cursor to the start of the current sentence.
}	Move the cursor to the end of the current paragraph.
{	Move the cursor to the start of the current paragraph.
/pattern	Search forwards for pattern.

?pattern	search backwards for pattern.
n	repeat a context search in the same direction.
N	repeat a context search in the opposite direction.
a	append text after the current cursor position.
i	insert text before the current cursor position.
o	open file to append text after the current line.
A	append text at the end of the current line.
O	open file to insert text before the current line.
s	substitute a string of characters for the cursor character.
cw	change a word.
cc	change an entire line.
x	delete the current character.
dd	delete the current line.
dw	delete the current word.
Y	yank lines into a buffer.
P	put back the deleted text after the current cursor position or current line.
P	put back the deleted text before the current cursor position or current line.
u	undo the last change made.
U	undo the last set of changes made to the current line.
.	repeat the last buffer change command.
J	join together the current line with the line below.

ex commands

The following `ex` commands are described in the chapter *Text editing*. They may be used from both `vi` and `ex`. Further commands are available: see

- *An Introduction to Display Editing with vi* by William Joy & Mark Horton in the *Berkeley 4.3 UNIX User's Supplementary Documents Manual*.
- *The Ex Reference Manual - Version 3.7* by William Joy & Mark Horton in the *Berkeley 4.3 UNIX User's Supplementary Documents Manual*.
- `vi(1)` and `ex(1)` manual pages.

Command	Use
<code>:w filename</code>	Write out the changes to <i>filename</i> .
<code>:q</code>	Quit. <code>:q!</code> bypasses checking.
<code>:e</code>	Edit a new file. <code>:e!</code> bypasses checking.
<code>:g</code>	Globally search for a string.
<code>:s</code>	Substitute one string for another.
<code>!: cmd</code>	Execute the shell command, <i>cmd</i> and return to <code>vi</code> .
<code>:sh</code>	Execute a shell.
<code>:n</code>	Edit next file in argument list. <code>:n!</code> bypasses checking.
<code>:set</code>	Print or set options.
<code>:map</code>	Define a macro command.

mail commands

The following list summarises the mail commands discussed in this guide:

Command	Syntax	Use
mail	mail <i>username</i>	start the mail program
Commands given after the mail prompt (&)		
headers		display numbered mail message list.
	<i>messagenumber</i>	display message with specified number.
print	[<i>messagenumber</i>]	display current message (or message with specified number).
quit		exit mail program.
reply	[<i>messagenumber</i>]	reply to current message (or to message with specified number).
save	[<i>messagenumber</i>] <i>filename</i>	save current mail message (or message with specified number) into specified file.
? or help		help information.
x		exit the mail program without saving unread mail.
Tilde escape commands		
~m	[<i>messagenumber</i>]	insert a copy of the current mail message (or message with specified number).
~r	<i>filename</i>	insert a copy of the specified file.
~v		enter vi text editor within mail.
~?		display a summary of tilde escape commands.

Floppy disc commands

The following list summarises the floppy disc commands discussed in this guide:

Command	Use
ffd	format floppy disc.
tar cvf /dev/fdf1024 <i>filename</i>	copy <i>filename</i> to a formatted floppy disc.
tar tvf /dev/fdf1024	list the contents of <i>tar</i> files on a floppy disc.
tar xvf /dev/rfdf1024	extract all the contents of a floppy disc and copy the contents to the current directory.
msdosls	list the contents of an MS-DOS floppy disc.
msdoscp -v <i>filename</i> /tmp	copy <i>filename</i> from an MS-DOS floppy disc to /tmp.
adfsls	list the contents of an ADFS floppy disc.
adfscp -v <i>filename</i> /tmp	copy <i>filename</i> from an ADFS floppy disc to /tmp.

Reference Section C: RISC iX manual pages

This reference section contains a selection of user reference manual pages for use with your RISC iX workstation. There are manual pages for most of the commands covered in this guide along with manual pages that have been significantly changed from the Berkeley 4.3 BSD originals to suit RISC iX.

NAME

adfscat – catenate and print from adfs micro diskettes.

SYNOPSIS

adfscat file . . .

DESCRIPTION

Adfscat writes to standard output the named file or files on an adfs micro diskette. The character '.' should be used to delimit subdirectory names.

FILES

/dev/rfd1024

SEE ALSO

adfsis(1), *wradfs(1)*, *adfsrm(1)*, *adfscp(1)*, *msdoscat(1)*, *msdosls(1)*, *wrmsdos(1)*, *msdosrm(1)*, *msdoscp(1)*, *fd(4)*.

NAME

adfscp - copy files from adfs micro diskettes.

SYNOPSIS

adfscp [**-RFV**] [**-Dchar**] file . . . directory

DESCRIPTION

Adfscp copies the specified file or files from the **adfs** diskette to the specified UNIX directory. The UNIX directory may be a "." to denote the current directory.

A "." or a directory name may be substituted for an **adfs** file name to denote all the files in the root or the specified directory name. However subdirectories of the specified directory are not copied unless the **-R** option is given.

If an existing UNIX file would be overwritten, confirmation is requested unless the **-F** option is given.

The **-V** option causes *adfscp* to give a blow-by-blow account of its activities.

The **-m** option causes *adfscp* to set the modification and access dates on files created from ADFS files which have them.

Slashes in the **adfs** file names are replaced by fullstops in the resulting UNIX file names unless the **-Dchar** option is given, which causes *adfscp* to look for the given character in the **adfs** file names and replace those by fullstops. (I.e. if the same **-D** option is given to *adfscp* as was given to *wradfs*, the resulting UNIX file names should be the same as before apart from truncation to 10 characters).

FILES

/dev/rfd1024

SEE ALSO

adfscat(1), *adfs*(1), *adfsrm*, *wradfs*(1), *msdoscat*(1), *msdosls*(1), *wrmsdos*(1), *msdosrm*(1), *msdoscp*(1), *fd*(4).

NAME

`adfsls` – list files on adfs micro diskettes.

SYNOPSIS

`adfsls` [`-CFRdl`] [`file`]

DESCRIPTION

Adfsls produces output on the standard output relating to an **adfs** micro diskette in a similar fashion to the UNIX utility *ls*(1).

By default (with no arguments), the root level directory is examined and the entries therein printed in a single column, sorted alphabetically.

One or more filenames (as with *adfs**cat*) may be given. Files are just printed, directories are listed out, all in the given order.

The `-C` option causes the entries to be listed in multiple columns.

The `-F` option causes a `.` to be appended to subdirectory names in the output.

The `-R` option causes subdirectories to be recursively examined.

The `-d` option causes subdirectory names to be listed instead of the contents being printed.

The `-l` option causes information in addition to the file names to be output, in particular the access bits, followed by either the load and execution addresses or the file type and date/time, the sectors (which are reckoned as 256 bytes in length, although the disc blocksize is 1024 bytes) used by the file and the file size.

The `-t` option causes the files to appear in date/time order, newest first, instead of alphabetic order. If files have a load and execution address rather than a date and time, then they are treated as very recent, i.e. they appear before all other files.

The `-r` option causes the files to be displayed in reverse order, i.e. in reverse alphabetic order without and in newest-last order with the `-t` option, in which case files with a load and execution address will appear last.

FILES

`/dev/rfd1024`

SEE ALSO

*adfs**cat*(1), *adfs**cp*(1), *wradfs*(1), *adfs**rm*(1), *msdos**cat*(1), *msdos**ls*(1), *wrmsdos*(1), *msdos**rm*(1), *msdos**cp*(1), *fd*(4).

NAME

`adfsrm` - remove files from adfs micro diskettes.

SYNOPSIS

`adfsrm [-f] [-r] file . . .`

DESCRIPTION

Adfsrm removes one or more named files or directories from an **adfs** diskette. File and directory names are used in the same fashion as with the other utilities, either fullstops or slashes being used as directory delimiters.

Directories are not removed unless the `-r` option is given, whereupon all files and subdirectories of the directory are also removed.

Confirmation is requested for the removal of read-only files unless the `-f` option is given.

FILES

`/dev/rfd1024`

SEE ALSO

`adfscat(1)`, `wradfs(1)`, `adfs(1)`, `adfscp(1)`, `msdoscat(1)`, `msdosls(1)`, `wrmsdos(1)`, `msdosrm(1)`, `msdoscp(1)`, `fd(4)`.

NAME

`cat` - catenate and print

SYNOPSIS

`cat [-u] [-n] [-s] [-v] file ...`

DESCRIPTION

Cat reads each *file* in sequence and displays it on the standard output. Thus

```
cat file
```

displays the file on the standard output, and

```
cat file1 file2 >file3
```

concatenates the first two files and places the result on the third.

If no input file is given, or if the argument '-' is encountered, *cat* reads from the standard input file. Output is buffered in the block size recommended by *stat*(2) unless the standard output is a terminal, when it is line buffered. The `-u` option makes the output completely unbuffered.

The `-n` option displays the output lines preceded by lines numbers, numbered sequentially from 1. Specifying the `-b` option with the `-n` option omits the line numbers from blank lines.

The `-s` option crushes out multiple adjacent empty lines so that the output is displayed single spaced.

The `-v` option displays non-printing characters so that they are visible. Control characters print like `^X` for control-x; the delete character (octal 0177) prints as `^?`. Non-ascii characters (with the high bit set) are printed as `M-` (for meta) followed by the character of the low 7 bits. A `-e` option may be given with the `-v` option, which displays a '\$' character at the end of each line. Specifying the `-t` option with the `-v` option displays tab characters as `^I`.

SEE ALSO

`cp`(1), `ex`(1), `more`(1), `pr`(1), `tail`(1)

BUGS

Beware of '`cat a b >a`' and '`cat a b >b`', which destroy the input files before reading them.

NAME

`cd` – change working directory

SYNOPSIS

`cd` *directory*

DESCRIPTION

Directory becomes the new working directory. The process must have execute (search) permission in *directory*.

Because a new process is created to execute each command, *cd* would be ineffective if it were written as a normal command. It is therefore recognized and executed by the shells. In *csh*(1) you may specify a list of directories in which *directory* is to be sought as a subdirectory if it is not a subdirectory of the current directory; see the description of the *cdpath* variable in *csh*(1).

SEE ALSO

csh(1), *sh*(1), *pwd*(1), *chdir*(2)

NAME

chmod – change mode

SYNOPSIS

chmod [-Rf] mode file ...

DESCRIPTION

The mode of each named file is changed according to *mode*, which may be absolute or symbolic. An absolute *mode* is an octal number constructed from the OR of the following modes:

4000	set user ID on execution
2000	set group ID on execution
1000	sticky bit, see <i>chmod(2)</i>
0400	read by owner
0200	write by owner
0100	execute (search in directory) by owner
0070	read, write, execute (search) by group
0007	read, write, execute (search) by others

A symbolic *mode* has the form:

[*who*] *op* *permission* [*op* *permission*] ...

The *who* part is a combination of the letters *u* (for user's permissions), *g* (group) and *o* (other). The letter *a* stands for all, or *ugo*. If *who* is omitted, the default is *a* but the setting of the file creation mask (see *umask(2)*) is taken into account.

Op can be *+* to add *permission* to the file's mode, *-* to take away *permission* and *=* to assign *permission* absolutely (all other bits will be reset).

Permission is any combination of the letters *r* (read), *w* (write), *x* (execute), *X* (set execute only if file is a directory or some other execute bit is set), *s* (set owner or group id) and *t* (save text – sticky). Letters *u*, *g*, or *o* indicate that *permission* is to be taken from the current mode. Omitting *permission* is only useful with *=* to take away all permissions.

When the *-R* option is given, *chmod* recursively descends its directory arguments setting the mode for each file as described above. When symbolic links are encountered, their mode is not changed and they are not traversed.

If the *-f* option is given, *chmod* will not complain if it fails to change the mode on a file.

EXAMPLES

The first example denies write permission to others, the second makes a file executable by all if it is executable by anyone:

```
chmod o-w file
chmod +X file
```

Multiple symbolic modes separated by commas may be given. Operations are performed in the order specified. The letter *s* is only useful with *u* or *g*.

Only the owner of a file (or the super-user) may change its mode.

SEE ALSO

ls(1), chmod(2), stat(2), umask(2), chown(8)

NAME

cp, cp0 - copy

SYNOPSIS

cp [-ip] file1 file2

cp [-ipr] file ... directory

DESCRIPTION

File1 is copied onto *file2*. By default, the mode and owner of *file2* are preserved if it already existed; otherwise the mode of the source file modified by the current *umask*(2) is used. The *-p* option causes *cp* to attempt to preserve (duplicate) in its copies the modification times and modes of the source files, ignoring the present *umask*.

In the second form, one or more *files* are copied into the *directory* with their original file-names.

Cp refuses to copy a file onto itself.

If the *-i* option is specified, *cp* will prompt the user with the name of the file whenever the copy will cause an old file to be overwritten. An answer of 'y' will cause *cp* to continue. Any other answer will prevent it from overwriting the file.

If the *-r* option is specified and any of the source files are directories, *cp* copies each subtree rooted at that name; in this case the destination must be a directory.

SEE ALSO

cat(1), mv(1), rcp(1C)

BUGS

Cp will create 'holes' in the output file for completely zero disc blocks, this saves considerable disc space on R140 systems. Some NFS fileservers (eg those running 4.2BSD) have a bug which cause them to add a zero byte to files copied in this manner. This can be avoided by using *cp0* which is identical to *cp* except that it copies the file byte for byte.

NAME

`learn` – computer aided instruction about UNIX

SYNOPSIS

`learn` [`-directory`] [`subject` [`lesson`]]

DESCRIPTION

Learn gives Computer Aided Instruction courses and practice in the use of UNIX, the C Shell, and the Berkeley text editors. To get started simply type `learn`. If you had used *learn* before and left your last session without completing a subject, the program will use information in `$HOME/learnrc` to start you up in the same place you left off. Your first time through, *learn* will ask questions to find out what you want to do. Some questions may be bypassed by naming a *subject*, and more yet by naming a *lesson*. You may enter the *lesson* as a number that *learn* gave you in a previous session. If you do not know the lesson number, you may enter the *lesson* as a word, and *learn* will look for the first lesson containing it. If the *lesson* is '-', *learn* prompts for each lesson; this is useful for debugging.

The *subject*'s presently handled are

```
files
editor
vi
morefiles
macros
eqn
C
```

There are a few special commands. The command 'bye' terminates a *learn* session and 'where' tells you of your progress, with 'where m' telling you more. The command 'again' re-displays the text of the lesson and 'again lesson' lets you review *lesson*. There is no way for *learn* to tell you the answers it expects in English, however, the command 'hint' prints the last part of the lesson script used to evaluate a response, while 'hint m' prints the whole lesson script. This is useful for debugging lessons and might possibly give you an idea about what it expects.

The `-directory` option allows one to exercise a script in a nonstandard place.

FILES

```
/usr/lib/learn    subtree for all dependent directories and files
/usr/tmp/pl*     playpen directories
$HOME/learnrc   startup information
```

SEE ALSO

`csh(1)`, `ex(1)`

B. W. Kernighan and M. E. Lesk, *LEARN – Computer-Aided Instruction on UNIX*

BUGS

The main strength of *learn*, that it asks the student to use the real UNIX, also makes possible baffling mistakes. It is helpful, especially for nonprogrammers, to have a UNIX initiate near at hand during the first sessions.

Occasionally lessons are incorrect, sometimes because the local version of a command operates in a non-standard way. Occasionally a lesson script does not recognize all the different correct responses, in which case the 'hint' command may be useful. Such lessons may be skipped with the 'skip' command, but it takes some sophistication to recognize the situation.

To find a *lesson* given as a word, *learn* does a simple `fgrep(1)` through the lessons. It is unclear whether this sort of subject indexing is better than none.

Spawning a new shell is required for each of many user and internal functions.

The 'vi' lessons are provided separately from the others. To use them see your system administrator.

NAME

lpq - spool queue examination program

SYNOPSIS

lpq [+[*n*]] [-*l*] [-*Pprinter*] [*job # ...*] [*user ...*]

DESCRIPTION

lpq examines the spooling area used by *lpd*(8) for printing files on the line printer, and reports the status of the specified jobs or all jobs associated with a user. *lpq* invoked without any arguments reports on any jobs currently in the queue. A *-P* flag may be used to specify a particular printer, otherwise the default line printer is used (or the value of the *PRINTER* variable in the environment). If a *+* argument is supplied, *lpq* displays the spool queue until it empties. Supplying a number immediately after the *+* sign indicates that *lpq* should sleep *n* seconds in between scans of the queue. All other arguments supplied are interpreted as user names or job numbers to filter out only those jobs of interest.

For each job submitted (i.e. invocation of *lpr*(1)) *lpq* reports the user's name, current rank in the queue, the names of files comprising the job, the job identifier (a number which may be supplied to *lprm*(1) for removing a specific job), and the total size in bytes. The *-l* option causes information about each of the files comprising the job to be printed. Normally, only as much information as will fit on one line is displayed. Job ordering is dependent on the algorithm used to scan the spooling directory and is supposed to be FIFO (First in First Out). File names comprising a job may be unavailable (when *lpr*(1) is used as a sink in a pipeline) in which case the file is indicated as "(standard input)".

If *lpq* warns that there is no daemon present (i.e. due to some malfunction), the *lpc*(8) command can be used to restart the printer daemon.

FILES

<i>/etc/termcap</i>	for manipulating the screen for repeated display
<i>/etc/printcap</i>	to determine printer characteristics
<i>/usr/spool/*</i>	the spooling directory, as determined from <i>printcap</i>
<i>/usr/spool/*cf*</i>	control files specifying jobs
<i>/usr/spool/*lock</i>	the lock file to obtain the currently active job

SEE ALSO

lpr(1), *lprm*(1), *lpc*(8), *lpd*(8)

BUGS

Due to the dynamic nature of the information in the spooling directory *lpq* may report unreliably. Output formatting is sensitive to the line length of the terminal; this can result in widely spaced columns.

DIAGNOSTICS

Unable to open various files. The lock file being malformed. Garbage files when there is no daemon active, but files in the spooling directory.

NAME

`lpr` - off line print

SYNOPSIS

```
lpr [ -Pprinter ] [ -#num ] [ -C class ] [ -J job ] [ -T title ] [ -i [ numcols ] ] [ -1234 font ] [ -wnum ] [ -pltndgvcfmbs ] [ name ... ]
```

DESCRIPTION

`Lpr` uses a spooling daemon to print the named files when facilities become available. If no names appear, the standard input is assumed. The `-P` option may be used to force output to a specific printer. Normally, the default printer is used (site dependent), or the value of the environment variable `PRINTER` is used.

The following single letter options are used to notify the line printer spooler that the files are not standard text files. The spooling daemon will use the appropriate filters to print the data accordingly.

- `-p` Use `pr(1)` to format the files (equivalent to `print`).
- `-l` Use a filter which allows control characters to be printed and suppresses page breaks.
- `-t` The files are assumed to contain data from `troff(1)` (cat phototypesetter commands).
- `-n` The files are assumed to contain data from `ditroff` (device independent troff).
- `-d` The files are assumed to contain data from `tex(1)` (DVI format from Stanford).
- `-g` The files are assumed to contain standard plot data as produced by the `plot(3X)` routines (see also `plot(1G)` for the filters used by the printer spooler).
- `-v` The files are assumed to contain a raster image for devices like the Benson Varian.
- `-c` The files are assumed to contain data produced by `cifplot(1)`.
- `-f` Use a filter which interprets the first character of each line as a standard FORTRAN carriage control character.

The remaining single letter options have the following meaning.

- `-r` Remove the file upon completion of spooling or upon completion of printing (with the `-s` option).
- `-m` Send mail upon completion.
- `-h` Suppress the printing of the burst page.
- `-s` Use symbolic links. Usually files are copied to the spool directory.

The `-C` option takes the following argument as a job classification for use on the burst page. For example,

```
lpr -C EECS foo.c
```

causes the system name (the name returned by `hostname(1)`) to be replaced on the burst page by `EECS`, and the file `foo.c` to be printed.

The `-J` option takes the following argument as the job name to print on the burst page. Normally, the first file's name is used.

The `-T` option uses the next argument as the title used by `pr(1)` instead of the file name.

To get multiple copies of output, use the `-#num` option, where `num` is the number of copies desired of each file named. For example,

```
lpr -#3 foo.c bar.c more.c
```

would result in 3 copies of the file `foo.c`, followed by 3 copies of the file `bar.c`, etc. On the other hand,

```
cat foo.c bar.c more.c | lpr -#3
```

will give three copies of the concatenation of the files.

The `-i` option causes the output to be indented. If the next argument is numeric, it is used as the number of blanks to be printed before each line; otherwise, 8 characters are printed.

The `-w` option takes the immediately following number to be the page width for *pr*.

The `-s` option will use *symlink(2)* to link data files rather than trying to copy them so large files can be printed. This means the files should not be modified or removed until they have been printed.

The option `-1234` Specifies a font to be mounted on font position *i*. The daemon will construct a *rainmag* file referencing */usr/lib/vfont/name.size*.

FILES

<i>/etc/passwd</i>	personal identification
<i>/etc/printcap</i>	printer capabilities data base
<i>/usr/lib/lpd*</i>	line printer daemons
<i>/usr/spool/*</i>	directories used for spooling
<i>/usr/spool/*/cf*</i>	daemon control files
<i>/usr/spool/*/df*</i>	data files specified in "cf" files
<i>/usr/spool/*/tf*</i>	temporary copies of "cf" files

SEE ALSO

lpq(1), *lprm(1)*, *pr(1)*, *symlink(2)*, *printcap(5)*, *lpc(8)*, *lpd(8)*

DIAGNOSTICS

If you try to spool too large a file, it will be truncated. *Lpr* will object to printing binary files. If a user other than root prints a file and spooling is disabled, *lpr* will print a message saying so and will not put jobs in the queue. If a connection to *lpd* on the local machine cannot be made, *lpr* will say that the daemon cannot be started. Diagnostics may be printed in the daemon's log file regarding missing spool files by *lpd*.

BUGS

Fonts for *troff* and *tex* reside on the host with the printer. It is currently not possible to use local font libraries.

NAME

lprm - remove jobs from the line printer spooling queue

SYNOPSIS

lprm [*-Pprinter*] [-] [job # ...] [user ...]

DESCRIPTION

Lprm will remove a job, or jobs, from a printer's spool queue. Since the spooling directory is protected from users, using *lprm* is normally the only method by which a user may remove a job.

Lprm without any arguments will delete the currently active job if it is owned by the user who invoked *lprm*.

If the *-* flag is specified, *lprm* will remove all jobs which a user owns. If the super-user employs this flag, the spool queue will be emptied entirely. The owner is determined by the user's login name and host name on the machine where the *lpr* command was invoked.

Specifying a user's name, or list of user names, will cause *lprm* to attempt to remove any jobs queued belonging to that user (or users). This form of invoking *lprm* is useful only to the super-user.

A user may dequeue an individual job by specifying its job number. This number may be obtained from the *lpq*(1) program, e.g.

```
% lpq -l
```

```
1st: ken                [job #013ucbarpa]
      (standard input)    100 bytes
```

```
% lprm 13
```

Lprm will announce the names of any files it removes and is silent if there are no jobs in the queue which match the request list.

Lprm will kill off an active daemon, if necessary, before removing any spooling files. If a daemon is killed, a new one is automatically restarted upon completion of file removals.

The *-P* option may be used to specify the queue associated with a specific printer (otherwise the default printer, or the value of the *PRINTER* variable in the environment is used).

FILES

```
/etc/printcap    printer characteristics file
/usr/spool/*     spooling directories
/usr/spool/*lock lock file used to obtain the pid of the current
                  daemon and the job number of the currently active job
```

SEE ALSO

lpr(1), *lpq*(1), *lpd*(8)

DIAGNOSTICS

"Permission denied" if the user tries to remove files other than his own.

BUGS

Since there are race conditions possible in the update of the lock file, the currently active job may be incorrectly identified.

NAME

`ls` - list contents of directory

SYNOPSIS

`ls [-acdfgilqrstu1ACLFR] name ...`

DESCRIPTION

For each directory argument, `ls` lists the contents of the directory; for each file argument, `ls` repeats its name and any other information requested. By default, the output is sorted alphabetically. When no argument is given, the current directory is listed. When several arguments are given, the arguments are first sorted appropriately, but file arguments are processed before directories and their contents.

There are a large number of options:

- l List in long format, giving mode, number of links, owner, size in bytes, and time of last modification for each file. (See below.) If the file is a special file the size field will instead contain the major and minor device numbers. If the file is a symbolic link the pathname of the linked-to file is printed preceded by "-->".
- g Include the group ownership of the file in a long output.
- t Sort by time modified (latest first) instead of by name.
- a List all entries; in the absence of this option, entries whose names begin with a period (.) are *not* listed.
- s Give size in kilobytes of each file.
- d If argument is a directory, list only its name; often used with -l to get the status of a directory.
- L If argument is a symbolic link, list the file or directory the link references rather than the link itself.
- r Reverse the order of sort to get reverse alphabetic or oldest first as appropriate.
- u Use time of last access instead of last modification for sorting (with the -t option) and/or printing (with the -l option).
- c Use time of file creation for sorting or printing.
- i For each file, print the i-number in the first column of the report.
- f Force each argument to be interpreted as a directory and list the name found in each slot. This option turns off -l, -t, -s, and -r, and turns on -a; the order is the order in which entries appear in the directory.
- F causes names to be marked with one of the following trailing characters: directories - '/', sockets - '=', symbolic links - '@', and executable files - '*'.
 - R recursively list subdirectories encountered.
 - l force one entry per line output format; this is the default when output is not to a terminal.
 - C force multi-column output; this is the default when output is to a terminal.
 - q force printing of non-graphic characters in file names as the character '?'; this is the default when output is to a terminal.

The mode printed under the -l option contains 11 characters which are interpreted as follows: the first character is

- d if the entry is a directory;
- b if the entry is a block-type special file;
- c if the entry is a character-type special file;

- l** if the entry is a symbolic link;
- s** if the entry is a socket, or
- if the entry is a plain file.

The next 9 characters are interpreted as three sets of three bits each. The first set refers to owner permissions; the next refers to permissions to others in the same user-group; and the last to all others. Within each set the three characters indicate permission respectively to read, to write, or to execute the file as a program. For a directory, 'execute' permission is interpreted to mean permission to search the directory. The permissions are indicated as follows:

- r** if the file is readable;
- w** if the file is writable;
- x** if the file is executable;
- if the indicated permission is not granted.

The group-execute permission character is given as **s** if the file has the set-group-id bit set; likewise the user-execute permission character is given as **S** if the file has the set-user-id bit set.

The last character of the mode (normally 'x' or '-') is **t** if the 1000 bit of the mode is on. See *chmod(1)* for the meaning of this mode.

When the sizes of the files in a directory are listed, a total count of blocks, including indirect blocks is printed.

FILES

/etc/passwd to get user id's for 'ls -l'.
/etc/group to get group id's for 'ls -g'.

BUGS

Newline and tab are considered printing characters in file names.

The output device is assumed to be 80 columns wide.

The option setting based on whether the output is a teletype is undesirable as "ls -s" is much different than "ls -s | lpr". On the other hand, not doing this setting would make old shell scripts which used *ls* almost certain losers.

NAME

`man` - find manual information by keywords; print out the manual

SYNOPSIS

```
man [ - ] [ -M path ] [ section ] title ...
man -k keyword ...
man -f file ...
```

DESCRIPTION

Man is a program which gives information from the programmers manual. It can be asked for one line descriptions of commands specified by name, or for all commands whose description contains any of a set of keywords. It can also provide on-line access to the sections of the printed manual.

When given the option `-k` and a set of keywords, *man* prints out a one line synopsis of each manual sections whose listing in the table of contents contains one of those keywords.

When given the option `-f` and a list of file names, *man* attempts to locate manual sections related to those files, printing out the table of contents lines for those sections.

When neither `-k` nor `-f` is specified, *man* formats a specified set of manual pages. If a section specifier is given *man* looks in that section of the manual for the given *titles*. *Section* is either an Arabic section number (3 for instance), or one of the words "new," "local," "old," or "public." A section number may be followed by a single letter classifier (for instance, 1g, indicating a graphics program in section 1). If *section* is omitted, *man* searches all sections of the manual, giving preference to commands over subroutines in system libraries, and printing the first section it finds, if any.

If the standard output is a teletype *man* pipes its output through *more*(1) with the option `-s` to crush out useless blank lines and to stop after each page on the screen. Hit a space to continue, a control-D to scroll 11 more lines when the output stops.

If the standard output is not a teletype, or the `-` flag is given, *man* pipes its output through *cat*(1). If `-` is specified the `-s` option is used to remove consecutive blank lines.

Normally *man* checks in a standard location for manual information (*/usr/man*). This can be changed by supplying a search path (like the *sh*(1) command search path) with the `-M` flag. The search path is a colon (":") separated list of directories in which manual subdirectories may be found; e.g. *"/usr/local:/usr/man"*. If the environment variable "MANPATH" is set, its value is used for the default path. If a search path is supplied with the `-k` or `-f` options, it must be specified first.

Man will look for the manual page in one of three forms, the *nroff* source, preformatted pages or the *manpages* archive found in */usr/man/manpages.**. If any version is available, the manual page will be displayed. If the preformatted version is available, and it has a more recent modify time than the *nroff* source, it will be promptly displayed. Otherwise, the manual page will be formatted with *nroff* and displayed. If neither preformatted nor *nroff* source versions are available and there is an entry in the archive this will be used. If the user has permission, the formatted manual page will be deposited in the proper place, so that later invocations of *man* will not need to format the page again.

Normally each manual page should have either an *nroff* source version or an archive entry - there is no point having both, the archive entry will simply be ignored.

FILES

<i>/usr/man</i>	standard manual area
<i>/usr/man/man?/*</i>	directories containing source for manuals
<i>/usr/man/cat?/*</i>	directories containing preformatted pages
<i>/usr/man/manpages.*</i>	man page archive and contents list
<i>/usr/man/whatis</i>	keyword database

SEE ALSO

apropos(1), more(1), whereis(1), catman(8), arcat(8)

BUGS

The manual is supposed to be reproducible either on the phototypesetter or on a typewriter. However, on a typewriter some information is necessarily lost.

The man pages archive and the preformatted versions of the man pages may contain escape sequences appropriate to the terminal for which they were formatted – if they are displayed on a terminal (or by a program) which expects different escapes the results are unpredictable.

NAME

`mkdir` – make a directory

SYNOPSIS

`mkdir` *dirname* ...

DESCRIPTION

Mkdir creates specified directories in mode *777*. Standard entries, '.', for the directory itself, and '..' for its parent, are made automatically.

Mkdir requires write permission in the parent directory.

SEE ALSO

`rmdir`(1)

NAME

more, page – file perusal filter for crt viewing

SYNOPSIS

more [*-cdfipstu*] [*-n*] [*+linenumber*] { *+lpattern* } [*name ...*]

page more options

DESCRIPTION

More is a filter which allows examination of a continuous text one screenful at a time on a soft-copy terminal. It normally pauses after each screenful, printing *--More--* at the bottom of the screen. If the user then types a carriage return, one more line is displayed. If the user hits a space, another screenful is displayed. Other possibilities are enumerated later.

The command line options are:

- n* An integer which is the size (in lines) of the window which *more* will use instead of the default.
- c* *More* will draw each page by beginning at the top of the screen and erasing each line just before it draws on it. This avoids scrolling the screen, making it easier to read while *more* is writing. This option will be ignored if the terminal does not have the ability to clear to the end of a line.
- d* *More* will prompt the user with the message "Press space to continue, 'q' to quit." at the end of each screenful, and will respond to subsequent illegal user input by printing "Press 'h' for instructions," instead of ringing the bell. This is useful if *more* is being used as a filter in some setting, such as a class, where many users may be unsophisticated.
- f* This causes *more* to count logical, rather than screen lines. That is, long lines are not folded. This option is recommended if *nroff* output is being piped through *ul*, since the latter may generate escape sequences. These escape sequences contain characters which would ordinarily occupy screen positions, but which do not print when they are sent to the terminal as part of an escape sequence. Thus *more* may think that lines are longer than they actually are, and fold lines erroneously. This option also stops *more* from trying to interpret the escape sequences it sees in the input.
- l* Do not treat *^L* (form feed) specially. If this option is not given, *more* will pause after any line that contains a *^L*, as if the end of a screenful had been reached. Also, if a file begins with a form feed, the screen will be cleared before the file is printed.
- p* Cause *more* to use page mode – effectively equivalent to *-c* or to using *page*.
- s* Squeeze multiple blank lines from the output, producing only one blank line. Especially helpful when viewing *nroff* output, this option maximizes the useful information present on the screen.
- u* Normally, *more* will handle underlining such as produced by *nroff* in a manner appropriate to the particular terminal: if the terminal can perform underlining or has a stand-out mode, *more* will output appropriate escape sequences to enable underlining or stand-out mode for underlined information in the source file. The *-u* option suppresses this processing. It also prevents *more* trying to interpret the escape sequences in the input unless the *-t* option is also given.
- t* Attempt to translate *vt100* escape sequences in the input to the appropriate control sequences for the terminal. This option also causes unrecognised escape sequences to be removed from the input, thus it is suitable for use with the output of *man(1)* on a terminal which does not support the standard *vt100* escape sequences.

+linenumber

Start up at *linenumber*.

*+/*pattern**

Start up two lines before the line containing the regular expression *pattern*.

If the program is invoked as *page*, then the screen is cleared before each screenful is printed (but only if a full screenful is being printed), and $k - 1$ rather than $k - 2$ lines are printed in each screenful, where k is the number of lines the terminal can display.

More looks in the file */etc/termcap* to determine terminal characteristics, and to determine the default window size. On a terminal capable of displaying 24 lines, the default window size is 22 lines.

More looks in the environment variable *MORE* to pre-set any flags desired. For example, if you prefer to view files using the *-c* mode of operation, the *csh* command *setenv MORE -c* or the *sh* command sequence *MORE='-c' ; export MORE* would cause all invocations of *more*, including invocations by programs such as *man* and *msgs*, to use this mode. Normally, the user will place the command sequence which sets up the *MORE* environment variable in the *.cshrc* or *.profile* file.

If *more* is reading from a file, rather than a pipe, then a percentage is displayed along with the *--More--* prompt. This gives the fraction of the file (in characters, not lines) that has been read so far.

Other sequences which may be typed when *more* pauses, and their effects, are as follows (*i* is an optional integer argument, defaulting to 1) :

i <space>

display *i* more lines, (or another screenful if no argument is given)

^D display 11 more lines (a "scroll"). If *i* is given, then the scroll size is set to *i*.

d same as ^D (control-D)

i z same as typing a space except that *i*, if present, becomes the new window size.

i s skip *i* lines and print a screenful of lines

i f skip *i* screenfuls and print a screenful of lines

i b skip back *i* screenfuls and print a screenful of lines

i ^B same as b

q or Q Exit from *more*.

= Display the current line number.

v Start up the editor *vi* at the current line.

h Help command; give a description of all the *more* commands.

i/*expr* search for the *i*-th occurrence of the regular expression *expr*. If there are less than *i* occurrences of *expr*, and the input is a file (rather than a pipe), then the position in the file remains unchanged. Otherwise, a screenful is displayed, starting two lines before the place where the expression was found. The user's erase and kill characters may be used to edit the regular expression. Erasing back past the first column cancels the search command.

i n search for the *i*-th occurrence of the last regular expression entered.

' (single quote) Go to the point from which the last search started. If no search has been performed in the current file, this command goes back to the beginning of the file.

!*command*

invoke a shell with *command*. The characters '%' and '!' in "command" are replaced with the current file name and the previous shell command respectively. If there is no current file name, '%' is not expanded. The sequences "\%" and "\!" are replaced by "%" and "!" respectively.

i:n skip to the *i*-th next file given in the command line (skips to last file if *n* doesn't make sense)

i:p skip to the *i*-th previous file given in the command line. If this command is given in the middle of printing out a file, then *more* goes back to the beginning of the file. If *i* doesn't make sense, *more* skips back to the first file. If *more* is not reading from a file, the bell is rung and nothing else happens.

:f display the current file name and line number.

:q or :Q exit from *more* (same as q or Q).

(dot) repeat the previous command.

The commands take effect immediately, i.e., it is not necessary to type a carriage return. Up to the time when the command character itself is given, the user may hit the line kill character to cancel the numerical argument being formed. In addition, the user may hit the erase character to redisplay the --More--(xx%) message.

At any time when output is being sent to the terminal, the user can hit the quit key (normally control-\). *More* will stop sending output, and will display the usual --More-- prompt. The user may then enter one of the above commands in the normal manner. Unfortunately, some output is lost when this is done, due to the fact that any characters waiting in the terminal's output queue are flushed when the quit signal occurs.

The terminal is set to *noecho* mode by this program so that the output can be continuous. What you type will thus not show on your terminal, except for the / and ! commands.

If the standard output is not a teletype, then *more* acts just like *cat*, except that a header is printed before each file (if there is more than one).

A sample usage of *more* in previewing *nroff* output would be

```
nroff -ms +2 doc.n | more -s
```

FILES

/etc/termcap	Terminal data base
/usr/lib/more.help	Help file

SEE ALSO

csh(1), man(1), msgs(1), script(1), sh(1), environ(7)

BUGS

Skipping backwards is too slow on large files.

The -t option deals with an unrecognised escape sequence by deleting the escape and the following character - in the manner of *col*(1). It should understand more escape sequences and be able to use more of the output terminals facilities.

NAME

msdoscat - catenate files from msdos 3.2 micro diskettes.

SYNOPSIS

msdoscat [*-b*] [*-t*] file . . .

DESCRIPTION

msdoscat writes to standard output the named file(s) on an *msdos* (version 3.2) micro diskette.

Each file name may be preceded by *-b* or by *-t* to indicate that that file and any subsequent files up to the next *-b* or *-t* indicator are to be regarded as binary or text. This is because text files under *msdos* have lines terminated by carriage-return and line feed rather than just line feed as under UNIX, and with the *-t* option, carriage-return characters are stripped.

If no *-b* or *-t* options are specified, text mode is everywhere assumed.

If a member of a subdirectory is required, UNIX-style */* notation is used to search down the directory tree on the diskette. The characters of the arguments may be in upper or lower case, and the filename extension and its initial dot may be omitted if blank.

FILES

/dev/rfd512

SEE ALSO

msdosrm(1), *wrmsdos(1)*, *msdosls(1)*, *msdoscp(1)*, *adfscat(1)*, *adfsls(1)*, *wradfs(1)*, *adfsrm(1)*, *adfscp(1)*, *fd(4)*.

NAME

`msdoscp` – copy files from msdos 3.2 micro diskettes.

SYNOPSIS

`msdoscp [-FVRbtm] file . . . directory`

DESCRIPTION

Msdoscp copies the specified file or files from the `msdos` (version 3.2) diskette to the specified UNIX directory. The UNIX directory may be a "." to denote the current directory.

A "." or a directory name may be substituted for an `msdos` file name to denote all the files in the root or the specified directory name. However subdirectories of the specified directory are not copied unless the `-R` option is given.

If an existing UNIX file would be overwritten, confirmation is requested unless the `-F` option is given.

The `-b` and `-t` options may be specified to denote that the files should be copied in binary or text mode. Alternatively the list of `msdos` files and directories may be interspersed with `-b` and `-t` options to change mode between each group of files.

The `-V` option causes *msdoscp* to give a blow-by-blow account of its activities.

The `-m` option causes the times and dates on the `msdos` files to be translated to UNIX-style times and dates and the access and modification times of the resulting files set accordingly.

FILES

`/dev/rfd512`

SEE ALSO

`msdoscat(1)`, `msdosrm(1)`, `wrmsdos(1)`, `msdosls(1)`, `adfscat(1)`, `adfls(1)`, `wradfs(1)`, `adfrsm(1)`, `adfscp(1)`, `fd(4)`.

NAME

`msdosls` - list files on msdos 3.2 micro diskettes.

SYNOPSIS

`msdosls [-CFLRdftr] [file]`

DESCRIPTION

`msdosls` produces output on the standard output relating to an `msdos` (version 3.2) micro diskette.

By default (with no arguments), the root level directory is examined and the entries therein printed in a single column, sorted alphabetically.

One or more filenames (as with `msdoscat`) may be given. Files are just printed, directories are listed out, all in the given order.

The `-C` option causes the entries to be listed in multiple columns.

The `-F` option causes a `/` to be appended to subdirectory names in the output.

The `-R` option causes subdirectories to be recursively examined.

The `-f` option suppresses the sorting of directory entries.

The `-d` option causes subdirectory names to be listed instead of the contents being printed.

The `-l` option causes information in addition to the file names to be output, in particular the attribute bits, the time and date, the clusters used by the file and the file size.

The `-t` option causes the files to be sorted into increasing order of age rather than alphabetically.

The `-r` option reverses the order of sorting of the directory entries.

FILES

`/dev/rfd512`

SEE ALSO

`msdoscat(1)`, `msdosrm(1)`, `wrmsdos(1)`, `msdoscp(1)`, `adfscat(1)`, `adfsfs(1)`, `wradfs(1)`, `adfsrm(1)`, `adfscp(1)`, `fd(4)`.

NAME

`msdosrm` – remove files from msdos (3.2) micro diskettes.

SYNOPSIS

`msdosrm [-frl] file`

DESCRIPTION

Msdosrm deletes one or more files from an `msdos` (version 3.2) diskette.

Confirmation is requested for the deletion of read-only and/or system files unless the `-f` option is given.

Directories are not deleted unless the `-r` option is given, whereupon the directory and all its contents are deleted.

FILES

`/dev/rfd512`

SEE ALSO

`msdoscat(1)`, `wrmsdos(1)`, `msdosls(1)`, `msdoscp(1)`, `adfscat(1)`, `adfs(1)`, `wradfs(1)`, `adfsrm(1)`, `adfscp(1)`, `fd(4)`.

NAME

`mv`, `mv0` – move or rename files

SYNOPSIS

`mv` [`-i`] [`-f`] [`-`] *file1* *file2*

`mv` [`-i`] [`-f`] [`-`] *file* ... *directory*

DESCRIPTION

Mv moves (changes the name of) *file1* to *file2*.

If *file2* already exists, it is removed before *file1* is moved. If *file2* has a mode which forbids writing, *mv* prints the mode (see *chmod(2)*) and reads the standard input to obtain a line; if the line begins with *y*, the move takes place; if not, *mv* exits.

In the second form, one or more *files* (plain files or directories) are moved to the *directory* with their original file-names.

Mv refuses to move a file onto itself.

Options:

`-i` stands for interactive mode. Whenever a move is to supercede an existing file, the user is prompted by the name of the file followed by a question mark. If he answers with a line starting with 'y', the move continues. Any other reply prevents the move from occurring.

`-f` stands for force. This option overrides any mode restrictions or the `-i` switch.

`-` means interpret all the following arguments to *mv* as file names. This allows file names starting with minus.

SEE ALSO

cp(1), *ln(1)*

BUGS

If *file1* and *file2* lie on different file systems, *mv* must copy the file and delete the original. In this case the owner name becomes that of the copying process and any linking relationship with other files is lost.

When copying to an output file *mv* will create 'holes' in the output file for completely zero disc blocks, this saves considerable disc space on R140 systems. Some NFS file servers (eg those running 4.2BSD) have a bug which cause them to add a zero byte to files copied in this manner. This can be avoided by using *mv0* which is identical to *mv* except that it copies the file byte for byte.

NAME

`psroff` - troff text formatting with PostScript output.

SYNOPSIS

`psroff` [option] ... [file] ...

DESCRIPTION

Psroff formats text in the named *files* for printing on a printer supporting postscript. Its capabilities are described in the *Nroff/Troff user's manual*.

If no *file* argument is present, the standard input is read. An argument consisting of a single minus (-) is taken to be a file name corresponding to the standard input.

By default *psroff* will spool its output using *lpr -PPostScript*. This action may be overridden by the use of the *-t* option, which causes the output to be directed to standard output.

The options accepted by *psroff* are exactly the same as those for *troff*.

FILES

<code>/tmp/ta*</code>	temporary file
<code>/usr/lib/tmac/tmac.*</code>	standard macro files
<code>/usr/lib/font/*</code>	font width tables for <i>psroff</i>
<code>/usr/adm/tracct</code>	accounting statistics for <code>/dev/cat</code>

SEE ALSO

`troff(1)`, `eqn(1)`, `tbl(1)`, `ms(7)`, `me(7)`, `man(7)`, `col(1)`

BUGS

The *fp* command is not supported - there is no way to access fonts other than the built in ones (Times-Roman, Times-Italic, Times-Bold and Symbol).

NAME

`pwd` – working directory name

SYNOPSIS

`pwd`

DESCRIPTION

Pwd prints the pathname of the working (current) directory.

SEE ALSO

`cd(1)`, `csh(1)`, `getwd(3)`

BUGS

In *csh(1)* the command *dirs* is always faster (although it can give a different answer in the rare case that the current directory or a containing directory was moved after the shell descended into it).

NAME

`rcp`, `rcp0` – remote file copy

SYNOPSIS

```
rcp file1 file2
rcp [-r] [-p] file ... directory
```

DESCRIPTION

`rcp` copies files between machines. Each *file* or *directory* argument is either a remote file name of the form `''rhost:path''`, or a local file name (containing no `':'` characters, or a `'/'` before any `':'`s.)

If the `-r` is specified and any of the source files are directories, `rcp` copies each subtree rooted at that name; in this case the destination must be a directory.

The `-p` option causes `rcp` to attempt to preserve (duplicate) in its copies the modified and accessed times of the source files.

If *path* is not a full path name, it is interpreted relative to your login directory on *rhost*. A *path* on a remote host may be quoted (using `\`, `"`, or ```) so that the metacharacters are interpreted remotely.

`rcp` does not prompt for passwords; your current local user name must exist on *rhost* and allow remote command execution via `rsh(1C)`.

`rcp` handles third party copies, where neither source nor target files are on the current machine. Hostnames may also take the form `''rname@rhost''` to use *rname* rather than the current user name on the remote host.

Please note: `rcp` is meant to copy from one host to another; if by some chance you try to copy a file on top of itself, you will end up with a severely corrupted file (for example, if you executed the following command from host `george`: `'george% rcp testfile george:/usr/me/testfile'`). Remember where you are at all times (putting your hostname in your prompt helps with this)!

SEE ALSO

`ftp(1C)`, `rsh(1C)`, `rlogin(1C)`

BUGS

Doesn't detect all cases where the target of a copy might be a file in cases where only a directory should be legal.

Is confused by any output generated by commands in a `.login`, `.profile`, or `.cshrc` file on the remote host.

`rcp` doesn't copy ownership, mode, and timestamps to the new files.

`rcp` requires that the source host have permission to execute commands on the remote host when doing third-party copies.

If you forget to quote metacharacters intended for the remote host you get an incomprehensible error message.

`rcp` will create 'holes' in the output file for completely zero disc blocks, this saves considerable disc space on R140 systems. Some NFS filesystems (eg those running 4.2BSD) have a bug which cause them to add a zero byte to files copied in this manner. This can be avoided by using `rcp0` which is identical to `rcp` except that it copies the file byte for byte.

`rcp` does not support the `host.user` syntax for destination addressing.

NAME

rlogin - remote login

SYNOPSIS

```
rlogin rhost [ -e c ] [ -S ] [ -L ] [ -I username ]
rhost [ -ec ] [ -S ] [ -L ] [ -I username ]
```

DESCRIPTION

Rlogin connects your terminal on the current local host system *lhost* to the remote host system *rhost*.

Each host has a file */etc/hosts.equiv* which contains a list of *rhost*'s with which it shares account names. (The host names must be the standard names as described in *rsh(1C)*.) When you *rlogin* as the same user on an equivalent host, you don't need to give a password. Each user may also have a private equivalence list in a file *.rhosts* in his login directory. Each line in this file should contain an *rhost* and a *username* separated by a space, giving additional cases where logins without passwords are to be permitted. If the originating user is not equivalent to the remote user, then a login and password will be prompted for on the remote machine as in *login(1)*. To avoid some security problems, the *.rhosts* file must be owned by either the remote user or root.

The remote terminal type is the same as your local terminal type (as given in your environment *TERM* variable). The terminal or window size is also copied to the remote system if the server supports the option, and changes in size are reflected as well. All echoing takes place at the remote site, so that (except for delays) the *rlogin* is transparent. Flow control via *^S* and *^Q* and flushing of input and output on interrupts are handled properly. The optional argument *-S* allows an eight-bit input data path at all times; otherwise parity bits are stripped except when the remote side's stop and start characters are other than *^S/^Q*. The argument *-L* allows the *rlogin* session to be run in litout mode. A line of the form "*~*," disconnects from the remote host, where "*~*" is the escape character. Similarly, the line "*~^Z*" (where *^Z*, control-Z, is the suspend character) will suspend the *rlogin* session. Substitution of the delayed-suspend character (normally *^Y*) for the suspend character suspends the send portion of the *rlogin*, but allows output from the remote system. A different escape character may be specified by the *-e* option. There is no space separating this option flag and the argument character.

SEE ALSO

rsh(1C)

FILES

*/usr/hosts/** for *rhost* version of the command

BUGS

More of the environment should be propagated.

NAME

rm, rmdir – remove (unlink) files or directories

SYNOPSIS

rm [**-f**] [**-r**] [**-i**] [**-**] file ...

rmdir dir ...

DESCRIPTION

Rm removes the entries for one or more files from a directory. If an entry was the last link to the file, the file is destroyed. Removal of a file requires write permission in its directory, but neither read nor write permission on the file itself.

If a file has no write permission and the standard input is a terminal, its permissions are printed and a line is read from the standard input. If that line begins with 'y' the file is deleted, otherwise the file remains. No questions are asked and no errors are reported when the **-f** (force) option is given.

If a designated file is a directory, an error comment is printed unless the optional argument **-r** has been used. In that case, *rm* recursively deletes the entire contents of the specified directory, and the directory itself.

If the **-i** (interactive) option is in effect, *rm* asks whether to delete each file, and, under **-r**, whether to examine each directory.

The null option **-** indicates that all the arguments following it are to be treated as file names. This allows the specification of file names starting with a minus.

Rmdir removes entries for the named directories, which must be empty.

SEE ALSO

rm(1), unlink(2), rmdir(2)

NAME

`rmdir`, `rm` – remove (unlink) directories or files

SYNOPSIS

`rmdir` dir ...

`rm` [`-f`] [`-r`] [`-i`] [`-`] file ...

DESCRIPTION

Rmdir removes entries for the named directories, which must be empty.

Rm removes the entries for one or more files from a directory. If an entry was the last link to the file, the file is destroyed. Removal of a file requires write permission in its directory, but neither read nor write permission on the file itself.

If a file has no write permission and the standard input is a terminal, its permissions are printed and a line is read from the standard input. If that line begins with 'y' the file is deleted, otherwise the file remains. No questions are asked and no errors are reported when the `-f` (force) option is given.

If a designated file is a directory, an error comment is printed unless the optional argument `-r` has been used. In that case, *rm* recursively deletes the entire contents of the specified directory, and the directory itself.

If the `-i` (interactive) option is in effect, *rm* asks whether to delete each file, and, under `-r`, whether to examine each directory.

The null option `-` indicates that all the arguments following it are to be treated as file names. This allows the specification of file names starting with a minus.

SEE ALSO

`rm(1)`, `unlink(2)`, `rmdir(2)`

NAME

rsh – remote shell

SYNOPSIS

```
rsh host [ -l username ] [ -n ] command
host [ -l username ] [ -n ] command
```

DESCRIPTION

Rsh connects to the specified *host*, and executes the specified *command*. *Rsh* copies its standard input to the remote command, the standard output of the remote command to its standard output, and the standard error of the remote command to its standard error. Interrupt, quit and terminate signals are propagated to the remote command; *rsh* normally terminates when the remote command does.

The remote username used is the same as your local username, unless you specify a different remote name with the `-l` option. This remote name must be equivalent (in the sense of *rlogin*(1C)) to the originating account; no provision is made for specifying a password with a command.

If you omit *command*, then instead of executing a single command, you will be logged in on the remote host using *rlogin*(1C).

Shell metacharacters which are not quoted are interpreted on local machine, while quoted metacharacters are interpreted on the remote machine. Thus the command

```
rsh otherhost cat remotefile >> localfile
```

appends the remote file *remotefile* to the localfile *localfile*, while

```
rsh otherhost cat remotefile ">>" otherremotefile
```

appends *remotefile* to *otherremotefile*.

Host names are given in the file `/etc/hosts`. Each host has one standard name (the first name given in the file), which is rather long and unambiguous, and optionally one or more nicknames. The host names for local machines are also commands in the directory `/usr/hosts`; if you put this directory in your search path then the **rsh** can be omitted.

FILES

```
/etc/hosts
/usr/hosts/*
```

SEE ALSO

rlogin(1C)

BUGS

If you are using *csht*(1) and put a *rsh*(1C) in the background without redirecting its input away from the terminal, it will block even if no reads are posted by the remote command. If no input is desired you should redirect the input of *rsh* to `/dev/null` using the `-n` option.

You cannot run an interactive command (like *rogue*(6) or *vi*(1)); use *rlogin*(1C).

Stop signals stop the local *rsh* process only; this is arguably wrong, but currently hard to fix for reasons too complicated to explain here.

NAME

uwm - a window manager for X

SYNTAX

uwm [-display *display*] [-f *filename*]

DESCRIPTION

uwm is a window manager for the X11 window server.

When **uwm** is invoked, it searches a predefined search path to locate any **uwm** startup files. If no startup files exist, **uwm** uses its built-in default file.

If startup files exist in any of the following locations, it adds the information contained in them to the defaults. In the case of contention, the variables in the last file found override previous specifications. Files in the **uwm** search path are:

```
/usr/lib/X11/uwm/system.uwmrc
$HOME/.uwmrc
```

To use only the settings defined in a single startup file, include the variables, **resetbindings**, **resetmenus**, **resetvariables** at the top of that specific startup file.

OPTIONS

-f *filename*

Names an alternate file as a **uwm** startup file.

STARTUP FILE VARIABLES

Variables are typically entered first, at the top of the startup file. By convention, **resetbindings**, **resetmenus**, and **resetvariables** head the list.

autoselect/noautoselect

places menu cursor in first menu item. If unspecified, menu cursor is placed in the menu header when the menu is displayed.

delta=*pixels*

indicates the number of pixels the cursor is moved before the action is interpreted by the window manager as a command. (Also refer to the **delta** mouse action.)

freeze/nofreeze

locks all other client applications out of the server during certain window manager tasks, such as move and resize.

grid/nogrid

displays a finely-ruled grid to help you position an icon or window during resize or move operations.

hiconpad=*n*

indicates the number of pixels to pad an icon horizontally. The default is five pixels.

hmenupad=*n*

indicates the amount of space in pixels, that each menu item is padded to the left and right of the text.

iconfont=*fontname*

names the font that is displayed within icons. Font names for a given server can be obtained using *xifonts(1)*.

maxcolors=*n*

limits the number of colors the window manager can use in a given invocation. If set to zero, or not specified, **uwm** assumes no limit to the number of colors it can take from the color map. **maxcolors** counts colors as they are included in the file.

normali/nonormali

places icons created with **f.newiconify** within the root window, even if it is placed partially off the screen. With **nonormali** the icon is placed exactly where the cursor leaves it.

normalw/nonormalw

places window created with **f.newiconify** within the root window, even if it is placed partially off the screen. With **nonormalw** the window is placed exactly where the cursor leaves it.

push=*n*

moves a window *n* number of pixels or a relative amount of space, depending on whether **pushabsolute** or **pushrelative** is specified. Use this variable in conjunction with **f.pushup**, **f.pushdown**, **f.pushright**, or **f.pushleft**.

pushabsolute/pushrelative

pushabsolute indicates that the number entered with push is equivalent to pixels. When an **f.push** (left, right, up, or down) function is called, the window is moved exactly that number of pixels.

pushrelative indicates that the number entered with the push variable represents a relative number. When an **f.push** function is called, the window is invisibly divided into the number of parts you entered with the push variable, and the window is moved one part.

resetbindings, resetmenus, and resetvariables

resets all previous function bindings, menus, and variables entries, specified in any startup file in the *uwm* search path, including those in the default environment. By convention, these variables are entered first in the startup file.

resizefont=*fontname*

identifies the font of the indicator that displays in the corner of the window as you resize windows. See *x1sfonts(1)* for obtaining font names.

resizerelative/noresizerelative

indicates whether or not resize operations should be done relative to moving edge or edges. By default, the dynamic rectangle uses the actual pointer location to define the new size.

reverse/noreverse

defines the display as black characters on a white background for the window manager windows and icons.

viconpad=*n*

indicates the number of pixels to pad an icon vertically. Default is five pixels.

vmenupad=*n*

indicates the amount of space in pixels that the menu is padded above and below the text.

volume=*n*

increases or decreases the base level volume set by the *xset(1)* command. Enter an integer from 0 to 7, 7 being the loudest.

zap/nozap

causes ghost lines to follow the window or icon from its previous default location to its new location during a move or resize operation.

BINDING SYNTAX

"function=[control key(s)]:[context]:mouse events:" menu name "

Function and mouse events are required input. Menu name is required with the *f.menu* function definition only.

Function**f.beep**

emits a beep from the keyboard. Loudness is determined by the volume variable.

f.circledown

causes the top window that is obscuring another window to drop to the bottom of the stack of windows.

f.circleup

exposes the lowest window that is obscured by other windows.

f.continue	releases the window server display action after you stop action with the f.pause function.
f.focus	directs all keyboard input to the selected window. To reset the focus to all windows, invoke <i>ffocus</i> from the root window.
f.iconify	when implemented from a window, this function converts the window to its respective icon. When implemented from an icon, f.iconify converts the icon to its respective window.
f.lower	lowers a window that is obstructing a window below it.
f.menu	invokes a menu. Enclose 'menu name' in quotes if it contains blank characters or parentheses. <i>f.menu=[control key(s)]:[context]:mouse events:" menu name "</i>
f.move	moves a window or icon to a new location, which becomes the default location.
f.moveopaque	moves a window or icon to a new screen location. When using this function, the entire window or icon is moved to the new screen location. The grid effect is not used with this function.
f.newiconify	allows you to create a window or icon and then position the window or icon in a new default location on the screen.
f.pause	temporarily stops all display action. To release the screen and immediately update all windows, use the f.continue function.
f.pushdown	moves a window down. The distance of the push is determined by the push variables.
f.pushleft	moves a window to the left. The distance of the push is determined by the push variables.
f.pushright	moves a window to the right. The distance of the push is determined by the push variables.
f.pushup	moves a window up. The distance of the push is determined by the push variables.
f.raise	raises a window that is being obstructed by a window above it.
f.refresh	results in exposure events being sent to the window server clients for all unobscured or partially obscured windows. The windows will not refresh correctly if the exposure events are not handled properly.
f.resize	resizes an existing window. Note that some clients, notably editors, react unpredictably if you resize the window while the client is running.
f.restart	causes the window manager application to restart, retracing the <i>uwm</i> search path and initializing the variables it finds.

Control Keys

By default, the window manager uses meta as its control key. It can also use ctrl, shift, lock, or null (no control key). Control keys must be entered in lower case, and can be abbreviated as: c, l, m, s for ctrl, lock, meta, and shift, respectively.

You can bind one, two, or no control keys to a function. Use the bar (|) character to combine control keys.

Note that client applications other than the window manager use the shift as a control key. If you bind the shift key to a window manager function, you can not use other client applications that require this key.

Context

The context refers to the screen location of the cursor when a command is initiated. When you include a context entry in a binding, the cursor must be in that context or the function will not be activated. The window manager recognizes the following four contexts: icon, window, root, (null).

The root context refers to the root, or background window. A (null) context is indicated when the context field is left blank, and allows a function to be invoked from any screen location. Combine contexts using the bar (!) character.

Mouse Buttons

Any of the following mouse buttons are accepted in lower case and can be abbreviated as l, m, or r, respectively: left, middle, right.

With the specific button, you must identify the action of that button. Mouse actions can be:

down function occurs when the specified button is pressed down.

up function occurs when the specified button is released.

delta indicates that the mouse must be moved the number of pixels specified with the delta variable before the specified function is invoked. The mouse can be moved in any direction to satisfy the delta requirement.

MENU DEFINITION

After binding a set of function keys and a menu name to *f.menu*, you must define the menu to be invoked, using the following syntax:

```

menu = " menu name " {
  "item name" : "action"
  .
  .
  .
}

```

Enter the menu name exactly the way it is entered with the *f.menu* function or the window manager will not recognize the link. If the menu name contains blank strings, tabs or parentheses, it must be quoted here and in the *f.menu* function entry. You can enter as many menu items as your screen is long. You cannot scroll within menus.

Any menu entry that contains quotes, special characters, parentheses, tabs, or strings of blanks must be enclosed in double quotes. Follow the item name by a colon (:).

Menu Action

Window manager functions

Any function previously described. E.g., *f.move* or *f.iconify*.

Shell commands

Begin with an exclamation point (!) and set to run in background. You cannot include a new line character within a shell command.

Text strings

Text strings are placed in the window server's cut buffer.

Strings starting with an up arrow (^) will have a new line character appended to the string after the up arrow (^) has been stripped from it.

Strings starting with a bar character (!) will be copied as is after the bar character (!) has been stripped.

Color Menus

Use the following syntax to add color to menus:

```
menu = "menu name" (color1:color2:color3:color4) {
  "item name" : (color5 :color6) : " action "
  .
  .
  .
}
```

color1 Foreground color of the header.
 color2 Background color of the header.
 color3 Foreground color of the highlighter, the horizontal band of color that moves with the cursor within the menu.
 color4 Background color of the highlighter.
 color5 Foreground color for the individual menu item.
 color6 Background color for the individual menu item.

Color Defaults

Colors default to the colors of the root window under any of the following conditions:

- 1) If you run out of color map entries, either before or during an invocation of *uwm*.
- 2) If you specify a foreground or background color that does not exist in the RGB color database of the server (see */usr/lib/X11/rgb.txt* for a sample) both the foreground and background colors default to the root window colors.
- 3) If you omit a foreground or background color, both the foreground and background colors default to the root window colors.
- 4) If the total number of colors specified in the startup file exceeds the number specified in the *maxcolors* variable.
- 5) If you specify no colors in the startup file.

EXAMPLES

The following is a very simple *uwm* setup file:

```
# Global variables
#
resetbindings;resetvariables;resetmenus
autoselect
delta=25
freeze
grid
hiconpad=5
hmenupad=6
iconfont=oldeng
menufont=timrom12b
resizefont=9x15
viconpad=5
vmenupad=3
volume=7
#
# Mouse button/key maps
```

```

#
# FUNCTION KEYS CONTEXT BUTTON MENU(if any)
# =====
f.menu = meta : :left down : "WINDOW OPS"
f.menu = meta : :middle down : "EXTENDED WINDOW OPS"
f.move = meta :wli :right down
f.circleup = meta :root :right down
#
# Menu specifications
#
menu = "WINDOW OPS" {
  "(De)Iconify": f.iconify
  Move: f.move
  Resize: f.resize
  Lower: f.lower
  Raise: f.raise
}

menu = "EXTENDED WINDOW OPS" {
  Create Window: ! "xterm &"
  Iconify at New Position: f.newiconify
  Focus Keyboard on Window: f.focus
  Freeze All Windows: f.pause
  Unfreeze All Windows: f.continue
  Circulate Windows Up: f.circleup
  Circulate Windows Down: f.circledown
}

```

RESTRICTIONS

The color specifications have no effect on a monochrome system.

FILES

```

/usr/lib/X11/uwm/system.uwmrc
$HOME/.uwmrc

```

SEE ALSO

X(1), Xserver(1), xset(1), xlsfonts(1)

COPYRIGHT

COPYRIGHT 1985, 1986, 1987, 1988
DIGITAL EQUIPMENT CORPORATION
MAYNARD, MASSACHUSETTS

ALL RIGHTS RESERVED.

THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT CORPORATION. DIGITAL MAKES NO REPRESENTATIONS ABOUT THE SUITABILITY OF THIS SOFTWARE FOR ANY PURPOSE. IT IS SUPPLIED "AS IS" WITHOUT EXPRESS OR IMPLIED WARRANTY.

IF THE SOFTWARE IS MODIFIED IN A MANNER CREATING DERIVATIVE COPYRIGHT RIGHTS, APPROPRIATE LEGENDS MAY BE PLACED ON THE DERIVATIVE WORK IN ADDITION TO THAT SET FORTH ABOVE.

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of Digital Equipment Corporation not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission.

AUTHOR

M. Gancarz, DEC Ultrix Engineering Group, Merrimack, New Hampshire, using some algorithms originally by Bob Scheifler, MIT Laboratory for Computer Science.

NAME

wm – a simple real-estate-driven window manager

SYNOPSIS

wm [-display *display*]

DESCRIPTION

Wm is a very primitive overlapping window manager for *X11*. It was developed to help with the debugging of the *X11* server; we do not suggest that the user interface presented here is a desired one, and we do not suggest that you try to use this program on a regular basis.

Wm decorates each mapped application window with a banner. The banner consists of four fields. Left-to-right, they are:

Circulate button - A command button which causes the window to change its position in the stacking order.

Title region - An area in which an applications name or other specified information is displayed. It is also used by the user to move the window.

Iconize button - A command button which causes the window to be replaced by an icon.

Resize button - A command button which allows the user to change the size of the window.

Wm supports the following user actions:

Raising or lowering a window in the stack of windows

Locating the pointer cursor in the Circulate box of a partially obscured window and clicking with any pointer button will raise this window to the top of the stack of windows so that it is no longer obscured. Locating a pointer cursor in the Circulate box of a window which is currently on top of the window stack will send the window to the bottom of the stack.

Iconizing a window

Locating the pointer cursor in the Iconize box and clicking any pointer button will cause the window to be unmapped, and the associated icon to become mapped. The icon will appear at its last location, or, if this window has never been iconized, under the cursor. However, if the client program initially set an icon position in the WM_HINTS property, then that icon position will be used instead as the initial icon position. To position an icon while iconizing the window, locate the cursor in the Iconize box and press down any pointer button. A rubber-band outline of the icon will appear under the cursor. While holding down the pointer button, drag the cursor to the desired location for the icon. The outline will follow the cursor on the screen. When the outline moves to the desired location for the icon, release the pointer button. The client window will be unmapped, and its icon will appear at the desired location. To cancel this operation while the pointer button is down, click another pointer button.

Deiconizing an icon

Locating the pointer cursor in an icon and clicking any pointer button will cause the icon to be unmapped, and the associated window to become mapped. To cancel this operation while the pointer button is down, click another pointer button.

Moving a window on the screen

Locating pointer cursor in the area of the title region and pressing any pointer button causes a "rubber-band" outline of the window to appear. As the user moves ("drags") the cursor (while holding down the pointer button), the outline moves accordingly. When the button is released, the window is repainted in the last location of the rubber-band outline. If the user presses another pointer button during the drag, the operation is cancelled, the rubber-band outline disappears, and the window is not moved. Note that a portion of the title region is constrained to remain on the screen.

Resizing a window.

Locating the pointer cursor in the resize box and pressing any pointer button initiates the spring-loaded resize mode. Then as soon as the cursor touches a border (while the pointer button is down), that border becomes a rubber-band line which follows the cursor until the button is released. If the cursor then touches an adjacent border, that border also becomes a rubber-band line, and the window can be resized in two dimensions at once. If the cursor touches a border after having touched the opposite border, the first border touched reverts to its original location, and the other one becomes a rubber-band line which follows the cursor. If the user presses another pointer button during the drag, the operation is cancelled, the rubber-band outline disappears, and the window does not change size. Note that the pointer cursor has to touch a border to initiate the resize action. As in the move operation, a portion of the title region is constrained to remain on the screen.

Moving an icon on the screen

To move an icon, press the Shift key and hold it, then position the pointer cursor in the icon, press any pointer button, and proceed dragging an outline of the icon around by moving the pointer cursor (with the pointer button down). When the outline moves to the desired position, release the pointer button and the Shift key. To cancel, click another pointer button during the drag; the icon will not move.

NOTES FOR CLIENT PROGRAMS

Wm uses the WM_ICON_NAME, WM_NAME, and WM_HINTS properties. It keeps the name in the Title region updated as the WM_NAME property changes. It keeps the name in the icon updated as the WM_ICON_NAME property changes; if a client does not set the WM_ICON_NAME property, *wm* will use the WM_NAME property for the icon name. *Wm* allows only text icons, and sets the icon sizes to accommodate the icon name. The maximum name length for both the icon name and the Title region name is 100 characters.

Of the WMHints, *wm* ignores all but icon_x and icon_y, which it uses for initial icon placement. These need to be set by the client before its window is mapped, because *wm* reads them only once, when it first encounters the window.

SEE ALSO

X(1), *Inter-Client Communication Conventions Manual*

BUGS

This program does not necessarily implement the current window manager protocols.

DIAGNOSTICS

If you try to run *wm* while you are already running a window manager, *wm* will let you know.

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology.
See *X(1)* for a full statement of rights and permissions.

AUTHOR

Hania Gajewska, DEC WSL Dave Rosenthal, Sun Microsystems

NAME

wradfs – write adfs micro diskettes.

SYNOPSIS

wradfs [**-fFaR**] [**-Dchar**] [**-ttitle**] [**-nname**] [**-bn**] [**-ddirectory**] [**-F**] [files]

DESCRIPTION

Wradfs writes a set of files to an **adfs** diskette.

By default the files are written to the root directory, and the diskette is assumed to be already formatted and initialised to an **adfs** file structure. Files of the same name are overwritten unless they are marked read-only, in which case confirmation is requested.

The argument **-R** causes UNIX directory names in the list of arguments to be recursively copied with their contents and subdirectories to the **adfs** file structure. Without this option directory names are assumed to be errors and cause diagnostics to be printed.

The argument **-a** causes the amount of free space left on the disc to be displayed after the files have been copied.

The argument **-i** causes the information already on the diskette to be discarded and a new **adfs** structure to be created.

The argument **-f** causes the diskette to be formatted before anything is written. This option implies the **-i** option. Confirmation is requested before proceeding.

The **-F** option suppresses confirmation messages and implies an affirmative answer in every case.

The argument **-nname** causes the disc name to be set to that specified. If this argument is not present a disc name is created from the time and day of initialisation (e.g. 20_54_Fri).

The argument **-ttitle** causes the title field in the root directory to be set to or replaced by the specified title. If no title is specified the title "Initialised by Unix" is used.

The argument **-bnumber** causes the boot option to be set or reset to the specified number.

The **-d** option specifies a directory other than the root directory which is created if necessary to which the files (and subdirectories created by the **-R** option) are written. The same notation as with **adfs** is used to specify directories.

The **-Tddd** option, with up to 3 hexadecimal digits may be used to specify an ADFS file type other than the default of FFF.

Each file name may be preceded by **-lnumber** and/or by **-enumber**. These flags cause the following file and each subsequent file up to the next such flag to have the load or execution address set to the specified hexadecimal number. If no such flags are given the load and execution addresses are set from the modification date of each file and the ADFS file type. If only one of these flags is given, then the other one is taken to be zero.

Fullstops in file or directory names are replaced by minus signs, unless the **-Dchar** option is specified, which causes the fullstops to be replaced by the specified character instead.

The write bit is not set on files copied from read-only UNIX files.

FILES

`/dev/rfd1024`

SEE ALSO

`adfscat(1)`, `adfs(1)`, `adfs(1)`, `adfscp(1)`, `adfsm(1)`, `msdoscat(1)`, `msdos(1)`, `wrmsdos(1)`, `msdosrm(1)`, `msdoscp(1)`, `fd(4)`.

NAME

wrmsdos - write files onto msdos (3.2) micro diskettes.

SYNOPSIS

wrmsdos [**-f****IaR**] [**-d** directory] [**-t**] [**-b**] file . . .

DESCRIPTION

Wrmsdos writes a set of files to an **msdos** (version 3.2) diskette.

By default the files are written to the root directory, and the diskette is assumed to be already formatted and initialised to an **msdos** structure. Files of the same name are overwritten unless they are marked read-only, in which case confirmation is requested.

The argument **-R** causes UNIX directory names in the list of arguments to be recursively copied with their contents and subdirectories to the **msdos** file structure. Without this option directory names are assumed to be errors and cause diagnostics to be printed.

The argument **-a** causes the amount of free space left on the diskette to be displayed after the files have been copied.

The argument **-i** causes the information already on the diskette to be discarded and a new **msdos** structure created.

The argument **-f** causes the diskette to be formatted before anything is written. This option implies the **-i** option. Confirmation is requested before proceeding.

The **-F** option suppresses confirmation messages and implies an affirmative answer in every case.

The **-d** option specifies a directory other than the root directory which is created if necessary to which the files (and subdirectories created by the **-R** option) are written. The same notation as with *msdosls* is used to specify subdirectories.

As with *msdoscat*, each file name may be preceded by **-b** or by **-t** to indicate that the file and subsequent files up to the next such flag are to be treated as binary or text respectively. Text files are output with a carriage-return character prepended to each linefeed character and a control-Z character at the end, and non-printing characters are stripped. If a file contains more than a small number of non-printing characters, the user is asked to confirm his intentions.

If no **-b** or **-t** options are specified, text mode is everywhere assumed.

The read-only attribute is set on files copied from read-only UNIX files and the time and date fields are set from the modification times of the UNIX files.

FILES

/dev/rfd512

SEE ALSO

msdoscat(1), *msdoscp*(1), *msdosls*(1), *msdosrm*(1), *adfscat*(1), *adfsls*(1), *wradfs*(1), *adfsm*(1), *adfscp*(1), *fd*(4).

NAME

X - a portable, network transparent window system

SYNOPSIS

X is a network transparent window system developed at MIT which runs under a wide variety of operating systems. The standard distribution from MIT works on Ultrix-32 Version 1.2 (and higher), 4.3BSD Unix, SunOS 3.2 (and higher), HP-UX 6.01, and DOMAIN/IX 9.7. In addition, many vendors support the X Window System under other operating systems.

THE OFFICIAL NAMES

The official names of the software described herein are:

X
X Window System
X Version 11
X Window System, Version 11
X11

Note that the phrases X.11, X-11, X Windows or any permutation thereof, are explicitly excluded from this list and should not be used to describe the X Window System (window system should be thought of as one word).

X Window System is a trademark of the Massachusetts Institute of Technology.

DESCRIPTION

X window system servers run on computers with bitmap displays. The server distributes user input to, and accepts output requests from various client programs through a variety of different interprocess communication channels. Although the most common case is for the client programs to be running on the same machine as the server, clients can be run transparently from other machines (including machines with different architectures and operating systems) as well.

X supports overlapping hierarchical subwindows and text and graphics operations, on both monochrome and color displays. For a full explanation of functions, see the *Xlib - C Language X Interface* manual, the *X Window System Protocol* specification, and various toolkit documents.

When you first log in on a display running X, you are usually using the *xterm(1)* terminal emulator program. You need not learn anything extra to use a display running X as a terminal beyond moving the mouse cursor into the login window to log in normally.

The core X protocol provides mechanism, not policy. Windows are manipulated (including moving, resizing and iconifying) not by the server itself, but by a separate program called a "window manager" of your choosing. This program is simply another client and requires no special privileges. If you don't like the ones that are supplied (see *uwm(1)* and *wm(1)*), you can write your own.

The number of programs that use X is growing rapidly. Of particular interest are: a terminal emulator (*xterm(1)*), window managers (*wm(1)* and *uwm(1)*), a mailer reader (*xmh(1)*), a bitmap editor (*bitmap(1)*), an access control program (*xhost(1)*), user preference setting programs (*xset(1)*, *xsetroot(1)*, and *xmodmap(1)*), a load monitor (*xload(1)*), clock (*xclock(1)*), a font displayer (*xfd(1)*), a protocol translator for running X10 programs (*x10tox11(1)*), and various demos (*ico(1)*, *muncher(1)*, *puzzle(1)*, etc.).

DISPLAY SPECIFICATION

When you first log in, the environment variable DISPLAY will be set to a string specifying the name of the machine on which the server is running, a number indicating which of possibly several servers to use, and possibly a number indicating the default screen of the server (usually this is omitted and defaults to 0). By convention, servers on a particular machine are numbered starting with zero. The format of the DISPLAY string depends on the type of communications

channel used to contact the server.

The following connection protocols are supported:

TCP/IP

DISPLAY should be set to "*host:dpy.screen*" where *host* is the symbolic name of the machine (e.g. expo), *dpy* is the number of the display (usually 0), and *screen* is the number of the screen. The *screen* and preceding period are optional, with the default value being zero (0). Full Internet domain names (e.g. expo.lcs.mit.edu) are allowed for the host name.

Unix domain

DISPLAY should be set to "*unix:dpy.screen*", where *dpy* is the display number and *screen* is the screen number; *screen* and the preceding period are optional, with the default value being zero (0).

DECnet

DISPLAY should be set to "*nodename::dpy.screen*" where *nodename* is the symbolic name of the machine, *dpy* is the display number, and *screen* is the screen number; *screen* and the preceding period are optional, with the default value being zero (0).

Most programs accept a command line argument of the form "*-display display*" that can be used to override the DISPLAY environment variable.

GEOMETRY SPECIFICATION

One of the advantages of using window systems over hardwired terminals is that applications don't have to be restricted to a particular size or location on the screen. Although the layout of windows on a display is controlled by the window manager that the user is running, most applications accept a command line argument that is treated as the preferred size and location for this particular application's window.

This argument, usually specified as "*-geometry WxH+X+Y*," indicates that the window should have a width of W and height of H (usually measured in pixels or characters, depending on the application), and the upper left corner X pixels to the right and Y pixels below the upper left corner of the screen (origin (0,0)). "*WxH*" can be omitted to obtain the default application size, or "*+X+Y*" can be omitted to obtain the default application position (which is usually then left up to the window manager or user to choose). The X and Y values may be negative to position the window off the screen. In addition, if minus signs are used instead of plus signs (e.g. *WxH-X-Y*), then (X,Y) represents the location of the lower right hand corner of the window relative to the lower right hand corner of the screen.

By combining plus and minus signs, the window may be placed relative to any of the four corners of the screen. For example:

555x333+11+22

This will request a window 555 pixels wide and 333 pixels tall, with the upper left corner located at (11,22).

300x200-0+0

This will request a window measuring 300 by 200 pixels in the upper right hand corner of the screen.

48x48--5--10

This will request a window measuring 48 by 48 pixels whose lower right hand corner is 5 pixel off the right edge and the screen and 10 pixels off the bottom edge.

COMMAND LINE ARGUMENTS

Most X programs attempt to use a common set of names for their command line arguments. The X Toolkit automatically handles the following arguments:

- bg *color*, -background *color***
Either option specifies the color to use for the window background.
- bd *color*, -bordercolor *color***
Either option specifies the color to use for the window border.
- bw *number*, -borderwidth *number***
Either option specifies the width in pixels of the window border.
- display *display***
This option specifies the name of the X server to use.
- fg *color*, -foreground *color***
Either option specifies the color to use for text or graphics.
- fn *font*, -font *font***
Either option specifies the font to use for displaying text.
- geometry *geometry***
This option specifies the initial size and location of the window.
- iconic**
This option indicates that application should start out in an iconic state. Note that how this state is represented is controlled by the window manager that the user is running.
- name**
This option specifies the name under which resources for the application should be found. This option is useful in shell aliases to distinguish between invocations of an application, without resorting to creating links to alter the executable file name.
- rv, -reverse**
Either option indicates that the program should simulate reverse video if possible, often by swapping the foreground and background colors. Not all programs honor this or implement it correctly. It is usually only used on monochrome displays.
- +rv**
This option indicates that the program should not simulate reverse video. This is used to override any defaults since reverse video doesn't always work properly.
- synchronous**
This option indicates that requests to the X server should be sent synchronously, instead of asynchronously. Since *Xlib* normally buffers requests to the server, errors do not necessarily get reported immediately after they occur. This option turns off the buffering so that the application can be debugged. It should never be used with a working program.
- title *string***
This option specifies the title to be used for this window. This information is sometimes used by a window manager to provide some sort of header identifying the window.
- xrm *resourcestring***
This option specifies a resource name and value to override any defaults. It is also very useful for setting resources that don't have explicitly command line arguments.

RESOURCES

To make the tailoring of applications to personal preferences easier, X supports several mechanisms for storing default values for program resources (e.g. background color, window title, etc.) Resources are specified as strings of the form "*name*subname*subsubname...: value*" (see the *Xlib* manual section *Using the Resource Manager* for more details) that are loaded into a client when it starts up. The *Xlib* routine *XGetDefault(3X)* and the resource utilities within the X Toolkit obtain resources from the following sources:

RESOURCE_MANAGER root window property

Any global resources that should be available to clients on all machines should be stored in the **RESOURCE_MANAGER** property on the root window using the *xrdb(1)* program.

application-specific directory

Any application- or machine-specific resources can be stored in the class resource files located in the **XAPLOADDIR** directory (this is a configuration parameter that is */usr/lib/X11/app-defaults* in the standard distribution).

XENVIRONMENT

Any user- and machine-specific resources may be specified by setting the **XENVIRONMENT** environment variable to the name of a resource file to be loaded by all applications. If this variable is not defined, the X Toolkit looks for a file named *.Xdefaults-hostname*, where *hostname* is the name of the host where the application is executing.

-xrm resourcestring

Applications that use the X Toolkit can have resources specified from the command line. The *resourcestring* is a single resource name and value as shown above. Note that if the string contains characters interpreted by the shell (e.g., asterisk), they must be quoted. Any number of **-xrm** arguments may be given on the command line.

Program resources are organized into groups called "classes," so that collections of individual "instance" resources can be set all at once. By convention, the instance name of a resource begins with a lowercase letter and class name with an upper case letter. Multiple word resources are concatenated with the first letter of the succeeding words capitalized. Applications written with the X Toolkit will have at least the following resources:

background (class **Background**)

This resource specifies the color to use for the window background.

borderWidth (class **BorderWidth**)

This resource specifies the width in pixels of the window border.

borderColor (class **BorderColor**)

This resource specifies the color to use for the window border.

Most X Toolkit applications also have the resource **foreground** (class **Foreground**), specifying the color to use for text and graphics within the window.

By combining class and instance specifications, application preferences can be set quickly and easily. Users of color displays will frequently want to set **Background** and **Foreground** classes to particular defaults. Specific color instances such as text cursors can then be overridden without having to define all of the related resources.

When a named resource is unavailable (for example, a color named *chartrusse* or a font named *teenyweeney*), normally no error message will be printed; whether or not useful results ensue is dependent on the particular application. If you wish to see error messages (for example, if an application is failing for an unknown reason), you may specify the value "on" for the resource named "StringConversionWarnings." If you want such warnings for all applications, specify "*StringConversionWarnings:on" to the resource manager. If you want warnings only for a single application named "zowie", specify "zowie*StringConversionWarnings:on" to the resource manager.

DIAGNOSTICS

The default error handler uses the Resource Manager to build diagnostic messages when error conditions arise. The default error database is stored in the file **XErrorDB** in the directory specified by the **LIBDIR** configuration parameter (*/usr/lib/X11* in the standard distribution). If this file is not installed, error messages will tend to be somewhat cryptic.

SEE ALSO

xterm(1), bitmap(1), ico(1), muncher(1), plaid(1), puzzle(1), resize(1), uwm(1), wm(1), x10tox11(1), xbiff(1), xcalc(1), xclock(1), xedit(1), xfd(1), xhost(1), xinit(1), xload(1), xlogo(1), xlsfonts(1), xmh(1), xmodmap(1), xpr(1), xprkbd(1), xprop(1), xrdp(1), xrefresh(1), xset(1), xsetroot(1), xwd(1), xwininfo(1), xwud(1), Xserver(1), Xapollo(1), Xqss(1), Xqvss(1), Xsun(1), kbd_mode(1), todm(1), tox(1), biff(1), init(8), ttys(5), *Xlib - C Language X Interface*, *X Toolkit Intrinsic - C Language X Interface*

COPYRIGHT

The following copyright and permission notice outlines the rights and restrictions covering most parts of the standard distribution of the X Window System from MIT. Other parts have additional or different copyrights and permissions; see the individual source files.

Copyright 1984, 1985, 1986, 1987, 1988, Massachusetts Institute of Technology.

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of M.I.T. not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission. M.I.T. makes no representations about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty.

This software is not subject to any license of the American Telephone and Telegraph Company or of the Regents of the University of California.

AUTHORS

It is no longer feasible to list all people who have contributed something to X, but see doc/contributors in the standard sources.

NAME

Xarm – Arm server for X Version 11

SYNOPSIS

```
Xarm [display] [ -dev framebuffer ] [ -kbd keyboard ] [ -bw2 | -c4 ] [ -lowres | -mediumres | -highres | -py # ] [ -px # ] [ -width[:0] size ] [ -height[:0] size ] [ -bp[:0] colour ] [ -wp[:0] colour ] [ -frame[:0] colour ] [ -ri time ] [ -rd time ] [ -cd time ] [ -hard | -soft ] [ +flog | -flog ] [ -sched # ] ...
```

DESCRIPTION

Xarm is the server for Version 11 of the X window system on Acorn ARM hardware. It will normally be started by *init*(8), as a result of an entry in */etc/tty* file - see *tty*(5). It may also be started using *xinit*(1) or directly from a shell.

CONFIGURATIONS

Xarm operates under all ARM BSD4.3 UNIX versions later than the Beta release. It may be used on A500, A440 and A680 ARM hardware. It auto-configures to use a 1 bit per pixel black and white display, finding the display from the device */dev/fb* (see *fb*(4)) and the monitor type from the current CMOS ram settings. It uses the standard keyboard and mouse - see *kbd*(4).

The monitor type used by the X server, determined by the setting of the 'monitortype' CMOS ram variable, will normally cause the X server to start up with the correct display mode. The following default modes are used with the three different monitor types:-

0 – low resolution

Low resolution monitors use a 640x240x1 screen mode. Because this mode gives rectangular pixels it is normally unsuitable for graphics applications which will display distorted images. It is suitable for text based applications so long as the restricted number of lines available on the screen is acceptable. Suitable monitors are the standard A440 low resolution monochrome and colour monitors and multi-sync monitors.

The '-c4' option may be used to give a colour mode (640x240x4) - the colour map is a normal X pseudo colour map with 16 colours.

1 – medium resolution, multi-sync

Medium resolution modes may be used with a multi-sync monitor or a reasonable quality VGA type monitor. The default mode is 640x480x1. The '-c4' option may be used to give a 640x480x4 mode. The latter mode has a very detrimental effect on overall system performance, hence the default to the monochrome mode. Both modes have square pixels. Low resolution modes may also be selected with monitor type 1.

2 – high resolution

High resolution modes only support 1152x900x1 - the '-c4' option will be faulted. The mode requires a particular model of high resolution monitor; consult your dealer for more information.

3 – medium resolution, VGA

These modes work with most reasonable quality 'VGA' type monitors. Only the 640x480x1 and 640x480x4 modes may be selected - the monitors will not work with the low resolution modes. Consult your dealer for details of suitable VGA monitors, or use an Acorn recommended monitor - not all available monitors work.

The command line options allow a particular mode to be specified. Since the required mode may require a particular monitor type it may be necessary to change the CMOS ram settings. This requires superuser privilege - either use the *cmos*(8) programme and *reboot* unix or use the *configure* command from Arthur before starting unix up. The server will not attempt to change the monitor type itself - this is almost invariably the wrong thing to do.

The server supports compressed fonts. To save disc space the fonts are supplied in compressed form in the font directory and uncompressed on demand. For this to work the *uncompress* program must either be on the path of the X server or must be located at */usr/lib/uncompress* – the X server tries the latter name first. Since uncompressing files takes some time it can be advantageous to uncompress frequently used fonts, such as the *fixed* font. What you should do depends on the precise configuration of your system, use the *+flog* option to log the fonts which are used, then *uncompress* those which are opened most often. Notice that fonts which are used continuously, such as the cursor font or the *fixed* font (used by the *xterm* terminal emulator) are normally only actually loaded once – so uncompressing these fonts is unlikely to be worthwhile.

OPTIONS

The options may appear in any order. Only the ARM specific options are described here. All other options work except that the bell pitch cannot be changed. The bell volume has only four settings (apart from off) as does the key click loudness. Normally if repeated or conflicting options are given only the last is used, the exception are the monitor type options – *-lowres*, *-mediumres* and *-highres* – which cause the number of lines and the screen dimensions to be defaulted correctly. These options only alter values which have not yet been set, thus only the first such option is used. To avoid confusion you should avoid conflicting requests.

-dev *device*

Overrides the default frame buffer device */dev/fb*.

-kbd *device*

Overrides the default keyboard/mouse device */dev/kbd*.

The following options allow control of the screen mode. Normally only the first option will be used (to force a colour display on a low resolution or multi-sync monitor.) The *-lowres* option can be useful with multi-sync monitors (monitor type 1) as these will also support low resolution video modes. The other options are provided for completeness.

-c4 Forces the server to use colour 4 bit per pixel mode if possible – this is only supported on medium and low resolution monitors.

-bw2 Forces the server to use a monochrome mode – this is the default mode, it gives significantly better performance than the colour mode.

-lowres

Forces the server to switch to a low resolution display mode. This is normally only useful with a multi-sync monitor, where it is possible to use either a low resolution or medium resolution mode. Equivalent to 240 lines on the display. This mode is the default with monitor type 0.

-mediumres

Forces the server to switch to a medium resolution display mode. This is the default for monitor types 1 and 3. Equivalent to 480 lines on the display.

-highres

Forces the server to switch to select a high resolution mode – this is only possible with monitor type 2, and is the default. Equivalent to 900 lines on the display.

-py *#* Specifies the number of lines on the display. The server treats the value as a hint, it will select the mode with the nearest number of lines, subject to the number of lines being approximately right (it would not, for example, use a high resolution mode if *-py 512* had been requested.)

-px *#* Specifies the width of the display in pixels – again this is only a hint.

The width and height of the screen in millimetres are defaulted to suitable values according to the CMOS ram monitor type setting or to *-lowres*, *-mediumres* or *-highres* if given. The defaults are intended to match the monitors which Acorn either recommends or supplies. They can be overridden using the following options:-

-width[:0] size

Specifies the width of the displayed image on the screen, excluding borders.

-height[:0] size

Specifies the height of the displayed image on the screen, excluding borders.

The following options allow control over the colours used in the selected video mode. Since colour X servers select the colour palette themselves the black and white pixel selection facility only has an effect on black and white (**-bw2**) modes. The display border colour can be selected in any mode. In all cases the colour can be specified in one of three ways:-

0/1 If '0' is given it is interpreted as 'black', '1' is interpreted as 'white'.

rgb Three hex digits of red, green and blue values making a particular colour.

colour-name

The name of a colour from the *rgb* colour data base (as specified by the **-co** option – see *Xserver(1)*.)

The options are:-

-bp[:0] colour

Specifies the colour of the black pixel (pixel 1.)

-wp[:0] colour

Specifies the colour of the white pixel. In both these cases, although the option only affects a 1bpp display, the colours may be other than black or white. This allows (for example) a colour monitor to be used to produce a green monochrome display as well as allowing simple reversal of black and white on the display. In high resolution display modes the video output is strictly monochrome – if a colour is specified for either the black or the white pixel the device driver will choose black or white according to the intensity of the colour.

-frame[:0] colour

This specifies the colour of the border round the display. Again, colour borders may be used on 1bpp displays (and the colour need not be the same as the black or white pixel colour.) On high resolution displays the effect of changing the border colour is to produce black and white bands round the display. This can be used to produce a grey border (try '555') but that is about all.

Some additional options are provided to control the keyboard autorepeat and click – these options allow the delay and duration to be set on the command line.

-rimilliseconds

This sets the delay in milliseconds before keyboard autorepeat starts – the key must be held down for the given time before the first repeat depression is delivered. See below for more details of which keys autorepeat by default. A value of 0 will restore the default (300 milliseconds.)

-rdmilliseconds

This sets the delay in milliseconds between the delivery of successive autorepeat depressions. A value of zero restores the default (50 milliseconds.)

-cdmilliseconds

This sets the duration of the keyclick "beep". A value of zero restores the default (8 milliseconds.)

It is possible to control whether the server uses a software or hardware cursor:-

-hard Use the hardware cursor. The cursor is automatically used in low and medium resolution modes, and a software cursor used in high resolution mode. Using the hardware cursor in high resolution mode may increase performance slightly, but the cursor looks awful.

-soft Use the software cursor. If you have applications which use particularly large cursors (more than 32 bits wide) you will need to select a software cursor or they will be truncated - if you have not changed the cursor fonts you certainly do not need to worry about this.

The remaining options are intended for use when debugging the server, although the **+flog** option can be used to find out about font, utilisation as suggested above.

+flog Turn on font logging. Each time a font is read from the disc a message is written to the server log file.

-flog Turn off font logging (the default.)

-sched #

This option allows the round robin scheduler timeout to be set. The timeout governs the number of requests from a single client which are executed before input is read from another client. Normally this should not be specified. There is no guarantee that it will have any particular effect. However it can be useful to set the number to '1' when using applications which submit expensive graphics requests without waiting for input from the server.

Other options are described under *Xserver(1)*.

KEYBOARD

Two varieties of ARM keyboard are supported - the A440/A680 (Archimedes) keyboard and the A500 keyboard. The Archimedes keyboard is mechanically similar to the IBM 101 keyboard, the A500 keyboard is significantly different. Both keyboards have the standard typewriter keys, cursor keys, a numeric keypad and at least 10 function keys (the Archimedes has 12.) The keys are mapped onto the standard X11 keycodes in an appropriate way.

Both keyboards generate up and down transitions for all keys but only have 2 key rollover for most keys. Certain keys are handled individually and, because these do not interfere with the 2 key rollover of the remainder, it is strongly recommended that these be used as modifier keys. The default modifier map uses these keys, as follows:-

<i>A500</i>	<i>Archimedes</i>	<i>X11 key</i>	<i>X11 modifier</i>
Left Shift	Left Shift	XK_Shift_L	Shift
Right Shift	-	XK_Shift_L	Shift
-	Right Shift	XK_Shift_R	Shift
CMD	-	XK_Control_L	Control
-	Left Control	XK_Control_L	Control
-	Right Control	XK_Control_R	Control
LOOKS	-	XK_Alt_L	Mod1
-	Left Alt	XK_Alt_L	Mod1
-	Right Alt	XK_Alt_R	Mod1
-	Print	XK_Print	Mod2
-	Home	XK_Home	Mod3
CAPS LOCK	CapsLock	XK_Caps_Lock	LockMask
-	Num Lock	XK_Num_Lock	Mod4
-	Scroll Lock	XK_Pause	Mod5

Although the A500 keyboard has both left and right shift keys these are connected electrically thus only a left shift is generated.

When the keyboard click is switched on (this is probably not a good idea) it only sounds on down transitions of non-modifier keys. All keys auto-repeat, the keyboard click does not sound on auto-repeats.

Notice that all the keys which have LEDs are in the above list. *Xarm* does not allow control of the keyboard LEDs, instead up and down transitions are transmitted on each alternate key press/release and the LED is always kept in step – ie the LED is on after a key down, off after a key up.

The **Home** and **Print** keys on the Archimedes keyboard are both modifier keys – this is simply because they are shifting keys in the hardware, there is no need to treat them specially.

Both keyboards have a keypad. These are treated as would be expected. The keys generate **XK_KP_...** key codes, except that the % key (on the A500) and the # key have no corresponding keypad code. These generate '%' and '#'. In addition the four keys on top on the top row of the keypad generate **XK_KP_Fx** codes when shifted, except for the **Num Lock** key on the Archimedes keyboard which just generates **XK_Num_Lock** (and thus **XK_KP_F1** is not available from the Archimedes keyboard.)

The */dev/kbd* driver on both keyboards takes one key for use in switching virtual terminals – this is the **Break** key on the Archimedes keyboard and the **Help** key on the A500 keyboard. Application programs never see these keys.

Both keyboards have cursor keys and function keys. Normal special control keys (**TAB**, **Return**, etc.) are also present. The remaining keys generate codes as follows:-

<i>Keyboard</i>	<i>Key</i>	<i>Code</i>	<i>Notes</i>
A440	International Currency Symbol	XK_currency	
Both	Pound Sterling	XK_sterling	
A500	Copyright	XK_copyright	
Both	Leftwards Delete	XK_BackSpace	
A440	Delete	XK_Delete	
A500	Rightwards Delete	XK_Delete	
A440	Insert	XK_Insert	
A440	Copy	XK_Select	(*)
A440	Page Up	XK_Prior	(*)
A440	Page Down	XK_Next	(*)
A500	MENU	XK_Menu	
A500	AGAIN	XK_Redo	(*)

Encodings marked '*' are arbitrary decisions – if a subsequent version of X11 supplies better codes these will probably be used unless the existing encodings become established in ARM applications. The same applies to the '#' and '%' keys on the keypad.

ENVIRONMENT

XDEVICE

If present, and if no explicit **-dev** options are given, specifies the display device to use instead of */dev/fb*.

XKBDDEV

If present, and if no explicit **-kbd** options are given, specifies the keyboard/mouse device to use instead of */dev/kbd*.

FILES

<i>/dev/fb</i>	default frame buffer device
<i>/dev/kbd</i>	default keyboard/mouse device
<i>/usr/adm/X*msgs</i>	error and warning messages from the X server
<i>/usr/lib/X11/*</i>	database files
<i>/usr/lib/X11/fonts</i>	default font files

SEE ALSO

Xserver(1), xinit(1), X(1)

BUGS

Bugs in the *Xarm* server or the Acorn X11 system should be reported to unixbugs@acorn.co.uk - bugs are more likely to be fixed if submitted with a concise example which shows the problem on a system consisting solely of Acorn hardware and software. The X11 system should work correctly with any other X11 software which obeys the X11 protocol and uses TCP/IP or unix domain streams for communication (the server does not support DecNet connections.)

Current features of the server are as follows:-

- 1 The colour modes are significantly slower than the monochrome modes - even taking into account the fact that they have to write as much as 4 times as much data per pixel. As a consequence certain graphics demos (in particular the 'plaid' demo) can render the system almost unusable while they are running. To do this the demo must be continuously producing output - it is therefore almost certainly not interacting with the user and is therefore unlikely to be doing anything useful. Avoid running such demos on the medium resolution colour system!
- 2 The hardware cursor is not used in any mode at present, despite the fact that the options exist to select it.

AUTHORS

Acorn Computers Ltd
JB, SH

NAME

`xterm` – terminal emulator for X

SYNOPSIS

`xterm [-toolkitoption ...] [-option ...]`

DESCRIPTION

The `xterm` program is a terminal emulator for the X Window System. It provides DEC VT102 and Tektronix 4014 compatible terminals for programs that can't use the window system directly. If the underlying operating system supports terminal resizing capabilities (for example, the SIGWINCH signal in systems derived from 4.3bsd), `xterm` will use the facilities to notify programs running in the window whenever it is resized.

The VT102 and Tektronix 4014 terminals each have their own window so that you can edit text in one and look at graphics in the other at the same time. To maintain the correct aspect ratio (height/width), Tektronix graphics will be restricted to the largest box with a 4014's aspect ratio that will fit in the window. This box is located in the upper left area of the window.

Although both windows may be displayed at the same time, one of them is considered the "active" window for receiving keyboard input and terminal output. This is the window that contains the text cursor and whose border highlights whenever the pointer is in either window. The active window can be chosen through escape sequences, the "Modes" menu in the VT102 window, and the "Tektronix" menu in the 4014 window.

OPTIONS

The `xterm` terminal emulator accepts all of the standard X Toolkit command line options along with the additional options listed below (if the option begins with a '+' instead of a '-', the option is restored to its default value):

-132 Normally, the VT102 DECCOLM escape sequence that switches between 80 and 132 column mode is ignored. This option causes the DECCOLM escape sequence to be recognized, and the `xterm` window will resize appropriately.

-b *number*

This option specifies the size of the inner border (the distance between the outer edge of the characters and the window border) in pixels. The default is 2.

-cr *color*

This option specifies the color to use for text cursor. The default is to use the same foreground color that is used for text.

-cu

This option indicates that `xterm` should work around a bug in the `curses(3x)` cursor motion package that causes the `more(1)` program to display lines that are exactly the width of the window and are followed by line beginning with a tab to be displayed incorrectly (the leading tabs are not displayed).

+cu

This option indicates that that `xterm` should not work around the `curses(3x)` bug mentioned above.

-e *program [arguments ...]*

This option specifies the program (and its command line arguments) to be run in the `xterm` window. The default is to start the user's shell. **This must be the last option on the command line.**

-fb *font*

This option specifies a font to be used when displaying bold text. This font must be the same height and width as the normal font. If only one of the normal or bold fonts is specified, it will be used as the normal font and the bold font will be produced by overstriking this font. The default bold font is "vtbold."

-j

This option indicates that `xterm` should do jump scrolling. Normally, text is scrolled one line at a time; this option allows `xterm` to move multiple lines at a time so that it doesn't fall as far behind. Its use is strongly recommended since it make `xterm` much faster

when scanning through large amounts of text. The VT100 escape sequences for enabling and disabling smooth scroll as well as the "Modes" menu can be used to turn this feature on or off.

- +j This option indicates that *xterm* should not do jump scrolling.
- l This option indicates that *xterm* should send all terminal output to a log file as well as to the screen. This option can be enabled or disabled using the "xterm X11" menu.
- +l This option indicates that *xterm* should not do logging.
- lf *filename*
This option specifies the name of the file to which the output log described above is written. If *file* begins with a pipe symbol (`|`), the rest of the string is assumed to be a command to be used as the endpoint of a pipe. The default filename is "XtermLog.XXXXX" (where XXXXX is the process id of *xterm*) and is created in the directory from which *xterm* was started (or the user's home directory in the case of a login window).
- ls This option indicates that shell that is started in the *xterm* window be a login shell (i.e. the first character of `argv[0]` will be a dash, indicating to the shell that it should read the user's `.login` or `.profile`).
- +ls This option indicates that the shell that is started should not be a login shell (i.e. it will be normal "subshell").
- mb This option indicates that *xterm* should ring a margin bell when the user types near the right end of a line. This option can be turned on and off from the "Modes" menu.
- +mb This option indicates that margin bell should not be rung.
- ms *color*
This option specifies the color to be used for the pointer cursor. The default is to use the foreground color.
- nb *number*
This option specifies the number of characters from the right end of a line at which the margin bell, if enabled, will ring. The default is 10.
- rw This option indicates that reverse-wraparound should be allowed. This allows the cursor to back up from the leftmost column of one line to the rightmost column of the previous line. This is very useful for editing long shell command lines and is encouraged. This option can be turned on and off from the "Modes" menu.
- +rw This option indicates that reverse-wraparound should not be allowed.
- s This option indicates that *xterm* may scroll asynchronously, meaning that the screen does not have to be kept completely up to date while scrolling. This allows *xterm* to run faster when network latencies are very high and is typically useful when running across a very large internet or many gateways.
- +s This option indicates that *xterm* should scroll synchronously.
- sb This option indicates that some number of lines that are scrolled off the top of the window should be saved and that a scrollbar should be displayed so that those lines can be viewed. This option may be turned on and off from the "Modes" menu.
- +sb This option indicates that a scrollbar should not be displayed.
- si This option indicates that output to a window should not automatically reposition the screen to the bottom of the scrolling region. This option can be turned on and off from the "Modes" menu.
- +si This option indicates that output to a window should cause it to scroll to the bottom.

- sk** This option indicates that pressing a key while using the scrollbar to review previous lines of text should cause the window to be repositioned automatically in the normal position at the bottom of the scroll region.
- +sk** This option indicates that pressing a key while using the scrollbar should not cause the window to be repositioned.
- sl number**
This option specifies the number of lines to save that have been scrolled off the top of the screen. The default is 64.
- t** This option indicates that *xterm* should start in Tektronix mode, rather than in VT102 mode. Switching between the two windows is done using the "Modes" menus.
- +t** This option indicates that *xterm* should start in VT102 mode.
- vb** This option indicates that a visual bell is preferred over an audible one. Instead of ringing the terminal bell whenever a Control-G is received, the window will be flashed.
- +vb** This option indicates that a visual bell should not be used.
- C** This option indicates that this window should be receive console output. This is not supported on all systems.
- L** This option indicates that *xterm* was started by *init*. In this mode, *xterm* does not try to allocate a new pseudoterminal as *init* has already done so. In addition, the system program *getty* is run instead of the user's shell. **This option should never be used by users when starting terminal windows.**
- Scn** This option specifies the last two letters of the name of a pseudoterminal to use in slave mode. This allows *xterm* to be used as an input and output channel for an existing program and is sometimes used in specialized applications.

The following command line arguments are provided for compatibility with older versions. They may not be supported in the next release as the X Toolkit provides standard options that accomplish the same task.

- %geom** This option specifies the preferred size and position of the Tektronix window. It is shorthand for specifying the "**tekGeometry*" resource.
- #geom** This option specifies the preferred position of the icon window. It is shorthand for specifying the "**iconGeometry*" resource.
- T string**
This option specifies the title for *xterm*'s windows. It is equivalent to **-title**.
- nstring** This option specifies the icon name for *xterm*'s windows. It is shorthand for specifying the "**iconName*" resource.
- r** This option indicates that reverse video should be simulated by swapping the foreground and background colors. It is equivalent to **-reversevideo** or **-rv**.
- w number**
This option specifies the width in pixels of the border surrounding the window. It is equivalent to **-borderwidth** or **-bw**.

The following standard X Toolkit command line arguments are commonly used with *xterm*:

- bg color**
This option specifies the color to use for the background of the window. The default is "white."
- bd color**
This option specifies the color to use for the border of the window. The default is "black."

- bw *number***
This option specifies the width in pixels of the border surrounding the window.
- fg *color***
This option specifies the color to use for displaying text. The default is "black".
- fn *font***
This option specifies the font to be used for displaying normal text. The default is "vtsingle."
- name *name***
This option specifies the application name under which resource are to be obtained, rather than the default executable file name.
- rv**
This option indicates that reverse video should be simulated by swapping the foreground and background colors.
- geometry *geometry***
This option specifies the preferred size and position of the VT102 window; see *X(1)*;
- display *display***
This option specifies the X server to contact; see *X(1)*.
- xrm *resourcestring***
This option specifies a resource string to be used. This is especially useful for setting resources that do not have separate command line options.

X DEFAULTS

The program understands all of the core X Toolkit resource names and classes as well as:

- name (class Name)**
Specifies the name of this instance of the program. The default is "xterm."
- iconGeometry (class IconGeometry)**
Specifies the preferred size and position of the application when iconified. It is not necessarily obeyed by all window managers.
- title (class Title)**
Specifies a string that may be used by the window manager when displaying this application.

The following resources are specified as part of the "vt100" widget (class "VT100"):

- font (class Font)**
Specifies the name of the normal font. The default is "vtsingle."
- boldFont (class Font)**
Specifies the name of the bold font. The default is "vtbold."
- c132 (class C132)**
Specifies whether or not the VT102 DECCOLM escape sequence should be honored. The default is "false."
- curses (class Curses)**
Specifies whether or not the last column bug in cursor should be worked around. The default is "false."
- background (class Background)**
Specifies the color to use for the background of the window. The default is "white."
- foreground (class Foreground)**
Specifies the color to use for displaying text in the window. Setting the class name instead of the instance name is an easy way to have everything that would normally appear in the "text" color change color. The default is "black."

- cursorColor** (class **Foreground**)
Specifies the color to use for the text cursor. The default is "black."
- geometry** (class **Geometry**)
Specifies the preferred size and position of the VT102 window.
- tekGeometry** (class **Geometry**)
Specifies the preferred size and position of the Tektronix window.
- internalBorder** (class **BorderWidth**)
Specifies the number of pixels between the characters and the window border. The default is 2.
- jumpScroll** (class **JumpScroll**)
Specifies whether or not jump scroll should be used. The default is "false".
- logFile** (class **Logfile**)
Specifies the name of the file to which a terminal session is logged. The default is "XtermLog.XXXXX" (where XXXXX is the process id of *xterm*).
- logging** (class **Logging**)
Specifies whether or not a terminal session should be logged. The default is "false."
- logInhibit** (class **LogInhibit**)
Specifies whether or not terminal session logging should be inhibited. The default is "false."
- loginShell** (class **LoginShell**)
Specifies whether or not the shell to be run in the window should be started as a login shell. The default is "false."
- marginBell** (class **MarginBell**)
Specifies whether or not the bell should be run when the user types near the right margin. The default is "false."
- multiScroll** (class **MultiScroll**)
Specifies whether or not asynchronous scrolling is allowed. The default is "false."
- nMarginBell** (class **Column**)
Specifies the number of characters from the right margin at which the margin bell should be run, when enabled.
- pointerColor** (class **Foreground**)
Specifies the color of the pointer. The default is "black."
- pointerShape** (class **Cursor**)
Specifies the name of the shape of the pointer. The default is "xterm."
- reverseVideo** (class **ReverseVideo**)
Specifies whether or not reverse video should be simulated. The default is "false."
- reverseWrap** (class **ReverseWrap**)
Specifies whether or not reverse-wraparound should be enabled. The default is "false."
- saveLines** (class **SaveLines**)
Specifies the number of lines to save beyond the top of the screen when a scrollbar is turned on. The default is 64.
- scrollBar** (class **ScrollBar**)
Specifies whether or not the scrollbar should be displayed. The default is "false."
- scrollInput** (class **ScrollCond**)
Specifies whether or not output to the terminal should automatically cause the scrollbar to go to the bottom of the scrolling region. The default is "true."

scrollKey (class **ScrollCond**)

Specifies whether or not pressing a key should automatically cause the scrollbar to go to the bottom of the scrolling region. The default is "false."

signalInhibit (class **SignalInhibit**)

Specifies whether or not the entries in the "xterm X11" menu for sending signals to *xterm* should be disallowed. The default is "false."

tekInhibit (class **TekInhibit**)

Specifies whether or not Tektronix mode should be disallowed. The default is "false."

tekStartup (class **TekStartup**)

Specifies whether or not *xterm* should start up in Tektronix mode. The default is "false."

visualBell (class **VisualBell**)

Specifies whether or not a visible bell (i.e. flashing) should be used instead of an audible bell when Control-G is received. The default is "false."

The following resources are specified as part of the "tek4014" widget (class "Tek4014"):

width (class **Width**)

Specifies the width of the Tektronix window in pixels.

height (class **Height**)

Specifies the height of the Tektronix window in pixels.

The following resources are specified as part of the "menu" widget:

menuBorder (class **MenuBorder**)

Specifies the size in pixels of the border surrounding menus. The default is 2.

menuFont (class **Font**)

Specifies the name of the font to use for displaying menu items.

menuPad (class **MenuPad**)

Specifies the number of pixels between menu items and the menu border. The default is 3.

EMULATIONS

The VT102 emulation is fairly complete, but does not support the blinking character attribute nor the double-wide and double-size character sets. *Termcap*(5) entries that work with *xterm* include "xterm", "vt102", "vt100" and "ansi", and *xterm* automatically searches the termcap file in this order for these entries and then sets the "TERM" and the "TERMCAP" environment variables.

Many of the special *xterm* features (like logging) may be modified under program control through a set of escape sequences different from the standard VT102 escape sequences. (See the "Xterm Control Sequences" document.)

The Tektronix 4014 emulation is also fairly good. Four different font sizes and five different lines types are supported. The Tektronix text and graphics commands are recorded internally by *xterm* and may be written to a file by sending the COPY escape sequence (or through the Tektronix menu; see below). The name of the file will be "COPYyy-MM-dd.hh:mm:ss", where yy, MM, dd, hh, mm and ss are the year, month, day, hour, minute and second when the COPY was performed (the file is created in the directory *xterm* is started in, or the home directory for a login *xterm*).

POINTER USAGE

Once the VT102 window is created, *xterm* allows you to select text and copy it within the same or other windows.

The selection functions are invoked when the pointer buttons are used with no modifiers, and when they are used with the "shift" key.

Pointer button one (usually left) is used to save text into the cut buffer. Move the cursor to beginning of the text, and then hold the button down while moving the cursor to the end of the region and releasing the button. The selected text is highlighted and is saved in the global cut buffer when the button is released. Double-clicking selects by words. Triple-clicking selects by lines. Quadruple-clicking goes back to characters, etc. Multiple-click is determined by the time from button up to button down, so you can change the selection unit in the middle of a selection.

Pointer button two (usually middle) 'types' (pastes) the text from the cut buffer, inserting it as keyboard input.

Pointer button three (usually right) extends the current selection. (Without loss of generality, that is you can swap "right" and "left" everywhere in the rest of this paragraph...) If pressed while closer to the right edge of the selection than the left, it extends/contracts the right edge of the selection. If you contract the selection past the left edge of the selection, *xterm* assumes you really meant the left edge, restores the original selection, then extends/contracts the left edge of the selection. Extension starts in the selection unit mode that the last selection or extension was performed in; you can multiple-click to cycle through them.

By cutting and pasting pieces of text without trailing new lines, you can take text from several places in different windows and form a command to the shell, for example, or take output from a program and insert it into your favorite editor. Since the cut buffer is globally shared among different applications, you should regard it as a 'file' whose contents you know. The terminal emulator and other text programs should be treating it as if it were a text file, i.e. the text is delimited by new lines.

The scroll region displays the position and amount of text currently showing in the window (highlighted) relative to the amount of text actually saved. As more text is saved (up to the maximum), the size of the highlighted area decreases.

Clicking button one with the pointer in the scroll region moves the adjacent line to the top of the display window.

Clicking button three moves the top line of the display window down to the pointer position.

Clicking button two moves the display to a position in the saved text that corresponds to the pointer's position in the scrollbar.

Unlike the VT102 window, the Tektronix window does not allow the copying of text. It does allow Tektronix GIN mode, and in this mode the cursor will change from an arrow to a cross. Pressing any key will send that key and the current coordinate of the cross cursor. Pressing button one, two, or three will return the letters 'l', 'm', and 'r', respectively. If the 'shift' key is pressed when a pointer button is pressed, the corresponding upper case letter is sent. To distinguish a pointer button from a key, the high bit of the character is set (but this bit is normally stripped unless the terminal mode is RAW; see *ty(4)* for details).

MENUS

Xterm has three different menus, named *xterm*, *Modes*, and *Tektronix*. Each menu pops up under the correct combinations of key and button presses. Most menus are divided into two sections, separated by a horizontal line. The top portion contains various modes that can be altered. A check mark appears next to a mode that is currently active. Selecting one of these modes toggles its state. The bottom portion of the menu are command entries; selecting one of these performs the indicated function.

The *xterm* menu pops up when the "control" key and pointer button one are pressed in a window. The modes section contains items that apply to both the VT102 and Tektronix windows. Notable entries in the command section of the menu are the **Continue**, **Suspend**, **Interrupt**, **Hangup**, **Terminate** and **Kill** which sends the SIGCONT, SIGTSTP, SIGINT, SIGHUP, SIGTERM and SIGKILL signals, respectively, to the process group of the process running under *xterm* (usually the shell). The **Continue** function is especially useful if the user has accidentally typed CTRL-Z, suspending the process.

The **Modes** menu sets various modes in the VT102 emulation, and is popped up when the "control" key and pointer button two are pressed in the VT102 window. In the command section of this menu, the soft reset entry will reset scroll regions. This can be convenient when some program has left the scroll regions set incorrectly (often a problem when using VMS or TOPS-20). The full reset entry will clear the screen, reset tabs to every eight columns, and reset the terminal modes (such as wrap and smooth scroll) to their initial states just after *xterm* has finished processing the command line options. The **Tektronix** menu sets various modes in the Tektronix emulation, and is popped up when the "control" key and pointer button two are pressed in the Tektronix window. The current font size is checked in the modes section of the menu. The **PAGE** entry in the command section clears the Tektronix window.

OTHER FEATURES

Xterm automatically highlights the window border and text cursor when the pointer enters the window (selected) and unhighlights them when the pointer leaves the window (unselected). If the window is the focus window, then the window is highlighted no matter where the pointer is.

In VT102 mode, there are escape sequences to activate and deactivate an alternate screen buffer, which is the same size as the display area of the window. When activated, the current screen is saved and replaced with the alternate screen. Saving of lines scrolled off the top of the window is disabled until the normal screen is restored. The *termcap*(5) entry for *xterm* allows the visual editor *vi*(1) to switch to the alternate screen for editing, and restore the screen on exit.

In either VT102 or Tektronix mode, there are escape sequences to change the name of the windows and to specify a new log file name.

ENVIRONMENT

Xterm sets the environment variables "TERM" and "TERMCAP" properly for the size window you have created. It also uses and sets the environment variable "DISPLAY" to specify which bit map display terminal to use. The environment variable "WINDOWID" is set to the X window id number of the *xterm* window.

SEE ALSO

resize(1), X(1), pty(4), tty(4)
 "Xterm Control Sequences"

BUGS

Xterm will hang forever if you try to paste too much text at one time. It is both producer and consumer for the pty and can deadlock.

Variable-width fonts are not handled reasonably.

This program still needs to be rewritten. It should be split into very modular sections, with the various emulators being completely separate widgets that don't know about each other. Ideally, you'd like to be able to pick and choose emulator widgets and stick them into a single control widget.

The focus is considered lost if some other client (e.g., the window manager) grabs the pointer; it is difficult to do better without an addition to the protocol.

There needs to be a dialog box to allow entry of log file name and the COPY file name.

Many of the options are not resettable after *xterm* starts.

This manual page is too long. There should be a separate users manual defining all of the non-standard escape sequences.

All programs should be written to use X directly; then we could eliminate this program.

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology.

See *X(1)* for a full statement of rights and permissions.

AUTHORS

Far too many people, including:

Loretta Guarino Reid (DEC-UEG-WSL), Joel McCormack (DEC-UEG-WSL), Terry Weissman (DEC-UEG-WSL), Edward Moy (Berkeley), Ralph R. Swick (MIT-Athena), Mark Vandevoorde (MIT-Athena), Bob McNamara (DEC-MAD), Jim Gettys (MIT-Athena), Bob Scheifler (MIT X Consortium), Doug Mink (SAO), Steve Pitschke (Stellar), Ron Newman (MIT-Athena), Jim Fulton (MIT X Consortium)

Index

A

access 12-14, 33-35
 displaying 33-34
 execute 14, 34
 group 12-13, 33
 network 117-120
 read 14, 34
 user 13, 33
 write 14, 34
adb 180
adfscat 155
adfscp 155, 156
adfsls 155
adfsrc 155
alert 196
apropos 196
as 180
at 184
atq 196
atrm 196
awk 183
awm 196

B

bc 184
Berkeley networking commands
 117-125
bitmap 196

C

cal 184
calendar 184
cat 27
cb 180
cc 180
cd 23
cheapernet 110
checkeq 182
chgrp 197
chmod 34-35, 57
clear 197
CLI *see* command line interpreter
client programs *see* X Window
 System
cmp 184
col 182
colcrt 182
comm 184
command 4-5
 arguments 20-21, 36
 executing sequence 49
 format 20
 options 20-21
 summaries 38-39, 59-60, 74, 80,
 106-107, 107, 126-127,
 157-159, 180-184, 208
 UNIX 16
command line interpreter 16

- communication
 - interactive 147–150
- communications
 - serial line 134–135
- compress 197
- connect 132
- cp 29
- csh 197
- CWD *see* directory, current working

D

- date 197
- dbx 180
- dc 197
- deroff 181
- df 197
- diction 181
- diff 184
- diff3 184
- directory 7
 - /bin 7
 - /dev 8
 - /etc 8, 118
 - /lib 8
 - /tmp 8
 - /usr 8
 - /bin 8
 - /lib 8
 - /man 36
 - /users 8–9
- changing 23
- creating 27
- current working 9, 23–24
- deleting 30
- home 9, 23
- listing 28
- moving 29
- parent 24

- renaming 29

- disc
 - formatting floppy 152
- du 197

E

- echo 42, 49–50
- Econet 110
- ed 64–80
 - adding text 71
 - appending text 64–65
 - command summaries 74, 80, 205
 - commands 67
 - current line 69–70
 - deleting text 71
 - editing text 70–72
 - entering 64
 - error message 67, 191
 - leaving 66–67
 - moving text 71
 - printing selected lines 67–70
 - replacing text 71–72, 76–77
 - saving text 66
 - searching for text 74–75
 - special characters 77–79
 - undoing changes 72
- edit 108
- eqn 181
- errors
 - editing 191–192
 - in RISC iX commands 188–190
 - networking 193–194
- ethernet 110
- ex
 - command summaries 107, 208
- exit 121
- expand 183
- explain 181

export 42, 43

F

fc 198

ffd 152

file

ADFS to RISC iX 155-156

command 198

copying 29

copying to remote workstation
129, 132

creating 25-26, 64, 81

deleting 30

displaying 27

hidden 26, 42, 44

moving 28

MS-DOS to RISC iX 154-155

printing 31-33

protection *see* access

renaming 28

special 7

transferring between
workstations 131, 132-
133, 152-156

file system 6-11

file types in 7

structure 6, 22

filename 26

characters 26

generation 45-48

filter 56

find 184

floppy disc

commands 210

floppy discs

file transfer 152

formatting 152

flpop 198

from 159

ftp 129, 130-131

command summary 157

G

gateway 113

get 131, 132

grep 56, 183

gs 198

H

head 183

help

on-line 37

host files 118

hostid 198

hostname 118

I

ico 198

inituser 198

input/output 15

redirecting 51-55

standard 51-53

install 198

internet 110

K

kermit 134

kernel 4-5

kill 198

L

LAN 110-111

ld 180

- learn 37, 108
- lex 180
- lint 180
- ln 198
- .login 43, 104, 105
- look 183
- lpq 32
- lpr 31
- lprm 32
- ls 16, 28, 43, 45-47, 55

M

- mail 53, 136-160
 - aborting send 138
 - carbon copies 146
 - command summary 158
 - commands 209
 - customising 145-146
 - deleting messages 142
 - help 140
 - inserting file into message 144
 - looking at messages from
 - outside mail 145
 - message header 139
 - on WAN 146
 - quitting 144-145
 - reading 138-140, 141
 - replying to 142-144
 - saving 141
 - security 138
 - sending 136-137
 - sorting 141-142
 - subject header 145
 - system mailbox 136
 - tilde escape commands 137, 140, 158
 - to unknown user 138
 - unread 145

- using vi 137-138
- make 180
- man 36, 55
- manual pages 35-37, 189
- message
 - system 151
- metacharacters 44-48
 - quoting 49-50
- mkdir 27
- mkruler 199
- modem 135
- more 27
- msdoscat 154
- msdoscp 154
- msdosls 154
- msdosrm 154
- muncher 199
- mv 24, 28-29, 47

N

- neqn 182
- network 109
 - access 117-120
 - Berkeley networking commands 117-125
 - client 115-116
 - command summary 126-127
 - copying between workstations 124-125
 - file system *see* NFS
 - id 117
 - information 122
 - log in to remote workstation 120-121
 - log out from remote workstation 121
 - remote commands 123
 - sending messages 123

- server 115–116
- setting up environment 117–120
- transparency 115
- wide area *see* WAN
- newline
 - hidden 50
- NFS 114–127
- nice 199
- nroff 181

O

- od 184
- open 130
- output *see* input/output

P

- parameter
 - in shell script 58
- passwd 199
- password 118, 120
- path
 - search 43
- pathname 10–11, 23
 - abbreviating 24
 - absolute 23
 - relative 25
- permission *see* access
- pipes 54–55
- plaid 200
- pr 183
- printer
 - queue 32
 - spooling queue 32
- process
 - id number 48
 - status 48
- .profile 42–43, 104, 105
- protocol

- communications 113
 - internet 110
- ps 48
- psrff 181
- put 131, 132
- puzzle 200
- pwd 23

R

- rcp 117, 124–125
- reborder 200
- refer 181
- rev 183
- .rhosts 119
- RISC iX 1
- rlogin 117, 120–121
- rm 30–31, 47
 - options 31
- rmdir 30
- root 11–12
- rsh 117, 123
- rup 117, 122
- rusers 117, 122
- rwall 117, 123

S

- script 200
- security 11–12
- sed 183
- sh 57
- shell 4–5, 15–16
 - as programming language 56–58
 - Bourne 41–42
 - C 41–42
 - login 42–44
 - script 57

- UNIX 204
- showsnf 200
- sleep 200
- sort 56, 183
- spell 53, 181
- split 183
- stty 200
- style 181
- su 201
- sum 184
- super-user *see* root

T

- tail 183
- talk 148–149, 159
 - blocking messages 150
 - quitting 149
 - replying to call 149
- tar 152, 153
- tbl 181
- tee 201
- telnet 129, 133–134
 - command summary 157
- tftp 129, 130, 132–133
 - command summary 157
- time 201
- tip 134
- touch 201
- tr 183
- troff 181
- tset 201
- twm 163

U

- uemacs 108
- unexpand 183

- uniq 184
- unit 201
- UNIX 3–4
 - components 4–5
- user
 - trusted 119–120
- users 147
- utilities
 - data manipulation 183
 - floppy disc 152–156
 - miscellaneous 184
 - programming 180
 - text preparation 181–182
- uucp 113, 134
- uux 201
- uwm *see* X Window Manager

V

- variable
 - shell 43
- vi 81–107
 - adding text 88–90
 - command summaries 96, 106–107, 206–207
 - editing text 88–95
 - entering 82
 - error messages 192
 - executing shell commands 102
 - global changes 100–101
 - high-level objects 97
 - inserting text 83–84
 - joining lines 98
 - leaving 85
 - line editing with ex 100–101
 - macro commands 105
 - moving around in 86–88
 - moving text 93
 - multiple buffers 99–100

- multiple files 102
- options 103–104
- repeating last command 98
- replacing text 90–93, 93
- saving text 84
- searching for text 97–98
- splitting lines 98
- undoing changes 95

W

- w 147
- wall 151, 159
- WAN 112–113
- wc 54–55, 181
- whatis 201
- whereis 201
- which 202
- who 56, 147
- whoami 202
- wildcards 44
- wm 202
- workstation
 - local 109
 - remote 109
- wradfs 155
- write 149–150, 159
 - blocking messages 150
 - replying to message 149–150
- wrmsdos 154

X

- X Window Manager
 - changing stacking order 170
 - focus window 172
 - keyboard short-cuts 174
 - moving window 169
 - opening new window 168

- Preferences menu 173
- redrawing screen 169
- redrawing window 169
- reducing window to icon 171
- resizing window 170
- restarting 172
- running applications 177
- running clients 176
- start-up file 167
- starting 166
- WindowOps menu 167–172

X Window System

- client programs 161, 162–163
- display 161
- focusing 164
- loading clients 175
- login xterm window 164
- root window 164
- running non-X applications 175
- server 161
- starting 164–166
- stopping 177
- terminal emulator 162

- Xarm 161
- xbiff 163
- xcalc 163
- xcalendar 163
- xclock 163
- xdpr 163
- xedit 163
- xfd 163
- xhost 163
- xinit 165
- Xlib 161
- xload 202
- xlsfonts 163
- xman 37, 163
- xmodmap 163
- xmore 163

xperfmon 163

xpr 163, 202

xprkbd 163

xprop 163

xrdb 202

xrefresh 202

xset 163

xsetroot 163

xterm 162, 164

xwd 163

xwininfo 163

xwud 163

Y

yacc 180

yellow pages 116

yorn 203

