ConvexOS Primer

First Edition

CONVEX COMPUTER CORPORATION

606

CONVEX Computer Corporation
3000 Waterview Parkway
P.O. Box 833851
Richardson, TX 75083-3851
United States of America
(214)497-4000

ConvexOS Primer

Order No. DSW-133

First Edition
November 1991

CONVEX Press
Richardson, Texas
United States of America

ConvexOS Primer

Order No. DSW-133

Printed in the United States of America

Revision information for

ConvexOS Primer

Edition	Document No.	Description
First	710-012130-000	Published November 1991. Replaces *CONVEX UNIX Primer* (710-000221-202).

Contents

Using this primer

1 Getting started and finding help

2 Your work environment

3 Working with the C shell: commands, processes, and jobs

4 Text editors: creating and altering text files

5 Organizing and protecting your files

6 Printing files

7 Communicating with other users

Appendix A New to computers? Read this.

Appendix B Reporting problems

Figures

Tables

Using this primer

Is ConvexOS new to you? The *ConvexOS Primer* teaches the essentials—a selected core of concepts and commands that quickly get you comfortable with using ConvexOS—without reflecting any specific release of ConvexOS.

This primer is for anyone new to ConvexOS. Whether you are a first-time computer user or an experienced systems programmer, this primer contains basic information that you will need. To assist those new to computers, Appendix A introduces computer basics and terminology.

Questions such as the ones listed here are addressed within the chapters of this primer:

- How do I start a work session on my CONVEX system?
- How do I enter a command?
- How do I access online help?
- How do I create directories and files?
- How do I print a file?
- How do I protect my files from unauthorized use?
- How do I communicate with other users on my system?

Notational conventions

This section defines the notational conventions used within this primer.

Command syntax

Throughout this primer you will be shown commands that you can enter. The elements of a command are presented in the form

command [*option*] *argument*

where each element is:

command	Typed as it appears.
option	Optional characters (indicated by brackets), substituted with an appropriate value, that modify the way in which the command works.
argument	Objects on which a command works, usually substituted with a file name or directory name.

Also used to define a command's syntax are:

[ ]	Square brackets flag an option or argument as optional; brackets are not typed as part of the command.
[...]	Ellipsis following an option or argument indicates that more than one value may be entered.

An example that combines all of the above:

command [*option...*] [*argument*]

where the name of the command must be typed exactly as it appears, more than one or no option may be specified, and an argument is not imperative.

General conventions

In general, the following conventions are used in this primer:

- **`Boldface courier`** (**monospaced**) designates user input in examples.
- *Italic* designates:
  - Optional command options and arguments
  - New and important terms
  - Document titles
- `Constant-width font` designates:
  - Command names, options, and arguments
  - Examples
- **KEYCAP** indicates keyboard keys you press. For example, **RETURN** refers to the carriage return key. Words separated by a hyphen indicate two keys that you must press simultaneously. For example, **CTRL-x** indicates that you must press and hold down the **CTRL** key as you press the **x** key.
- Notes appear in text as follows:

Note

A Note highlights supplemental information.

About entering and typing

In this primer there is a distinction between "entering" a command and "typing" a command. This distinction is:

type	Type the command without pressing the **RETURN** key.
enter	Type the command then press the **RETURN** key.

Assistance from other sources

Additional resources are available to you. They are introduced in the sections that follow.

Associated documentation

For additional information on the topics presented in this primer, refer to these documents:

- *ConvexOS Tutorial Papers* (DSW-002), is a collection of documents that discuss the Berkeley 4.3 BSD operating system and ConvexOS, which is derived partially from 4.3 BSD.
- *ConvexOS Man Pages for Users* (DSW-331), documents Sections 1 and 7 man pages for the ConvexOS operating system.
- *ConvexOS Man Pages for Programmers* (DSW-332), documents Sections 2, 3, 4, and 5 man pages for the ConvexOS operating system.
- *ConvexOS Man Pages for System Managers* (DSW-333), documents Section 8 man pages for the ConvexOS operating system.
- `vi` *Quick Reference* (DSW-019), consolidates `vi` and `ex` commands into sections according to their functions.
- *ConvexOS Extensions User's Guide* (DSW-053), documents utilities unique to ConvexOS and utilities significantly modified by CONVEX, such as Large Files, Share, Checkpoint/Restart, POSIX Concepts, and more.

For supplemental information to the topics and concepts presented in this primer, a bibliography is included.

CONVEX Technical Assistance Center (TAC)

If you have question that are not answered in this book, contact the CONVEX Technical Assistance Center (TAC).

- Within the continental U.S., call 1(800)952-0379.
- From Canada, call 1(800)345-2384.
- Outside the continental U.S., contact the local CONVEX office.

Ordering other documents

To order the current edition of this or any other CONVEX document, send requests to:

CONVEX Computer Corporation
Customer Service
P.O. Box 833851
Richardson, TX 75083-3851 USA

Reporting problems

The TAC recommends using the `contact` utility to report a hardware, software, or documentation problem. This utility is an interactive program that helps TAC route each report to the CONVEX personnel most qualified to fix it and to track its resolution. Instructions for how to use this utility are in Appendix B of this primer.

Getting started and finding help

1

This chapter shows you how to enter commands to your CONVEX system and how to look up online reference materials for answers to your questions.

The topics explained in this chapter are:

- How to open a work session: logging in
- How to change your password
- How to close a work session: logging out
- How to enter commands and correct typing mistakes
- How to request online information

Accessing your CONVEX system

Each time you use a CONVEX system, your first action is to *log in*. This tells the system who you are so it can verify that you are an *authorized user*. Logging in has two parts: you enter first your login *username*, then your *password*.

Opening a work session: logging in

Before you learn how to log in, if you have any questions, ask the person who is in charge of your system. This person is the *system manager*, whose duties usually include: loading new software, establishing user accounts (the definition and allocation of resources available to a user), and keeping the system in good running order.

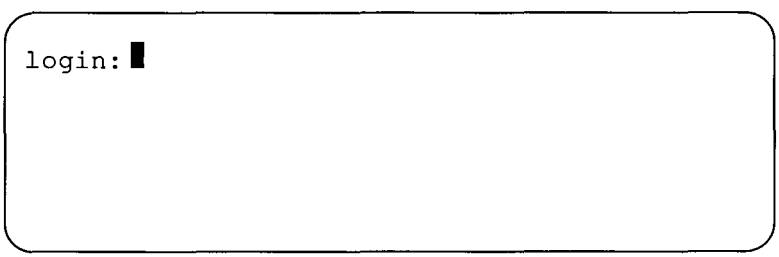
First step: entering your user name

Prior to entering your user name, your system manager has entered information that identifies you as an authorized user. When you know your user name and password, you are ready to log in.

When you log in, what you type is compared with what the system knows about you. If what you type differs from what the system has on file, you will be denied access. Typically, a user name is a person's first or last name or the name of a project.

If your terminal is already powered on, its screen should look something like Figure 1-1.

Figure 1-1
The login prompt



If your terminal does not look like Figure 1-1, ask your system manager what you need to do to get the login prompt. Today's systems support a variety of windowing packages and communications packages that alter how users access a system.

Note

In all commands and text entry, ConvexOS is case sensitive: it distinguishes uppercase and lowercase letters. The name newton is not the same as the name Newton or NEWton or newTON.

Whenever you enter a command, make sure that you are using lowercase and uppercase exactly as specified.

Beside `login:`, enter your user name. Figure 1-2 shows the user name **`marble`** directly beside the prompt.

Figure 1-2
Entering your user name

```
login: marble
Password:
```

Remember to press **RETURN** after typing your user name. Pressing **RETURN** indicates that you have finished typing and signals software to start interpreting what you typed. Throughout this primer, always press **RETURN** after text that you are told to *enter,* unless you are requested to just "type" something.

Note

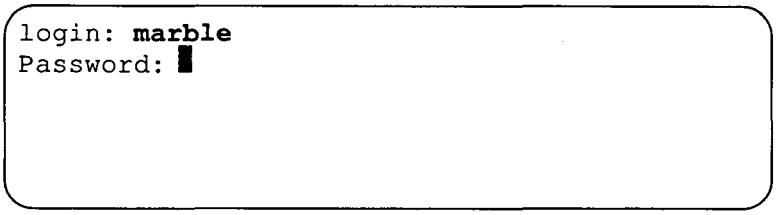
If you make a typing mistake as you are entering your user name or password, press RETURN until the login prompt displays again, so you can start over.

Second step: entering your password

After you enter your user name, you are asked for your password. A password ensures that you are who you say you are. On systems with several users, as is the case with CONVEX systems, passwords help prevent someone from impersonating you by using your user name.

After you enter your user name, your screen should look like Figure 1-3 as the system waits for you to enter your password.

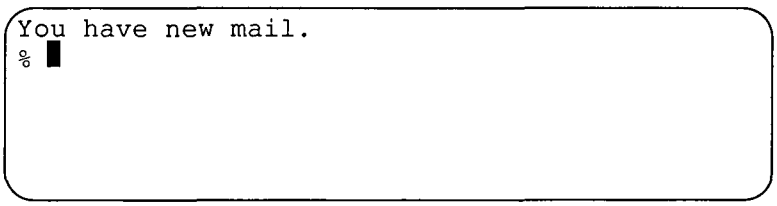
Figure 1-3
Entering your password



Enter your password, and don't forget to press **RETURN**. Remember, passwords must be confidential. ConvexOS honors confidentiality by not displaying the characters of your password as you type them.

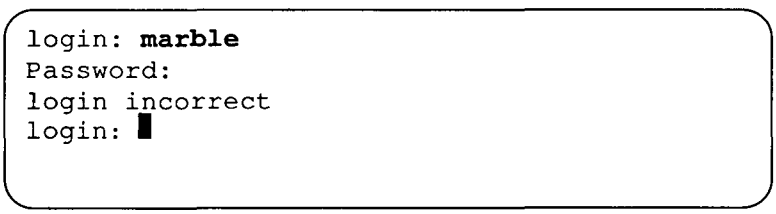
Seeing the shell prompt on your screen tells you that you are logged in. Some systems notify you when you have mail, as shown in Figure 1-4. Chapter 7 explains how to send and receive mail.

Figure 1-4
Mail notification at login



If you incorrectly enter your user name or password, a message displays as shown in Figure 1-5.

Figure 1-5
Incorrect login entered



You must reenter your user name and password. If you are unsuccessful after several tries, check that you have the correct user name and password.

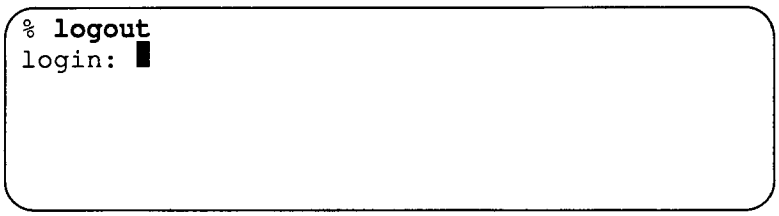
Closing a work session: logging out

When you are through using the system, terminate your work session by *logging out*. Logging out terminates your connection with the computer so that no one else can use your login.

To log out, as shown in Figure 1-6, enter

```
logout
```

Figure 1-6
Logging out

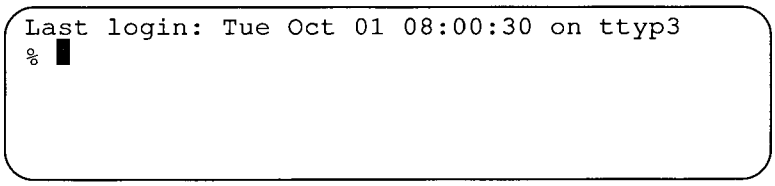


Pressing **CTRL-d** may also log you out. When the `login:` prompt appears on your screen, you are no longer communicating with your CONVEX system. Now, you can either turn your terminal off or leave it on ready for another user to log in.

Ready to use the system

You communicate with your CONVEX system though a command interpreter called the *shell*. Figure 1-7 shows a sample of what your screen may look like when you first log in.

Figure 1-7
The shell prompt and cursor



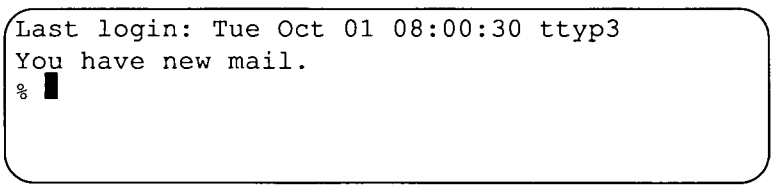
Directly beneath the notice regarding your last login (you may also have a mail notice) is the *shell prompt*, the % shown in Figure 1-7. Beside the shell prompt is the *cursor*— a solid rectangle— marking where you enter requests to your CONVEX system. This line is the *command line* where you type *commands*, your requests to ConvexOS. When you see the shell prompt and the cursor, you know the system is waiting for input.

Logging in successfully places you in your *home directory*. Chapter 5 explains your home directory and how it fits into the ConvexOS file system. With the shell prompt on your screen, move on to the next section.

Typing requests to your CONVEX system

In the previous section you learned about the command line where the shell prompt sits. On this line you enter commands that are ultimately interpreted into a sequence of instructions to be carried out by system software. In Figure 1-8, the cursor is on the command line, ready for you to enter a command.

Figure 1-8
Ready to enter commands



Every activity that you want to perform has a command that calls a program to perform it. Actually, commands are utility programs. After each command executes, the shell prompt reappears to let you know that the shell is ready for another command. Because this primer uses a specific shell (the C shell) for discussion, you will see the C shell prompt, `%`, in examples. Shell prompts are explained in Chapter 2.

Control keys

Before you start entering commands, you should know about the functions performed by some special control keys. Table 1-1 introduces these keys, showing that you press and hold down the **CTRL** key as you press the lowercase key stated beside it. Do not type the hyphen.

Table 1-1
Control keys and what they do

To do this	Press
Erase last character typed	**CTRL-h**
Erase last character typed	**BACKSPACE**
Erase entire line	**CTRL-u**
Suspend output to screen	**CTRL-s**
Resume output to screen	**CTRL-q**
Discontinue output to screen	**CTRL-o**
Abort an executing command	**CTRL-c**

You can assign (bind) these functions to keys other than the defaults listed in Table 1-1. If these keys produce results different from what is shown in this chapter, ask your system manager about their current assignments.

Various documents, including man pages (explained later in this chapter) show an uppercase key with **CTRL**, when the lowercase of that character is what you enter.

For practice, try the examples presented in Table 1-2.

Table 1-2
Erase examples

Type this	Then press	Result
datee	**CTRL-h**	date
date	**CTRL-u**	
dates	**BACKSPACE**	date
dated	**DELETE**	date

Entering commands

The best way to learn how to enter commands is to try them. The sections that follow give you three simple and useful commands to try:

date	Gives the date and time
ls	Lists the files in a directory
who	Lists users currently on the system

Try the special control keys defined in Table 1-1 as you enter these commands.

Displaying today's date and time

To display today's date and time, use the `date` command. Enter this command as shown in Figure 1-9. Do not forget to press **RETURN.**

Figure 1-9
Sample `date` output

```
% date
% Tue Jan 01 13:00:30 CST 1991
% 
```

Listing files in your current directory

The `ls` command lists the names of files in your current directory, which is your home directory when you log in. Sample output from this command is shown in Figure 1-10.

Figure 1-10
Sample `ls` output

```
% ls
business    editing     old
carrier     examples    plan2
documents   first       reports
% 
```

As you learn more about commands in the upcoming chapters, you will also learn more about the `ls` command.

Checking to see who is using the system

To get a list of users currently logged in, enter **who**. The output, similar to that shown in Figure 1-11, lists:

- User name
- Terminal (tty) line used (tty originally came from the brand name Teletype)
- Date and time of login

Figure 1-11
Users currently logged in

```
% who

belmont     ttyq1    Aug 5 01:22
cambridge   ttyp7    Aug 5 08:43
center      ttyp6    Aug 5 10:14
dedham      console  Aug 4 18:52
hingham     ttyp1    Aug 4 21:48
%
```

Changing your password

Having a password on your account helps safeguard you from unauthorized access to your files. Even if your system manager installed a password for your login account, good security guidelines suggest that you change it so no one but you can log into your home directory. Changing your password is simple and quick with the `passwd` utility.

A valid password

A valid password must contain at least six characters, and at least one character must be alphabetic. If you make your password longer than eight characters, only the first eight characters are used.

Avoid using your first name, last name, family name, anniversary or birth dates as your password. Choose a password that cannot be easily guessed. Once you assign it, remember it (do not write it down); otherwise, you will have to ask your system manager for help.

How to change or first assign a password

To prevent someone from walking up to your terminal and changing your password, the `passwd` utility first requests your current password as the *old password*. Next, it requests the new password, not just once but twice, in order to check that both words that you type match. To change your password, enter:

```
passwd
```

Figure 1-12 shows how the `passwd` utility prompts for information.

Figure 1-12
Changing your password

```
% passwd
Changing password for hingham on convex
Old password:
New password:
Retype new password:
Rebuilding passwd and pwrestrict databases...
% 
```

Neither the old nor the new passwords display when you type them. Entering an incorrect old password also stops the change. A `Sorry` message displays, and you must start over. Should your first and second password entries not match, this message displays and you must start over:

```
Mismatch - password unchanged.
```

Entering a password with fewer than six characters stops the change with a request for a longer password. Figure 1-13 shows the display after an attempt to change to a short password.

Figure 1-13
Refusing a short password

```
Please use a longer password.
New password:
Retype new password:
Rebuilding passwd and pwrestrict databases...
```

System prompts inform you as to what you need to do or enter next.

Requesting command and topic information

The two easiest ways to find command information are by using the following commands:

- `apropos` —Lists commands relating to a topic you specify.

  When you do not know the appropriate command to use for a task, refer to the next section, "What command do I use?"

- `man` —Displays the information page or man page for the command or topic you specify.

  If you need to know how to use a particular command, refer to the section "Where are command and topic explanations?"

What command do I use?

The command

`apropos` *topic*

matches the *topic* you supply against the description stored for each ConvexOS command. The output is a list of commands that relate to the topic you enter. For example, if you want to know all the commands from which you can get information on users, enter:

```
apropos users
```

Figure 1-14 shows a list of commands that fit this request.

Figure 1-14
Output from `apropos users`

```
binmail (1)      - send or receive mail among users
getusershell, setusershell, endusershell (3) - get legal user shells
last (1)         - indicate last logins or users and teletypes
lim (8)          - change share data base information for one or more users
pl (1)           - print share information for designated users
rnusers, rusers (3r)- return information about users on remote machines
rusers (1c)      - who's logged in on local machines (RPC version)
rusersd, prc.rusersd (8c)- network rusers server
rwall (1)        - write to all users over a network
users (10        - compact list of users who are on the system
```

When a precise topic is not known, matching can be done on a keyword, where *keyword* is a word, part of a word, or a phrase. Phrases must be enclosed in single quotes. The syntax is

`apropos` *topic keyword*

Where are command and topic explanations?

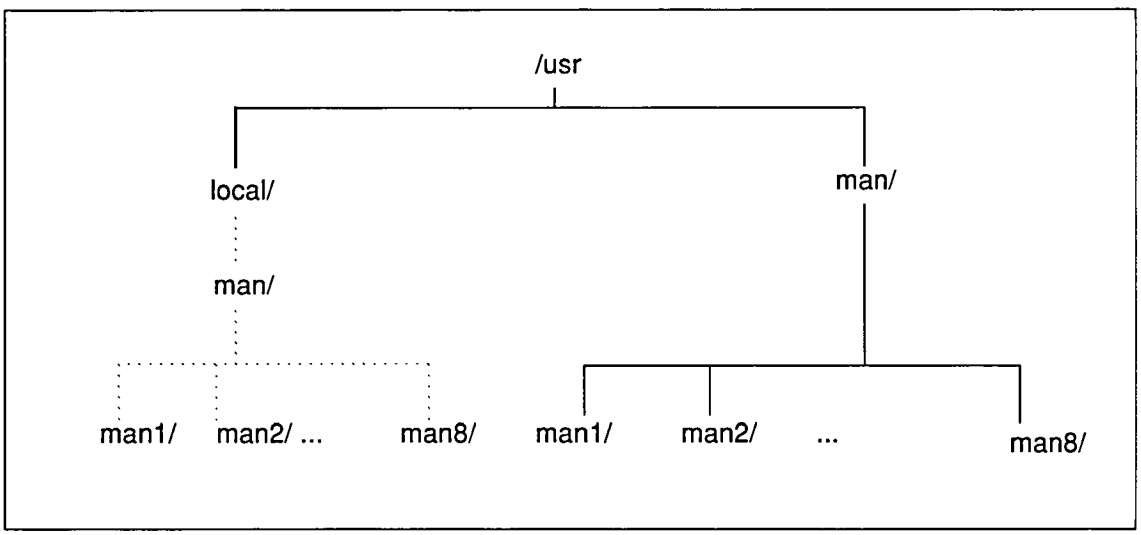
Information on topics such as commands, functions, and system calls can be requested from your keyboard and viewed on your screen. This type of information access is known as command *online information*. One online information program maintained by ConvexOS is `man`.

On the system, ConvexOS and local (site-specific) `man` pages branch from separate subdirectories under the /usr directory.

- ConvexOS man pages are in /usr/man.
- Local man pages are in /usr/local/man.

This arrangement prevents site-specific (local) man pages from being overwritten during an update of ConvexOS man pages. Figure 1-15 diagrams this directory layout.

Figure 1-15
Directory organization of man pages



Directories and files are introduced in Appendix A. Chapter 5 explains the ConvexOS file system in detail.

Finding what you need

On your system, man pages are organized by information type into eight subdirectories: man1 through man8. Each subdirectory is referred to as a man page section. Listed below, for each man page section, is the type of information given for each topic explained in that section:

man1	Commands of general utility and commands for communication with other systems
man2	System calls and error numbers
man3	Various library functions
man4	Special files, related driver functions, and networking support
man5	Format of various files
man6	Games; not supplied by CONVEX
man7	Miscellaneous commands, primarily for text processing and terminal environments
man8	System management and maintenance commands

Some topics are explained in more than one section, because each section explains the topic from a specific viewpoint. Within each subdirectory, each topic entry is stored in a separate file.

Reading a man page: written or online

Written references to man pages use this form:

tty(4)

This example of tty(4) refers to Section 4 that contains the explanation on the handling of tty input and output.

Online references to a man page are called by the `man` command. To read the manual page for the `man` command, enter

```
man man
```

The command syntax for requesting a man page is:

`man` [*section_number*] *topic*

where *section _number* is the number of the `man` section (subdirectory) and *topic* is the name of the command, system call, function call, or topic that you want explained. For example, to view the man page for the `ls` command, enter

```
man 1 ls
```

When a section number is omitted, the predefined search order for /usr/man directory is followed. Usually, Section 1 is the first section in the search order.

To get the information you need on a topic, you may need to request information from several sections as illustrated with tty:

man tty Section 1 explains the command `tty`.

man 4 tty Section 4 explains the handling of tty input and output.

Using the tty examples introduced above, full file names for these topics are:

man tty /usr/man/man1/tty.1

man 4 tty /usr/man/man4/tty.4

The extension on each file name states the section (subdirectory) in which the topic resides.

Reading a man page on your screen

When you view a man page, the `man` utility filters the text through a paging system called `more`, displaying one screenful of text at a time. Pressing **RETURN** displays the next line; pressing the **SPACEBAR** displays the next screenful.

Subsections organize text within a man page. The subsection headings are search arguments that direct the `man` utility to go directly to that point within the file and to start the text display right there. Table 1-3 describes the information explained in each subsection.

Table 1-3
Subsections within a man page

Subsection heading	Information
NAME	Name and one-line description of the command or topic.
SYNOPSIS	Syntax or format of the command or topic.
DESCRIPTION	Explanation of what the command or topic does and how to use it and any of its arguments.
FILES	Files used by the command or subroutine.
SEE ALSO	Associated topics.
DIAGNOSTICS	Discussion of system diagnostic messages.
BUGS	Explanation of known bugs or deficiencies.

Not all subsections are relevant to all entries. You can inquire as to which ones exist for an entry. To request a list of the subsections documented for an entry, enter

man -i *topic*

Once you know the subsections, you can request a search directly to a subsection within a man page. To request a subsection, enter

man *topic/subsection*

For **man passwd/files**, man searches the passwd file until it finds the subsection FILES. When located, the display begins at the subsection FILES.

The man page for `passwd` is long. To fit all the subsection headings into a screen example, Figure 1-16 is a condensed version of the **man passwd** output.

Figure 1-16
Condensed output from `man passwd`

```
PASSWD(1)                ConvexOS Man Pages          PASSWD(1)

NAME
       passwd - change login password

SYNOPSIS
       passwd [ -f filename ] [ username ]

DESCRIPTION
       This command changes (or installs) a password associated with the
       username (your own name by default). If the -f option is given,
       filename is treated as the password file.

       The program prompts for the old password and then for the new one.
       You must supply both, and the new password must be typed twice to
       forestall mistakes.

       Only the owner of the name or the superuser may change a password;
       the owner must prove he knows the old password.

FILES
       /etc/passwd /etc/passwd.dir /etc/passwd.pag /etc/pwrestrict
       /etc/pwrestrict.dir /etc/pwrestrict.pag

SEE ALSO
       login(1), passwd(5), pwrestrict(5), crypt(3), chsh(1), chfn(1),
       mkpasswd(8)

RESTRICTIONS
       Passwd will accept passwords longer than eight characters, but only
       the first eight characters will be used.

BUGS
       Passwd will change a local password, but not a password in the network
       NIS.

       Since passwd rebuilds the passwd and pwrestrict database files, a
       large password file may produce a long wait.

CONVEX Release
```

Searching when the command is unknown

The `-k` option of the `man` command has the same capability as the `apropos` command described earlier in this chapter. Use whichever one you like.

How to find online explanations

The CONVEX Info System contains a wide-range of information. You can select explanations on specific tasks, descriptions of how to use a command, or an explanation about a specific topic. You can learn how to talk with other users and machines, use online help, enter commands, work with files, develop programs, perform arithmetic calculations, and request an explanation on a command.

How to use the Info system

The first time you request Info, you get a one-time introduction that explains the main menu, its submenu system, command descriptions, and help screens. From any of the menus, you can enter a topic name to access information about that topic or to get command descriptions.

To request Info, enter

info

After you read the Info introduction, you can request the main menu as shown below in Figure 1-17.

Figure 1-17
Main menu for Info

```
                CONVEX INFO SYSTEM MAIN MENU
                ============================

          1. Contact other users or machines
          2. Use UNIX online help
          3. Execute Commands
          4. Edit, find, print, modify, analyze,
             and archive files
          5. Develop programs
          6. Check user, job, or system status
          7. Modify system and file accessibility
          8. Perform arithmetic calculations



Enter<1..8>,<q>uit,<?> help,<t>opic/command list,a command name,a topic

Please type your selection and press <RETURN>:
```

All of these menu options are task-directed. Choosing an option brings you to its submenu; some topics have more than two or three levels. The further down in the submenus you travel, the more detailed the task explanation; the lowest level contains command descriptions.

At the bottom of the main menu, all submenus, and information screens is an instruction line stating how to move from screen to screen, how to access information, and how to exit. Help screens are also available for the main menu, submenus, and command descriptions.

Requesting command and topic information

Information on a command can be found by entering a related topic name. If you cannot think of an appropriate topic name, or if the topic name you choose is not found in the Info system, read the topic/command list. To display this list, refer to the main menu, as shown in Figure 1-17, where it tells you to enter **t**.

If you choose the number 2 entry from the main menu, you will get the Online Help submenu as shown in Figure 1-18.

Figure 1-18
Help menu

```
            USE UNIX ONLINE HELP - Submenu of Main Menu

            1. Find information on a command or tasks
            2. Use the VMS/DCL user interface (COVUE command)
            3. Send a problem report to CONVEX
            4. Display a brief description of a command
            5. Access the limited UNIX online tutorials
            6. Display reference information about commands
            7. List UNIX commands related to a keyword
            8. List misspelled words in a file
            9. Find the correct spelling for a word
           10. Print a calendar for the specified year or month
           11. Remind yourself about dates in your calendar file


Enter<1..11>,<b>ack,<ESC>main menu,<q>uit,<?>help,a command name,a topic

Please type your selection and press <RETURN>:
```

Table 1-4 presents the keys that invoke functions at various levels within Info.

Table 1-4
Info functions and keys

Key	Action
b	Displays previous screen
n	Displays next screen
SPACEBAR	Displays next screen
ESC	Displays main menu
q	Exits Info System
?	Displays information on using command description screens
m	Displays the manual page
x	Displays prompt for entering ConvexOS commands
r	Invokes the problem-reporting utility: `contact`

Chapter highlights

Check off the topics that you feel you have mastered. Return to those that require more of your time.

- ❒ **Logging in**
  - -User name
  - -Password
  - -Login prompt
  - -Mail notification
- ❒ **Logging out**
- ❒ **Ready to use the system**
  - -Typing commands
  - -Editing what you type
- ❒ **Changing your password**
  - -A valid password
  - -An invalid password
- ❒ **Accessing online command information**
  - -Matching a task to the right command
  - -Learning about a command
  - -Selecting an explanation on a task or topic

Your work environment

2

You work on your CONVEX system from within your *shell environment*. This environment is a link between you and your CONVEX system. Stated briefly, the *shell* accepts commands that you enter and oversees their execution.

The topics explained in this chapter are:

- Definition of a shell and what it does
- Characteristics of your shell environment
- Shells that are available to you
- How your shell is configured
- How you can alter your shell's configuration

Introducing the shell

The shell is a *command interpreter*; it evaluates commands that you enter and invokes programs needed to execute your requests. ConvexOS supports more than one shell. Changing the shell that you use from one to another is easily done with a command, because shells are part of the user-level portion of the operating system, not part of the core of the operating system—the kernel. (Appendix A briefly introduces the user-level and kernel-level of the operating system.)

A shell is a programming language. A series of shell commands can be saved in a file, called a *shell script*, to perform specialized functions. Typing complicated command sequences in shell scripts saves time and programming effort. (Shell programming is outside the scope of this primer.)

In addition to its programming capability, some of the shell's features enable you to direct output from one program to be input to another. The shell also enables you to run more than one program at a time and to switch back and forth between these programs. Chapter 3 discusses these capabilities as part of showing you how to work with your shell.

The shell is an interesting and powerful program. Its features can greatly ease your work.

ConvexOS supported shells

The shells supported on ConvexOS are described in Table 2-1.

Table 2-1
ConvexOS shells

Shell	Start-up file	Description
Bourne (`sh`)	.profile	The first UNIX shell. C programmers tend to find `sh` shell constructs less convenient than those of the C shell.
C (`csh`)	.cshrc	The standard for BSD systems. It is incompatible with the Bourne or Korn shells, but has some of the extended features similar to `ksh`. Its programming constructs look much like the C programming language.
Korn (`ksh`)	.kshrc	The standard for System V UNIX systems. It is an extension of the Bourne shell that incorporates features to make entering, editing, and reissuing commands easier. An optional product that is part of the CONVEX Toolbox software.
COVUEshell	LOGIN.COM	Sets a CONVEX-to-VAX user environment. Supports DCL-like commands, lexical functions, and qualifiers on CONVEX systems. An optional CONVEX product.

If you are not sure which shell you are using, enter:

```
echo $shell
```

Table 2-2 associates the output from `echo $shell` with its shell type.

Table 2-2
`$shell` output and shell type

`$shell` response	**Shell type**
`/bin/sh`	Bourne shell
`/bin/csh`	C shell
`/bin/ksh`	Korn shell

If you are using COVUEshell, **`echo $shell`** is not appropriate. The `$` is the COVUEshell prompt. Interactive ConvexOS shells can be requested from the `$`.

All shells have variables that are used during shell execution and start-up files that assign the values to these variables. *Environment variables* and *local variables* are two important terms; they are not interchangeable. Although they are related in some ways, you will learn how they differ in the next section: "C shell variables and files."

Your login shell

As part of logging in, ConvexOS invokes your (default) *login shell*, the shell type defined by the environment variable SHELL. Environment and local variables are explained in this chapter. How your login shell handles your commands and jobs is based on the values set to these variables.

Working within your shell environment

Your shell environment is your work environment. Just as you work within an office, you work within your login shell when you use ConvexOS. When your login shell is ready to accept commands, its prompt displays on your screen. It first displays after the successful completion of these login events:

- You respond at the login prompt with your user name.
- You respond to the password prompt with your password.
- Software reads the contents of your start-up files and sets your shell environment.
- The shell process executes, and the shell prompt displays on your screen, waiting for you to enter commands.

Changing your login shell

Changing from one shell to another is done with the `chsh` command. Before you attempt to change your login (default) shell, talk with your system manager; there may be considerations that could affect your work environment.

Figure 2-1 is a log of the commands issued while changing the current shell from `csh` to `sh`.

Figure 2-1
Changing your login shell

```
% echo $shell
/bin/csh
% chsh marble sh
Rebuilding passwd database...
% echo $shell
/bin/csh
% logout
login: marble
password:
% echo $shell
/bin/sh
%
```

C shell variables and files

This section focuses on the most widely-used shell, `csh`. Its popularity can be attributed to its many features that help all levels of users become more productive and to its shell programming language. The programming language's likeness to the C programming language is how this shell got its name.

To summarize some of its basic capabilities, `csh` lets you:

- Create shorthand notations for a command or series of commands
- Execute several jobs simultaneously, with or without user intervention, known as foreground and background execution
- Stop a job and restart it
- Execute previously-issued commands
- Tailor your shell work environment to meet your needs
- Write command-level programs

Some of these capabilities are beyond the scope of this primer. Consult the *ConvexOS Tutorial Papers* and the ConvexOS documentation set for further information.

What are variables?

As `csh` handles your commands and jobs (executing programs), it maintains a record of:

- Your home directory
- Terminal type in use
- Shell prompt definition
- Directory search order to follow for locating commands

To do all of this, the shell consults two types of variables: *environment variables* and *local variables*. Environment variables are inherited by all programs executed from the shell, as well as by new shells. Local variables, however, are known only to the shell in which they are defined. Some local variables are initally predefined (initialized from system files), but user redefinable, while other local variables are always user definable.

Environment variables

Each user's work environment maintains data about the environment, such as home directory and terminal type, in a special set of variables called environment variables.

Figure 2-2 shows the most commonly used environment variables. Names of environment variables must be in uppercase.

Figure 2-2
Values of environment variables

```
% printenv
HOME=/mnt/belmont
SHELL=/bin/csh
TERM=vt100
USER=belmont
PATH=.:/mnt/belmont/bin:/usr/texas:/usr/etc
HOST=customer
EDITOR=vi
PRINTER=swift
%
```


ConvexOS passes or exports these variables to programs and new shells. Such an example is the `vi` editor that checks TERM variable for the type of terminal you are using, so it can correctly interpret the keystrokes you enter. Other commands and programs reference specific variables when they execute. These variables have default settings that you may change.

To view the values set to your environment variables, enter

`printenv`

To view the value of one specific environment variable, enter

`printenv` *ENV_VARIABLE_NAME*

where *ENV_VARIABLE_NAME* is the name of the environment variable.

To define or change the value of an environment variable, enter

`setenv` *ENV_VARIABLE_NAME value*

where *ENV_VARIABLE_NAME* is the name of the environment variable, type a space, followed by the *value* you are assigning to it.

For example, setting your default editor to `vi`, the command line would be similar to

`setenv EDITOR /usr/ucb/vi`

To disable an environment variable for the current work session, enter

unsetenv *ENV_VARIABLE_NAME*

where *ENV_VARIABLE_NAME* is the name of the environment variable.

Table 2-3 describes some common environment variables.

Table 2-3
Standard environment variables

Variable	Purpose
HOME	Path name of your home directory (explained in Chapter 5).
SHELL	The absolute path name of your login shell (initialized from the /etc/password file by the login program).
TERM	Your terminal type so the shell knows how to interpret certain keys such as controls keys and the **BACKSPACE** key (obtained from the /etc/ttys file by the login program).
USER	Your login name (initialized from the /etc/password file by the login program).
PATH	The search route followed by the shell when it searches for a command or program. (A colon marks the end of an individual directory path.)
HOST	The system where you are currently logged in.
EDITOR	Your default editor. When text editing commands execute, they look at this variable to determine which editor to use.
PRINTER	The system printer that receives your data from the commands `lpr`, `lpq`, and `lprm`, unless you specify another printer name in your print request.

Local variables

csh uses other variables called *local variables* to customize your work environment. Some are predefined by csh—path, term, shell, user—and are also redefinable by users; while others are user-defined variables. To view the values assigned to local variables, enter the set command, as shown in Figure 2-3.

Figure 2-3
Common local variables

```
% set
cdpath /mnt/belmont
history
savehistory 15
home /mnt/belmont
ignoreeof
mail (/usr/spool/mail/belmont)
noclobber
notify
path (. /mnt/belmont /bin /usr/texas)
shell /bin/csh
term vt100
user belmont
%
```


The first name on a line is the variable; the character string to the right is its value. Some variables (for example, ignoreeof, noclobber, and notify) have no value. These variables are set when listed in a start-up file, and unset when not listed or disabled by the unset command on the command line.

Value assignments for local variables require an equal sign (=) and no spaces between the variable name and its value. To set a local variable or change its current setting for the current work session, enter

set *variable_name=value*

For example, to change your shell, you would enter

```
set shell=/bin/ksh
```

To unset a variable for the current work session, on the command line enter

unset *variable_name*

Unlike environment variables, local variables are not inherited by subsequent shells. Each new shell must have its own local variables.

Some local variables have default settings, while others get their values automatically from ConvexOS information files or environment variables. Table 2-4 describes variables that you might find in your start-up files. Start-up files are explained in the next section.

Table 2-4
Standard local variables

Variable	Purpose
cdpath	Changes the search map used by the shell to locate the directory where you want to move. It means change path.
home	Defines the path name of your home directory taken from the HOME environment variable.
ignoreeof	Instructs the shell to ignore a **CTRL-d** from the terminal.
mail	Specifies the files where the shell looks for your mail.
notify	Requests the shell to send notice when a job completes.
noclobber	Prevents accidental overwrites to existing files. Also, prevents appending output to a nonexistent file.
shell	Contains the path name of the shell.
term	Establishes the type of terminal that you have. Copied from the TERM environment variable.
user	Identifies your user name.

Looking into csh start-up files

A shell start-up file is an ordinary text file containing a list of one or more commands that you would ordinarily enter one at a time. Instead, you place them in a start-up file in the order that they are to execute. The naming convention of this type of file tells the shell to sequentially execute the contents of the file. Commands that customize or change your environment should reside in one or both of the two shell start-up files. Because these files are text files, you can add, change, or delete information from them using a text editor.

When you log in, ConvexOS creates your login shell for you. This shell looks in your home directory for csh's start-up file—.cshrc—and follows the instructions it finds in this file. Next, the shell looks for a file called .login and follows the instructions in it. The difference between these two start-up files is that the .cshrc file executes each time a new csh shell starts, while the .login file executes only when you log in.

Listing the contents of your .cshrc file

To look at the contents of your .cshrc file, as shown in Figure 2-4, enter

```
cat .cshrc
```

Figure 2-4
Sample .cshrc file

```
% cat .cshrc
set autologout=30
set history=100
set savehist=500
set shell=/bin/csh
#set noclobber
set notify
%
```

For the duration of the current terminal session, at the command line you can change or unset a variable's current setting. To permanently disable or remove a setting, use an editor and open the start-up file and delete the setting. Inserting a # (pound sign) as the first character on a line, preceding a variable setting, tells the shell to ignore from the # to the end of that line. This is known as *commenting out* a line, and the ignored text is now a *comment*. Removing the # makes it available to be read by the shell.

Listing the contents of your .login file

Commands that need to be executed at the beginning of a work session and variables that must be available throughout the work session should be placed in your .login file. Use this file to declare global variables. (These variables have values that remain constant regardless of where or in what program they are used.) Entries common to .login files are shown in Figure 2-5 and explained following the figure.

Figure 2-5
Sample .login file

```
% cat .login
date
uptime
set ignoreeof
setenv TERM vt100
setenv EDITOR vi
setenv PRINTER river
setenv SHELL /bin/csh
biff y
mesg n
%
```

`date`	Prints the day, date, time, and year when you log in.
`uptime`	Prints the current time, the length of time the system has been up, and the average number of jobs in the run queue over the last 1, 5, and 15 minutes when you log in.
`ignoreeof`	Instructs the shell to ignore the end-of-file input from the terminal, preventing the shell from being accidentally killed by **CTRL-d**. You must enter the `logout` command.
`setenv TERM`	Defines the terminal type as described earlier.
`setenv EDITOR`	Defines your default editor.
`setenv PRINTER`	Defines your default printer.

`setenv SHELL`	Defines your default shell.
`biff y`	Requests notification of mail when it arrives.
`mesg n`	Requests that the system not notify you when you receive messages.

In Figure 2-6 you see a # (pound) character in front of the `set noclobber` variable. The # is telling the shell to ignore (not to interpret) what follows it up to the end of that line. This ignored line is now referred to as a *comment*. Removing the # makes the line available to be read by the shell. This approach is how you can enable and disable settings quickly.

Figure 2-6
Disabling a variable in .login

```
% cat .login
date
uptime
set ignoreeof
#set noclobber
crt erase ^h kill ^u
setenv TERM vt100
biff y
mesg n
% 
```

Changes made to either .cshrc or .login do not take effect until the file executes again. To run these files without having to log out and log in again, use the `source` command. `source` executes the commands found in the .cshrc or .login files just as if you were logging in; enter

`source` *filename*

Setting up a .logout file

When you log out, the shell looks for the .logout file in your home directory. If one exists, the shell executes its contents. You create a .logout file with a text editor. Figure 2-7 shows how a .logout can serve you.

Figure 2-7
Sample .logout file

```
% cat .logout
clear
echo -n "Logging out at: "
date
% 
```

`clear` Wipes the screen clean

`echo` Writes `Logging out at:` on your screen

`date` Writes the date after the echoed text

Changes made to .logout are ineffective until the file executes again. To test the commands that you write to this file, you can run it immediately by entering

```
source .logout
```

Figure 2-8 shows what displays on a screen as a result of the settings shown in Figure 2-7.

Figure 2-8
Sample output from .logout

```
Logging out at: Fri Jan 11 10:05:00 CST 1991
login:
```

Automatic logout is controlled by the variable autologout. The value assigned to autologout defines the number of minutes of inactivity allowed before you are automatically logged out. This value is also disabled by the `unset autologout` command. The default of the login shell is 60 minutes.

Chapter highlights

Check off the topics that you feel you have mastered. Return to those that require more of your time.

- ❒ **Facts about shells**
  - -ConvexOS supported shells
  - -Your login shell
  - -The shell prompt
- ❒ **Shell variables**
  - -Environment variables: viewing, setting, and unsetting
  - -Local variables: viewing, setting, and unsetting
- ❒ **Start-up files**
  - -Your .cshrc file
  - -Your .login file
- ❒ **Logout file**
  - -Why use it
  - -Automatic logout

Working with the C shell: commands, processes, and jobs

3

The C shell (`csh`) is both a command interpreter and a programming language. As a command interpreter, the shell executes commands that you enter. It also supports features that you will find useful as you gain experience with ConvexOS.

For users who are familiar with the `vi` editor or the `emacs` editor, you may want to read the csh(1) man page. `csh` supports the ability to request some C shell functions using either `vi`-style or `emacs`-style key sequences (bindings). This chapter explains selected C shell features using the shell's default key bindings.

The topics explained in this chapter are:

- Elements of a command
- Using character patterns to generate file names
- Asking the shell to complete file and command names or to list names of possible matches
- Redirecting command input and output
- Transferring data between commands
- Entering multiple commands on a command line
- Controlling your executing programs
- Controlling your jobs

Entering commands

A successful login is followed by the display of the shell prompt, telling you that your login shell is ready to accept commands. The line on which the shell prompt sits is called the *command line*. On this line, you enter commands and use the features supported by the shell.

`csh` does require that you follow rules when typing in a command line. These rules are explained before you learn how to use some of the useful features of `csh`.

Arranging elements of a command

Command line *syntax* dictates the spelling, ordering, and separation of the command elements on a command line. These elements are:

- *command_name*

  Each command name requests a specific operation.

  An example is: **`ls`**

- [*option...*]

  Optional characters that modify the way in which a command works. Preceded with a flag notation (a hyphen), they are single characters that may be strung together.

  An example is: **`ls -lar`** or **`ls -l -a -r`**

- [*argument...*]

  Technically, anything that follows the command name. But in actual use, an argument is the file name or directory name that a command is to act upon. Arguments can be optional.

  An example is: **`ls /mnt/belmont`**

In general

Place a space after the command name. Do not insert spaces within an element. One space must separate one element from another.

Commands are case sensitive. The shell distinguishes uppercase and lowercase letters. The name newton differs from Newton or NEWton or newTON.

Commands vary as to the elements they use. The man page for a command describes how to use the command and its elements.

Simple syntax

Basic command syntax has this form:

command_name [*option...*] [*argument...*]

An example would be to the list all entries, including your start-up files, either in your current working directory or in a directory that you name; you would enter

ls -a [*dirname*]

Many commands have several options that direct their operation. `ls` has an option that lists all entries (`-a` includes files that begin with a dot) with their creation time (`-c`), and another (`-F`) that marks directories with a " / " and executable files with an " * ". The command line that requests this output is

ls -acF [*dirname*]

Complex syntax

Commands taking multiple options and arguments, follow this syntax:

command_name [*option1*] [*argument1*] [*option2*] [*argument2*]

An example is:

tpmount -m label -s tape1 vol1,vol2,vol3

where **tpmount** is the command, and labeled below are the options and arguments.

[*option1*]	
-m	Specifies the access mode.
[*argument1*]	
label	Defines the mode as label.
[*option2*]	
-s	Specifies a symbolic link of tape1.
[*argument2*]	
tape1	Names the symbolic link as tape1;
vol1, vol2, vol3	are the volume serial numbers.

Requesting command syntax information

ConvexOS has many commands. Some you will use all the time, some occasionally, and others not at all. Whenever you need information, enter

man *command_name*

You can also request command syntax information as you are typing a command. The HPATH variable has to be set for this feature to work:

```
setenv HPATH /usr/lib/csh-help
```

As you are typing a command, press **ESC-h**. Displayed for that command is a brief definition with a syntax statement.

`csh` also corrects spelling mistakes in command, file, and user names. When you finish typing a command line and notice a mistake, press **ESC-s**; `csh` corrects the mistake, rewrites the line, leaving the cursor at the end of the line.

Practicing

You need to practice entering commands. The commands introduced in this section not only give you useful information, but an opportunity to practice using options, arguments, and **ESC-h** and **ESC-s**. Try the commands described in the following sections at your terminal.

Listing file information

To view information about files within a directory, use the `ls` (list) command. For each *dirname*, `ls` lists the contents of that directory; for each *filename*, `ls` repeats its name and gives any information requested by an accompanying option. Enter,

ls [*option*] [*argument*]

Figure 3-1 shows `ls` without an option or argument. Omitting an argument defaults to the current directory.

Figure 3-1
Files within current directory

```
% ls
10          TUTORIAL        business
667         bear            conews
Calendar    before          editing
News        box             examples
% 
```

Before you continue, read the ls(1) man page by entering

```
man ls
```

Read how the `-l` option provides a long listing on the files contained within a requested directory and how the `-F` option marks files that are directories with a trailing slash (/).

Figure 3-2 is a long listing of Figure 3-1. It also identifies the files that are directories. (`ls -lF` is the same as entering `lf`.)

Figure 3-2
Directory long listing

```
% ls -lF
total 41
-rw-r--r--        1 hingham      29     Jan  21   13:16    10
-rw-r--r--        1 hingham      28     Jan  18   12:30   667
drwx------        2 hingham      51     Nov  28   16:47   Calendar/
drwxr-xr-x        2 hingham      24     Dec  13   13:46   News/
-rw-r--r--        1 hingham      3791   Jan   2   07:35   TUTORIAL
drwxr-xr-x        2 hingham      24     Dec   2   17:59   bear/
-rw-r--r--        1 hingham      9837   Nov  29   13:47   before
-rw-r--r--        1 hingham      2243   Jan  17   13:18   box
drwxr-xr-x        2 hingham      512    Dec  17   13:04   business/
-rw-r--r--        1 hingham      3155   Jan  17   14:39   conews
-rw-r--r--        1 hingham      1002   Dec  17   12:14   editing
drwxr-xr-x        2 hingham      512    Jan   8   14:20   examples/
%
```


Displaying file contents

`cat` *filename* reads the contents of a file and displays it on the standard output, which, by default, is your screen. Figure 3-3 shows how the `-n` option of the `cat` (concatenate) command sequentially numbers each output line. In this figure, the argument is the file `phonelist`.

Figure 3-3
Looking inside a file

```
% cat -n phonelist
 1      bank          887-4140
 2      carpenter     942-3948
 3      cobbler       887-3367
 4      doctor        887-2343
 5      tailor        887-5928
%
```

Generating file names from character patterns

You may not always know the complete or correct name of a file or directory. Special characters, called *metacharacters* or *wild cards*, when typed in a file name, instruct the shell to replace that location with one or more characters. The shell uses metacharacters to generate file names. During its search, the shell compares the wildcard-generated file names against the directory-resident file names for matches.

Interpreting *metacharacters* in arguments occurs during these shell functions:

- Locating files and directories
- Directing input to commands and output from commands
- Executing a series of commands

Each metacharacter defines a specific character pattern. Those most frequently used are explained.

Question mark

The ? matches any one character except a leading period. For example:

```
ls key?
```

matches keys, key1, keyx.

Asterisk

The * matches any character or any grouping of zero or more characters, except those leading with a period. For example:

```
ls chap*
```

matches chap2.doc, chapter1-2, chap, and chaps.

Enclosed square brackets

The [ ] defines within it:

- Characters to be matched. For example:

  ```
  ls key[2468]
  ```

  matches key2, key2, key4, key6, key8. Not matched would be key3, key5 and so on.

- Range of characters or digits defined with the dash. For example:

  ```
  ls key[1-9]
  ```

  matches key1, key2, key3, key4, and so on up to key9.

Completing names

Completing names is another feature offered by csh. A name to be completed can be a:

- File name
- Directory name
- Command name

You only need to enter a unique abbreviation. Pressing **TAB** at the end of any number of characters causes csh to search, according to your PATH variable definition, for a unique match that completes the name. Frequently names are partially completed because the characters you entered match several longer names. Character matching stops at the point of ambiguity. Your next step would be to ask for a list of possibilities, meaning names that build from these base characters. Both of these features are explained next.

Completing directory and file names

Type the characters that csh is to match uniquely. csh halts searching when two or more names match on the same character. At this point a name becomes ambiguous; a beep sounds. A beep also sounds when there is no match. (To silence the beep, enter **set nobeep** in your .cshrc.)

This command line asks for a file name completion,

vi /mnt/beTAB

which completes up to the point shown in Figure 3-4.

Figure 3-4
Completing a file name

```
% vi /mnt/belmont
```

This file could be any type of file, including a directory.

Expanding uniqueness

The `set fignore` local variable enables you to define the suffixes to be ignored during the search for matches. This variable accepts a list of values, just like the path variable. Hence, even a single value must be enclosed in parentheses.

For example, in Figure 3-3, following the list of files in the current directory, `set fignore` is entered. Next, a completion request is entered, and the match displays followed by the cursor.

Figure 3-5
Setting suffixes to ignore

```
% ls /mnt/belmont/graph
chap.doc1          chaptera
chap.o             prep_files
chap.c             shell_notes
% set fignore (.o .c .doc1)
% vi /mnt/belmont/graph/chap TAB
% vi chaptera
```

TAB shows chaptera to be unique.

Expanding completeness

The `set recexact` variable enables **TAB** to expand to the point where ambiguity ends. Matching ends either on:

- An exact match (the shortest character string)
- Furthest point of ambiguity (where a decision must be made)

In Figure 3-3, the first **TAB** matched out to /dep. The second **TAB** matched and displayed the first unique match— dept.

Figure 3-6
Expanding **TAB** key matches

```
% set recexact
% ls
dept          depts         depth
% cd /mnt/belmont/de TAB
% cd /mnt/belmont/dep TAB
% cd /mnt/belmont/dept
```

Completing a command name

Asking `csh` to complete the path saves time. For example, entering

aprTAB

completes to **apropos**.

Listing name possibilities

At any time while typing a file name or a command name, including when a **TAB** reaches the "ambiguous" point, you can request a list of possible matches.

To demonstrate how this feature works, look at the directory contents shown in Figure 3-7.

Figure 3-7
Names used in examples

```
% ls /mnt
action                  needham
arlington               newton
belmont                 newtonville
boston                  newweston
cambridge               maynard
%
```

Entering

vi neCTRL-d

displays the matches of newton, newtonville, and newweston.

When the matches are special file types, the type is identified by one of these symbols:

Directories	/
Executable files	*
Symbolic links	@ (explained in Chapter 5)

Listing commands

Listing command names is useful. `csh` searches according to the PATH variable and lists commands that match the character pattern that you entered. For example,

lp CTRL-d

displays the matches `lpmv`,`lpq`, `lpr`, `lprm`, `lprman`, and `lprx`.

Listing login names

To list all the login names on your system, enter

cd ~CTRL-d

To make a search more specific, for example, a request for a list of logins that begin with the letters "wi," is

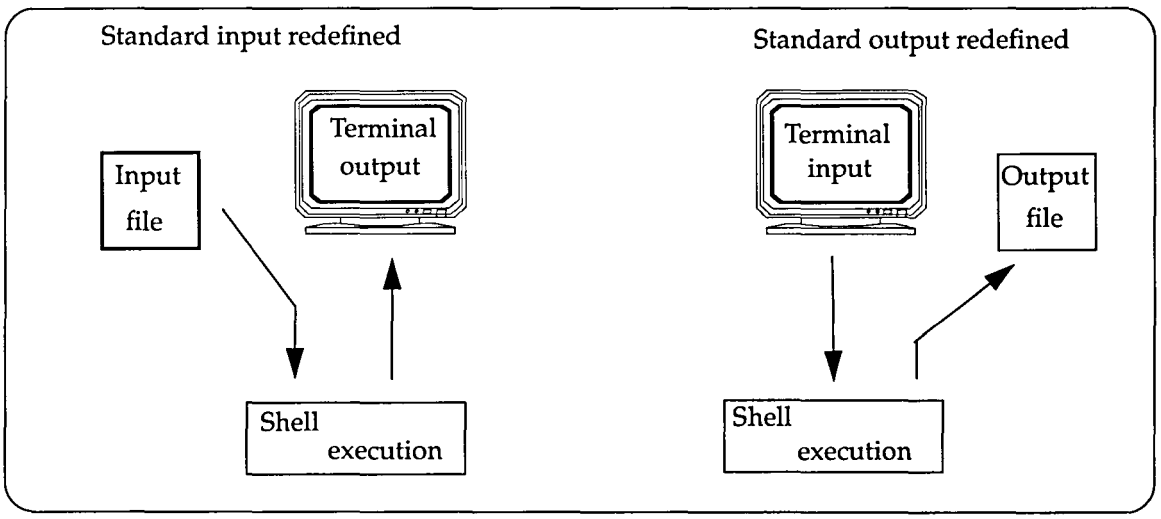
cd ~wiCTRL-d

Redirecting command output and input

Redirection describes methods for causing the shell to change where a command gets its *standard input* from or where it sends its *standard output* to. Standard input is the predefined source from which a program receives its input. Standard output is the predefined recipient of program output. By default, the shell associates a command's standard input and standard output with your terminal's screen.

You can cause the shell to redirect either or both standard input and standard output of any command (program). The sections that follow explain how to use the symbols that redirect input and output. Figure 3-8 illustrates redirection.

Figure 3-8
Redefining standard input and standard output



Redirecting standard output

The *redirect output symbol* is the greater than sign (>). It instructs the shell to send a command's output to the file specified instead of to your terminal, the default. The syntax is

command_name [*argument*] > *filename*

where *command_name* is an executable program, application, or ConvexOS utility; *argument* is optional; and *filename* is the name of the ordinary file to which the shell directs the output.

Note

Use caution when redirecting output. If the destination file exists, it is overwritten.

To prevent files from being overwritten by redirected output, enter `set noclobber` in the .cshrc file in your home directory. This setting prevents:

- Redirecting output to an existing file
- Appending a file to a nonexistent file

Once `noclobber` is set, you cannot redirect output to an existing file until you either unset this command or override it by entering the **>!** command.

Examples of output redirections are:

To send the contents of a file to another file, enter

cat *filename1* **>** *filename2*

To direct a list of current users to a specific file, enter

who > *filename*

Redirecting standard input

Just as you can redirect a command's standard output, you can redirect its standard input. The *redirect input symbol* is the less than sign (<). It instructs the shell to receive a command's input from the specified file instead of your terminal. The syntax for input redirection is

command_name [*argument*] < *filename*

where *command_name* is an executable program, application, or ConvexOS utility that produces output; *argument* is optional; and *filename* is the name of the ordinary file from which the shell receives input.

An example of input redirection follows.

For **sort** to receive its input from **who** and to direct the output of **sort** to some *filename*, enter

sort < who > *filename*

Appending output

Sending data to a file presents the chance that you may overwrite existing information. With the append output symbol, two greater than signs (>>), you instruct the shell to add new information to the end of an existing file, leaving the original contents intact.

Figure 3-9 displays command lines that request:

- A listing of users on system sys2 be written to the file current.sys2
- The contents of the file users.sys2 be appended to the file current.sys1

Figure 3-9
Appending redirected output

```
% who > current.sys2
% cat current.sys2 >> current.sys1
% 
```

>> cannot be used on a file that does not exist. To create an empty file, enter

touch *filename*

To learn about the `touch` command, read its man page description, by entering

```
man touch
```

Passing data between commands

Besides redirecting input and output to files, you can transfer data between commands. This ability is called *piping*. This is a *multiprocess* activity where two or more commands (processes) execute at the same time, eliminating the need for sequential processing of single commands.

The vertical bar (|) is the *pipe* symbol that passes the output of the first command to be the input of the second command. Two or more commands can be piped on a command line; no intermediate file is needed when you pipe. For example,

```
who | sort
```

passes the output of the `who` command to the `sort` command, and the sorted user list displays on your screen. This is illustrated in Figure 3-10.

Figure 3-10
Piping `who` and `sort`

```
% who | sort
arlington      ttyg7      Aug 8 12:00
belmont        ttyh8      Aug 8 13:01
boston         ttyp5      Jun 4 15:13
cambridge      ttyp0      Jun 4 17:09
center         ttypc      Jun 4 16:43
framingham     ttyh9      Aug 9 14:12
natick         ttypa      Jun 4 18:01
%
```

Multiple transfers create a pipeline. The command shown in Figure 3-10 could be piped to a printer using a command similar to

```
who | sort | lpr -Pkraft
```

where "kraft" is the name of the printer.

Entering multiple commands

You can enter more than one command, each requesting a different operation, on a single command line. Separate each command with a semicolon (;). The semicolon instructs the shell to execute commands sequentially from left to right.

To continue a command entry onto the next line, enter the backslash (\) as the last character on the line. Any spaces placed at the beginning of the continuation line are ignored. If spaces are needed, insert them before the "\." As shown in Figure 3-11, the "\" at the end of the command line tells the shell that the command line continues onto the next line.

Figure 3-11
Multiple commands on one command line

```
% who | sort > current; date >> current;\
 cat current
belmont       ttyp5     Sep 3 15:13
cambridge     ttyp0     Sep 3 17:09
center        ttypc     Sep 3 16:43
natick        ttypa     Sep 3 18:01
Tues Sep 3 18:02:12 CST 1991
%
```

Reusing executed commands

The C shell keeps a record of the commands you give it so you can reissue any command. When you initially enter a command, `csh` assigns it a number, known as its *event number.* You reissue a command by specifying its event number. You can edit or change a command before reissuing it. This recorded list of events is called the *history.*

Predefined is the maximum number of events that `csh` stores for you. This limit is defined with the `set history` command and is usually stored in your .cshrc file. You can set this limit on the command line or use an editor to change the value already set in your .login or .cshrc file. To view the events stored for you, enter

```
history
```

The event numbers appear in the history. Figure 3-12 shows an example of what you see.

Figure 3-12
Viewing a history of events

```
10% history
      1  pwd
      2  ls -alF
      3  mail
      4  cat .cshrc
      5  set prompt = "\!\%"
10%
```

To visually step up and down the entries of history, press

CTRL-p	To step up
CTRL-n	To step down

To execute a command when it steps onto the command line, press **RETURN.** To clear the command line without executing any history command, press **CTRL-u.** The command line is cleared for input.

The number to be assigned to the command you are about to enter can be set to display with your shell prompt, as shown in Figure 3-12, by entering

```
set prompt = "!%"
```

To get this prompt to work, you may have to precede each symbol inside the quotes with a backslash (\). Usually this command is set in your . cshrc file.

csh can save the history list between login sessions. At the end of the current session, the saved history is written to your .history file in your home directory, written as ~/.history. For example, these settings

```
set history=25
```

```
savehist=20
```

instruct csh to save the last 25 commands on the history list and to save the last 20 of them in ~/.history. It is an error when the savehist value is larger than the defined value of set history.

Reexecuting an event

An exclamation point (!) tells csh that you want to reexecute an event. To reexecute the event just completed, enter two exclamation points:

```
!!
```

In the case of Figure 3-12, it is history, so the history command executes.

To reexecute any earlier numbered event saved in the history, enter the exclamation point followed by the event's number. As shown in Figure 3-12, to redisplay .cshrc, enter

```
!4
```

You can also specify an event relative to the current one. If you enter `!-3`, the third command previous to the current one is executed.

Editing reexecuted commands

Suppose you make a typing error in a command and receive an error message. To avoid retyping the entire command, csh lets you include a substitution instruction in the reexecute command.

Figure 3-13 shows a correction sequence. The sequence defined is:

- First exclamation point defines reexecution
- Second exclamation point defines the current event
- `:s` defines a substitution
- `/lw` is the pattern to be changed
- `/ls` is the pattern to be included

Figure 3-13
Fixing an incorrect command

```
10% lw /usr/belmont/graph/doc
lw: Command not found.
10% !!:s/lw/ls
ls /usr/belmont/graph/doc
```

Correcting the most recently executed event, as in Figure 3-13, can be done using an even shorter form. Omit the exclamation points. Type a caret (^), followed by the incorrect pattern, followed by another caret and the substitution pattern. Enter:

```
^lw^ls
```

Controlling processes and jobs

When you log in, you are in your *login shell*, which runs within a *process*. A process is a ConvexOS execution of a program. Multiple processes enable multiple users to run multiple programs, or many users to run the same program when there is actually only one copy of that program. An editor is a typical example of many users running one program. Each request for an editor runs in its own process.

Commands are programs. Whenever the shell interprets a command to be external (versus one of its built-in commands that executes within the shell's process), it starts a new process. Starting a new process is called *forking* a process; this new process is known as a *child process*. The process that forks a child is known as the *parent process*.

While the child process executes, the parent waits inactive until the child finishes executing. When the executing child completes, its process terminates, and its parent, which was inactive, wakes up and prompts for another command.

Multitasking using the shell's job control

Multitasking means running two or more programs (processes) at the same time. When you look at the processes running on your system, usually each one has a different owner. Processes are system-wide entities. A *job* is a process that is tracked as a unit of work related to you, initiated from your terminal and directed to the computer for execution. A job can be a single program or several programs that work together.

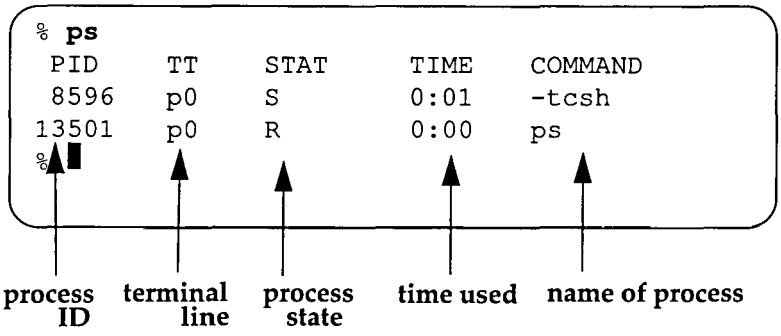
The commands that enable you to control and track a process either as a process or a job are explained in the sections that follow.

Checking on a process

Each process requires a subshell (a copy of the original shell) and a unique *process ID* number, known as its *PID*. As long as a process exists, ConvexOS tracks a process by its PID.

To view the status of your processes, enter the **ps** command as shown in Figure 3-14.

Figure 3-14
Checking your processes



Stated across the screen for each process are these facts:

PID	Unique process identification number
TT	Terminal from which command was entered
STAT	State of the process
TIME	CPU time used
COMMAND	Initiating command

STAT is a five-position field. Its second position states the execution status for a process. The most common run conditions are described below.

R	Running
T	Stopped
S	Active (sleeping for less than 20 seconds)
I	Idle (sleeping longer than 20 seconds)

For further information read the ps(1) man page, by entering **man ps**.

Cancelling a process

With the PID number, you can terminate, known as "kill," a process. The syntax is

`kill` *PID*

Figure 3-15 shows the command sequence that cancels a process.

Figure 3-15
Terminating a process

```
% ps
 PID    TT    STAT      TIME    COMMAND
 8596   p0    S         0:01    -tcsh
01299   p0    R         0:02    sort
13501   p0    R         0:00    ps
% kill 01299
[2]          Done      sort
%
```

Working several jobs at a time

Jobs are your processes that originate from the commands that you enter. If you have several commands (meaning programs) to run, it would be a waste of time to wait for each one to complete before you could begin the next one. Using the shell's *job control* eliminates having to wait for one program to complete before you invoke another. A program can execute either in foreground, which monopolizes your keyboard as it executes, or in *background*, which frees your keyboard so you can work on other jobs.

`csh` job control gives you the ability to stop a process and restart it later as a foreground or a background job. A program that you are about to run can be sent directly to background for processing and later, if you want, you can move it to foreground.

Stopping your current process

In the midst of editing a large file, if you need to read another file or compile another program, you can stop your current job (process) by pressing

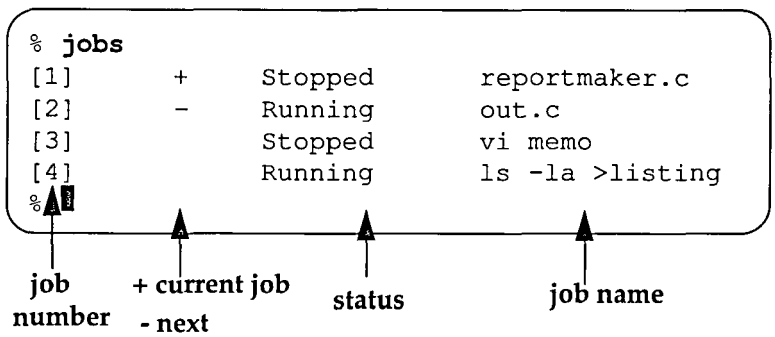
CTRL-z

The shell prompt displays waiting for you to enter another command. The process that you stopped is considered a stopped job; it has been assigned a job number. Each process that you stop or send to background receives a job number in addition to its PID. To check on your jobs, you enter the `jobs` command, which is explained next.

Listing your jobs

Job numbers assigned to your jobs are local to your terminal. These numbers allows you to control your jobs. The first job you stop or put into background is numbered [1], the next job [2], and so on. The command `jobs` displays a labeled list of the jobs initiated at your terminal, as shown in Figure 3-16.

Figure 3-16
A labeled list of your jobs



The + (plus sign) identifies a job as the current job, and the - (minus sign) identifies a job as the next most current job. The most recently stopped job is the most current.

A job is referenced by its job number. Stopping a job, moving it to background, or terminating it, requires that you specify it by its job number.

Sending a new job to background

To send a new job to background, before you press **RETURN** end the command line with an &, as shown in Figure 3-17. After you enter a background request, the shell displays the assigned job number followed by its PID, also shown in Figure 3-17.

Figure 3-17
Sending a job to background

```
% who | sort > current.sys1&
 [1] 01299
% 
```

Sending an executing job to background

You stop a currently executing job before you send it to background. The steps required are:

1. Press **CTRL-z** to stop the job.
2. Enter the **bg** command to send the job to background.

Running a stopped job in foreground or background

Once a job is stopped, you can request that it run in foreground or background.

Job [1] in Figure 3-16 is a stopped job; you run it in foreground by entering

```
%1&
```

or

```
fg %1
```

or

```
fg PID
```

You can run it in background by entering

```
bg %1
```

Stopping a background job

Figure 3-16 shows job [4] to be running in background. The `stop` command suspends a job until you restart it (either in foreground or background) or terminate it. The `stop` syntax is

`stop %`*job_number*

where `%` introduces a job number.

To stop job [4] in Figure 3-16, enter

```
stop %4
```

Bringing a background job to foreground

To move job [2] to foreground, because it is not the current job, you must enter its job number

```
%2
```

To move the current job, the job identified by the +, enter

```
%fg
```

Terminating a job

The `kill` command cancels a job; its syntax is:

`kill %`*job_number*

If you omit the `%`, you will attempt to kill the process assigned that number. The `%` identifies a number as a job number, distinguishing it from a PID number. You can also enter the `ps` command to learn a job's PID, then kill the job using its PID number.

Chapter highlights

Check off the topics that you feel you have mastered. Return to those that require more of your time.

- ❒ **Rules for entering commands**
  - -Syntax: simple and complex
  - -Case sensitivity
  - -Spell correction available on command line
- ❒ **Command information**
  - -man pages contain command descriptions
  - -Help available during the typing of a command
- ❒ **Command use of options and arguments**
  - -Examples of commands
  - -Metacharacters that generate file names
- ❒ **csh completes file names**
  - -Variables expand the match operation
- ❒ **csh lists possible matches to a character pattern**
  - -Lists command names
  - -Lists login names
- ❒ **Redirecting standard input and standard output to another file**
  - -Preventing overwrite during redirection
  - -Appending output to an exiting file
- ❒ **Piping data from one program to another**
- ❒ **Multiple commands on a command line**
  - -Multitasking
- ❒ **How processes and jobs relate**
  - -Checking on processes
  - -Sending a job to background; requesting it to foreground
  - -Stopping a job
  - -Terminating a job

Text editors: creating and altering text files

4

The types of files we create over time will vary as widely as our jobs. Depending on what we find easy to use and considering what was originally used to create the files we work on, we select our favorite editor.

This chapter explains the features of the editors supported on CONVEX systems. Because ConvexOS supports `vi`, like most BSD-based systems, this chapter explains how to use basic `vi` commands and features.

The topics explained in this chapter are:

- What text editors offer you
- Categories of text editors
- Highlights of line editors
- Highlights of screen editors
- How to use the `vi` screen editor

Why text editors?

Text editors provide a convenient way of placing text into a file and changing incorrect characters, words, or lines without having to retype the entire file. Even though a single command can create a file (the `touch` command), and another command can display the contents of a file (the `cat` command), you need a text editor to:

- Add text to a new file
- Insert text in an existing file
- Change text within a file
- Delete text within a file

Files containing text are known as *text files*. Text files contain character groups written in *ASCII* (American Standard Code for Information Interchange) format, an internal binary representation of characters used by many computers for data transfer and storage. Typically, text files are in human-readable form; they contain memos, reports, program source code, messages, or any information entered as a series of characters (alphabetic, numeric, and punctuation).

Text editors should not be confused with word processors, because text editors do not address the stylistic appearance of text. They do, however, allow you to enter format codes for interpretation by programs that control margins, headers and footers, character typefaces, and point sizes. Most editors also accept commands to locate specific lines or words in the text and to add, delete, change, and print lines.

Classifying text editors

Editors can be grouped into three categories:

- Line editors
- Screen editors
- Batch editors

Line and screen editors are *interactive*, meaning you give direct input to the computer and receive an immediate response. You can issue commands and enter text until you are satisfied. Batch editors, on the other hand, are *noninteractive*. They operate without user intervention and perform specific tasks on a file by using a set of commands. An example of a batch editor is `sed`. Batch editors are not discussed in this chapter, but you can refer to the sed(1) man page for more information.

Several types of editors run on CONVEX systems, including the optional COVUEedt, an emulation of the VAX EDT editor for users working under COVUEshell or ConvexOS.

How a text editor handles your text

Text editors, generally speaking, place a temporary copy of the file you request into an *edit buffer* in the computer's memory. Every change you make to a file, whether adding new text, deleting unwanted text, or changing text, is applied to the copy in the buffer. Only when you save the contents of the buffer are the edits applied to the original file on disk, changing the original file.

Using buffers to edit text has these advantages:

- Increased editing speed—Computers work faster on data stored in a buffer memory than data stored on disk.
- Protected original on disk—Only buffered edits are lost if you accidentally delete text, exit the editor without saving the edits, or if the system goes down.
- Defined destination of edits—Text changes can be directed as you like, either to the original file or to a different file.

Remember to save your changes before you exit an editor. If you do not save them, they will be lost. To avoid losing your edits, periodically save your changes.

Facts about line editors

Line-oriented editors display only one line at a time, unless told to display a range of lines. The last line displayed is the *current line.* Even if you display several lines, you can only edit or change the last (current) line displayed.

To move either forward or backward from the current line, you must specify the number of lines for the move. Line editor commands are sometimes difficult to use because you must first identify the characters you want to change (or append) before you indicate the change.

Line editors available on CONVEX systems are `ed`, `edit`, and `ex`; all provide interactive editing. `ex` contains features of both `edit` and `ed`, plus its own. Table 4-1 identifies key facts about `ex`.

Table 4-1
ex highlights

Highlights of the ex line editor
• Counterpart of the screen editor `vi` and extension of the line editor `ed`. You can invoke `ex` either directly or from within `vi`. • Writes, by default, the buffer contents back to the same file, overwriting the old version. Buffer contents are saved periodically in case of system failure. • Prompt is the colon (:). • The current line, marked by a period (.), is the default. The last line is marked by a dollar sign ($). • Commands consist of one or two characters. It works on the current line or a range of lines. • An address is the line number upon which an operation is to be performed. Address symbols are codes that define line numbers. The default is the current line.

For more information on `ex`, `ed`, and `edit`, enter

```
man ex
```

For information on `ed`, enter

```
man ed
```

Facts about screen editors

Screen editors display a "screenful" of text at a time, versus the one line or a range of lines displayed by a line editor. A feature called *scrolling* moves a single line or a screenful of text on to the screen. Moving the cursor up or down causes text to move one line onto the screen at one end, while one line moves off the screen at the other end. Changing text is easier when you can view it within its context.

Screen editors can only be used on video terminals (terminals with screens). If you are working on a hardcopy terminal, such as a teletype, you must use a line editor. The screen editors that run on a CONVEX system include `emacs` (GNU Emacs), COVUEedt (optional product), and `vi`.

emacs ships as part of ConvexOS. An emacs (GNU Emacs) general public license agreement is available from the Free Software Foundation; source code is also available. Table 4-2 highlights key facts about emacs.

Table 4-2
emacs highlights

Highlights of the emacs screen editor
• To invoke emacs, enter **emacs**.
• Recommended use: start emacs only once when you log in, and open multiple windows. (Most editors are invoked to edit only one file at a time.)
• Supports multiple windows and multiple independent buffers, with only one cursor that sits in the selected window. At startup, emacs opens one temporary buffer.
• An echo area holds emacs commands as you type and displays emacs' prompts and messages.
• To view a different file in each window, put a different buffer in each window. emacs needs a buffer for each editing session.
• To save edits permanently, write buffer contents to a file.
• To request online help from within emacs: –For the tutorial, enter **CTRL-h**, then press **t**. –For help, enter **CTRL-h**, then press **i**.

COVUEedt emulates the VAX EDT editor. It runs under ConvexOS and COVUEshell, an optional interface to ConvexOS that emulates the VAX/VMS Digital Command Language (DCL). Emulating EDT, it has line mode and change (screen) mode. In line mode, text editing is done by line. In change mode, screen editing is done in either of two submodes: keypad or nokeypad. COVUEedt has a help facility and a learn facility.

vi is available on many systems because it works on a variety of terminals: cursor positioning does not depend on a keypad or arrow keys, and the display of text is not dependent on a specific type of video terminal. vi is also the default screen-oriented editor delivered with all CONVEX systems. For these reasons, this primer describes vi in more detail.

Using vi

This section explains vi in enough detail to get you started and become reasonably proficient. Most of the basic features and commands are covered. Additional information on commands and options can be found in the following:

- The vi(1) man page, available online or in the *ConvexOS Man Pages for Users*
- *CONVEX vi Quick Reference*

Before you use vi, the TERM environment variable must state the type of terminal you are using. The shell checks your terminal type so it can correctly interpret certain keys that you press. Your system manager has probably set this for you, but you should check by looking at your environment variables, similar to those listed in Figure 4-1; to check, enter

```
printenv
```

Figure 4-1
Sample environment variables

```
% printenv
HOME=/mnt/belmont
SHELL=/bin/csh
TERM=vt100
USER=belmont
PATH=.:/mnt/belmont/bin:/usr/ucb:/bin
HOST=texas
EDITOR=vi
PRINTER=system
%
```

Understanding modes

During a `vi` editing session, you will use both *command mode* and *input mode*. Table 4-3 summarizes what you can do in each mode.

Table 4-3
Operations by mode

Command mode	Input mode
Move the cursor	Add new text
Search for characters	Correct typing mistakes
Change characters, words, lines, and large increments	
Make global changes	
Undo changes	

Press **ESC** to enter command mode. When in doubt as to which mode you are in, press **ESC** until you hear a beep; you are now in command mode.

Altering and positioning text: command mode

In command mode, you issue commands that alter or position text; you cannot add text to a file. For example, pressing the **x** key in command mode deletes the current character, meaning the character under the cursor, as shown in Figure 4-2.

Figure 4-2
Typing in command mode

```
vi is now in command mode; typing the
command 'x' deletes the character under the
cursor. The word now reads as cursr.
```

Typing text: input mode

You can enter input mode through one of two approaches: insert or append. Pressing

- **i** Holds the cursor at its current position and inserts the character preceding the cursor, as explained in Figure 4-3.
- **a** Moves the cursor to the right one character and inserts the character preceding the cursor's new position. Text now typed is acutally appended to the cursor's original position, as explained in Figure 4-4.

Selecting either insert or append needs to be based on where you want the new text placed: before or after the current cursor position.

Figure 4-3
Inserting text

```
This is insert:
With the cursor on the letter "s", press the
"i" to select insert. The cursor stays on the
"s." When the "x" is pressed, it precedes the
letter "s", and the curxsor is still on the
letter "s." Whatever you type appears to the
left of the cursor.
```

Figure 4-4
Appending text

```
This is append:
With the cursor on the letter "s", press "a"
to select append. The cursor moves onto the
letter "o." When the "x" is pressed, it
appears before the "o" and after the "s." The
cursxor is on the "o." Whatever you type
appears to the left of the cursor.
```

The letter "x" goes into text, actually into the edit buffer. You must issue a write command in order for this change to be written to your file.

If the option is set, a status line appears at the bottom of the screen naming the file you are currently editing, the number of lines, and the number of characters in the file. This line is not part of the text file.

Invoking `vi`

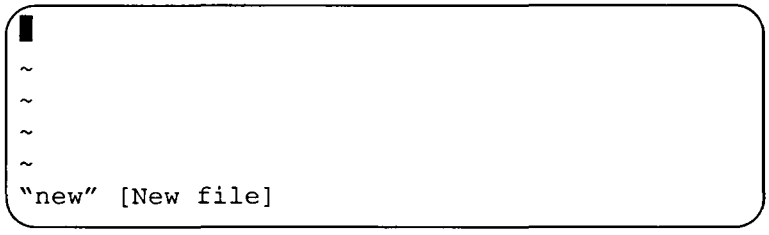
To start a `vi` editing session, enter

`vi` *filename*

where *filename* is either the name of an existing file or a file you want to create. `vi` opens a file in command mode.

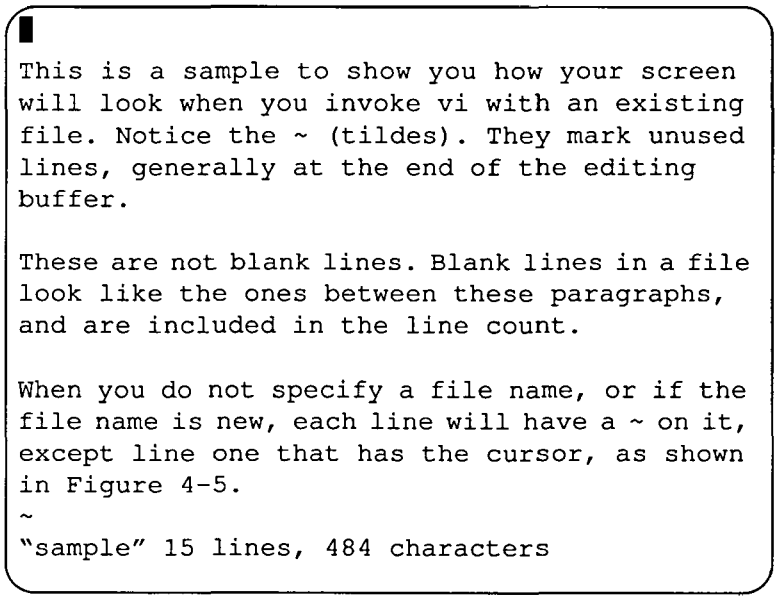
Figure 4-5 shows how your screen looks when `vi` opens a new file.

Figure 4-5
`vi` opens a new file



To compare, Figure 4-6 shows `vi` invoked with an existing file. Note the status line at the bottom of the screen; it is a display and not part of the file.

Figure 4-6
`vi` opens an existing file



Exiting `vi`

Exiting focuses on saving your edits from the temporary buffer to a file or ignoring them and quitting `vi`. Terminology is critical:

- Exiting involves saving changes then closing `vi`.
- Quitting bypasses the save step and closes `vi`.

To make sure you are in command mode, press **ESC** twice, and listen for a beep. Table 4-4 gives the commands you may now type.

Table 4-4
Commands that close `vi`

Key	Action
ZZ	Saves changes, then exits `vi`.
:x	Saves changes, if needed, then exits `vi`.
:wq	Saves (write) changes, then exits `vi`.
:q!	Ignores the buffer contents, bypasses changes; quit `vi`.

Learning the vi command syntax

The command syntax for `vi` is:

`[`*count*`]` *operator* `[`*count*`]` *object*

where

- The required parts are *operator* and *object*.
  - *operator* is the action to be taken—a command.
  - *object* is the entity upon which a command acts.
- The optional information is [*count*]. It defines the number of times to repeat an operation or the number of objects to be affected by the operation.

Type, do not enter, `vi` commands. (A **RETURN** is not needed for commands to execute.)

Frequently used operators and objects

Operators, commands, to try:

`d`	Delete
`c`	Change
`f`	Find
`r`	Replace

Objects frequently used with the above commands:

`w`	Word
`)`	Sentence forward
`(`	Sentence backward
`}`	Paragraph
`l`	Current cursor position

Using count with operators and objects

count is the integer that precedes the *operator*, and in some cases precedes the *object* (also referred to as the target).

To delete only one word, enter:

dw

To backup five sentences, the command is

5(

To request a change within a range starting at the cursor up through the third occurrence of the letter " g, " the command is

```
3ctg
```

where **3** is the number of occurrences to move the cursor, **c** is the change operation, **t** means up to, and **g** is the letter that 3 acts upon. This command inserts a $ after the third occurrence of the letter **g** and switches the mode to insert, so you can begin typing over the characters from the cursor's position up to the $. When the cursor reaches the $, the text following it moves to the right as you type.

To delete several consecutive words with a single command, use count operators, as shown below.

12dw	Deletes 1 word 12 times
d12w	Deletes 12 words 1 time
3d4w	Deletes 4 words 3 times
4d3w	Deletes 3 words 4 times

All of these commands produce the same results—they delete 12 words from a file.

Moving around a file

Moving the cursor around the screen and through a file is something that you do continually. The cursor moves in increments of characters, words, lines, sentences, and paragraphs. Once you move the cursor to the desired place within the file, you issue the commands that enable you to work on the text. The operations explained in the sections ahead are either performed or initiated from within command mode.

Moving the cursor by characters

The most basic cursor movement is to move from character to character. Character movements are controlled by the letter keys (**h**, **j**, **k**, **l**), as defined in Table 4-5.

Table 4-5
Cursor movement by characters

Key	Action
h	Moves the cursor one character to the left. If the cursor is at the beginning of a line, `vi` beeps to let you know your selection is invalid, and the cursor does not move.
k	Moves the cursor to the same position in the line above. If the line above does not extend as far to the right as the cursor, the cursor is placed at the last character of the line above.
l	Moves the cursor one character to the right. If the cursor is at the end of a line, `vi` beeps to let you know your selection is invalid, and the cursor does not move.
j	Moves the cursor to the same position in the line below. If the line below does not extend as far to the right as the cursor, the cursor is placed at the last character of the line below.

Note

If your terminal supports the arrow cursor keys, they move the cursor in the direction of the arrow on the key.

Moving the cursor by words

Although you can move anywhere in a file one character at a time, moving by character increments is impractical if you need to move far from the current position. `vi` supports moving the cursor in *word* increments, where a word is a string of alphanumeric characters separated by a nonalphanumeric character. For example, `THISstringisoneword`, but after the comma&space follows seven words.

Table 4-6 describes word movement keys.

Table 4-6
Cursor movement by words

Key	Action
w	Moves the cursor to the beginning of the next word in the file. When the cursor reaches the end of a line, it moves to the beginning of the first word in the next line; it also stops at punctuation marks.
e	Moves the cursor to the end of the next word in the file. When the cursor reaches the end of a line, it moves to the end of the first word in the next line; it also stops at punctuation marks.
b	Moves the cursor to the beginning of the previous word. When the cursor reaches the beginning of a line, it moves to the first character of the last word on the previous line; it also stops at punctuation marks.
W	Moves the cursor to the beginning of the next word without stopping for punctuation marks.
B	Moves the cursor to the beginning of the previous word without stopping for punctuation marks.
E	Moves the cursor to the end of the current word or the next word without stopping for punctuation marks.

Moving the cursor by lines

`vi` also moves the cursor through a file to the beginning and end of lines. Table 4-7 describes each line movement key.

Table 4-7
Cursor movement by lines

Key	Action
0 (zero)	Advances to the beginning of the current line.
$	Advances to the end of the current line.
RETURN	Advances to the beginning of the next line.

Moving the cursor by other scopes

The `vi` editor also lets you move through a file to the previous and next sentence, paragraph, or section. To `vi`, the end of a sentence is signaled by a period (.), exclamation point (!), or a question mark (?) followed by two spaces. Table 4-8 defines these scopes and their keys.

Table 4-8
Cursor movement by large increments

Key	Action
(	Moves to the beginning of the previous sentence.
)	Moves to the beginning of the next sentence.
}	Moves forward up to the first character of the next paragraph.
{	Moves backward to the first character of the current or previous paragraph.
]]	Moves forward up to the first character of the next nroff section or C function.
[[	Moves backward to the first character of the current or previous nroff section or C function.
*n***G**	Goes to line *n*. If *n* is omitted, goes to end of file.

Moving the cursor by scrolling

Moving the cursor by characters, words, and lines is useful when you can see the destination. But usually there is more text in the buffer than can fit on the screen, and the text you need is not on your screen. To bring more text onto the screen, move the cursor past the top or bottom of the screen. This movement of text is called *scrolling*.

Table 4-9 describes keys that move text to the screen or to a specific position on the screen.

Table 4-9
Scroll keys

Key	Action
CTRL-f	Pages forward one screen.
CTRL-b	Pages backward one screen.
CTRL-d	Scrolls down half a page.
CTRL-u	Scrolls up half a page.
CTRL-e	Scrolls down one line.
CTRL-y	Scrolls up one line.
RETURN	Scrolls current line to top of screen.
z-	Scrolls current line to bottom of screen.
z.	Scrolls current line to center of screen.

Adding text

To add text, you have six options (commands). Each command places the cursor at a certain position within text and changes the mode to input.Table 4-10 describes the cursor placement, hence text placement, of each of the text-entry commands.

Table 4-10
Text-entry commands

Key	Action
a	Appends text after the cursor.
A	Appends text at the end of the current line.
i	Inserts text to the left of the cursor.
I	Inserts text to the left of the first printable character on the line.
o	Opens the line below the current line and positions the cursor for text entry.
O	Opens the line above the current line and positions the cursor for text entry.

In input mode:

BACKSPACE Deletes the character immediately preceding the cursor. Text far from the cursor is quicker to reach if you use command mode to position the cursor.

RETURN Creates a new line beneath the current line. To position the cursor, press **ESC** to leave input mode and enter command mode.

Deleting text

`vi` performs operations in units of cursor movement, meaning by characters, words, sentences, lines, and larger blocks of text. Individually, each of these units is a *target*. Table 4-11 defines delete commands; the `d` command requires a *target*. Targets are defined in Tables 4-6 through 4-8.

Table 4-11
Delete text commands

Key	Action
x	Deletes the character under the cursor.
d *target*	Deletes the target following the cursor.
D	Deletes text from the cursor to the end of the line.
dd	Deletes the current line regardless of the cursor position.

Some **d** *target* examples are:

`dw`	Deletes characters to the end of the word.
`db`	Deletes characters to the beginning of the word.
`dG`	Deletes everything from the cursor to the end of the file.
`d1G`	Deletes everything from the cursor to the beginning of the file.

Searching for text using regular expressions

Searching for text that you name takes the cursor directly to that text. The text that you name can be a single character or a group of characters. The unit of text to search for is known as the *regular expression*.

The search commands are:

/*reg_exp* **RETURN**	Search forward for the regular expression.
?*reg_exp* **RETURN**	Search backward for the regular expression.
n	Repeat previous **/** or **?** command.
N	Repeat previous **/** or **?** command in reverse direction through the file.

Table 4-12 describes the regular expressions.

Table 4-12
Regular expression descriptions

Expression	Action
c	Matches character *c*.
c	Matches character *c*. Used with regular expression characters that have special meanings. See examples below.
^	Matches beginning of line.
$	Matches end of line.
.	Matches any single character.
[$c_1c_2c_3$...]	Matches any character within the brackets.
[^$c_1c_2c_3$...]	Matches any character not within the brackets.
x^*	Matches zero or more occurrences of *x*, where *x* is any regular expression.

Examples of regular expressions are:

CONVEX	Matches the word CONVEX anywhere on a line.
^CONVEX	Matches the word CONVEX when it begins a line.
CONVEX$	Matches the word CONVEX when it ends a line.
CONVEX.$	Matches a line ending in any single character that is preceded by the word CONVEX.
^CONVEX$	Matches a line that contains only one occurrence of the word CONVEX.
^$	Matches an empty line. The line will not be matched if it has any characters on it, including blanks or nonprinted characters.
[Cc]onvex	Matches the word convex or Convex anywhere on any line.
CONVEX\.$	Matches any line ending in CONVEX followed by a period.

Undoing changes

Once in a while you will accidentally delete or change text or use a command that does not do what you thought it would. Fortunately, `vi` provides two undo commands that restore the file to its condition prior to the most recent change.

Undoing the last command change

Immediately after making the change:

1. Press **ESC** to leave input mode.
2. Type **`u`**.

The undo command is also considered as a change to a file; hence, a second undo command reinstates the change.

Restoring the current line to its previous state

Before the cursor leaves the line to be corrected:

1. Press **ESC** to leave input mode.
2. Type **`U`**.

Note

Moving the cursor from the line to be corrected voids the reversal of the changes.

The number of changes made to a line does not govern how this command works. No matter how many changes are made to the current line, they are all undone.

For example, if you delete a word on a line, then move to the beginning of that line and add several characters, then delete the last word in the line, the **`U`** command reverses all these changes. Typing this command a second time does not reinstate the changes.

Ignoring changes

Leaving the editor without applying the changes sitting in the edit buffer is one way to "undo" changes; type

`:q!`

The file remains at the state of the last save; the changes in the edit buffer are discarded (ignored).

Modifying text

One of the main reasons for using a text editor is to change text already stored in the file. Two operations that modify text are:

- Replace modifies characters.
- Change modifies larger targets.

Replacing and changing original text or even a single character, with new text requires access to the edit buffer. In command mode, the edit buffer is closed. Entering input mode opens the edit buffer.

Most commands that modify text change the mode from command to input. As you type, *text* goes to the edit buffer. Read the command descriptions in Table 4-13 carefully so you understand each command. The **ESC** ending a command invokes command mode.

Table 4-13
Replacing, changing, and substituting text

Key	Action
r*x*	Replaces character at cursor with character *x*.
R *text* **ESC**	Replaces original characters, starting at cursor, with *text*.
s *text* **ESC**	Substitutes character at the cursor with *text*.
S *text* **ESC**	Substitutes entire current line with *text*.
c *target text* **ESC**	Changes *target* to *text*.
cc *text* **ESC**	Changes current line to *text*.
C *text* **ESC**	Changes from cursor to end of line with *text*.

Copying and moving text

`vi` provides a way to conveniently move or copy blocks of text within a file. It dedicates a single unnamed buffer where the last deleted or changed text is saved. Text that you copy, called *yank*, goes into this unnamed buffer. Deleted text also goes to this buffer, available for placement elsewhere in the file. You "put" (retrieve) deleted text back into the file. The delete and put commands how you move text within a file. Table 4-14 describes commands that rearrange text.

Table 4-14
Rearranging text

Key	Action
d	Deletes text and places it in the buffer.
y	Yanks (copies) text and places it in the buffer.
p	Puts text stored in the buffer into the file following the cursor.
P	Puts text stored in the buffer into the file preceding the cursor.

The yank and delete commands can put any amount of text into the buffer; *the key is*: this buffer only contains the text from one command, the last command. To better clarify this statement, read the descriptions of the two commands below.

7dw	Performs one delete of seven words, so seven words reside in the buffer.
dw, typed seven times	Performs seven deletes of a single word, so only the last deleted word resides in the buffer.

A yank command works similar to a delete, except the original text remains in the file.

Recovering lost files

Should the system crash, you can recover the file you were in up to the most recent changes. The name of the recover file saved for you displays on your screen the next time you log in.

Go to the directory where you were working when the system failed, and enter:

vi -r *recover_file*

where *recover_file* is the name of the file to be recovered. Lines are rarely lost. `vi` replaces unrecoverable lines with the text "LOST."

To get a listing of all that was saved for you, enter:

```
vi -r
```

When more than one version of a file is saved, `vi` recovers the more recent version. Older saved copies can be recovered after you first save the new copies.

Special features

`vi` has special features that allow you to:

- Specify your terminal type and adjust the screen size
- Adjust indentation, tabs, and margin wraps
- Create macros and abbreviations to complex operations and repetitive command entries
- Edit, consecutively, two or more files
- Use advanced search and replace commands
- Use `ex`-like commands

`vi` offers so much more than what this primer can present. Many third-party manuals clearly explain this editor. The bibliography provided in this primer names documents that contain additional `vi` information.

Chapter highlights

Check off the topics that you feel you have mastered. Return to those that require more of your time.

- ❒ **Facts about text editors**
  - -Line editors and their capabilities
  - -Screen editors and their capabilities
- ❒ **Entering and exiting `vi`**
- ❒ **Using the `vi` editor**
  - -The two modes: command and input
  - -Recovering lost files
  - -Special features
- ❒ **Command mode operations**
  - -Command syntax: operators, objects, and targets
  - -Operations performed or initiated
  - -Moving the cursor around
  - -Undoing changes
  - -Moving the cursor by character, word, line large increments, and scrolling
  - -Adding and deleting text
  - -Searching for text using regular expressions
  - -Replacing and changing text
  - -Copying and moving text
- ❒ **Input mode: typing and altering text**
  - -Entry commands

Organizing and protecting your files

5

ConvexOS uses a hierarchical file structure to organize files. This layered structure places files within directories and directories under directories. Before you create files and place them anywhere, read this chapter to find out how this system works. Smartly creating and arranging directories and files saves you disk space and access time.

The topics explained in this chapter are:

- ConvexOS hierarchical file system
- File types within the file system
- Conventions for naming files
- Paths to directories and files
- Commands for creating and working with directories
- Commands for protecting directories and files

ConvexOS file system structure

In Chapter 4, you learned how to create and edit text files. This chapter explains how files are organized on disk and protected against unauthorized access. Understanding the ConvexOS hierarchical directory structure will enable you to make better decisions as you organize your files and directories. Layering directories so the system can efficiently access your files saves access time and system resources.

How files are organized

Files are organized into *file systems* that reside on mass storage devices, such as disk drives. A file system is a hierarchical organization of directories and files that builds down from a single directory. Each directory can contain files or additional directories. This structure forms what is known as the *directory tree*. Visually, this hierarchical organization is a tree, where the topmost directory is referred to as the root directory or "slash." Root is the directory from which all other directories in a file system hang. Directories, in general, provide an entrance into the next level in the hierarchy.

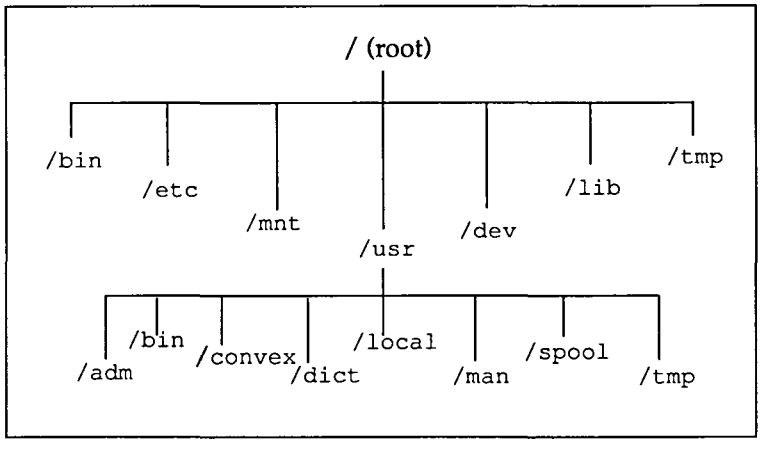
Within this hierarchical arrangement of files are several types of files, of which directories are one type. Four file types are discussed in this chapter; they are:

- *Ordinary files* — Contain data, text, and executable programs.
- *Special device* files — Contain operating information for devices such as printers, disks, tapes, and terminals.
- *Directory* files — Contain ordinary files, special device files, and other directory files.
- *Symbolic link*s — Contain path name information to another file.

Directories

When delivered to a customer, ConvexOS contains the directories shown in Figure 5-1: / (root), /bin, /etc, /mnt, /usr, /dev, /lib, and /tmp. The /mnt and /usr directories are file systems that hang from root. The other directories—/bin, /etc, /dev, /lib, and /tmp— make up what is known as the root file system.

Figure 5-1
ConvexOS directory structure



Most likely your system manager has created additional directories to serve your site's needs.

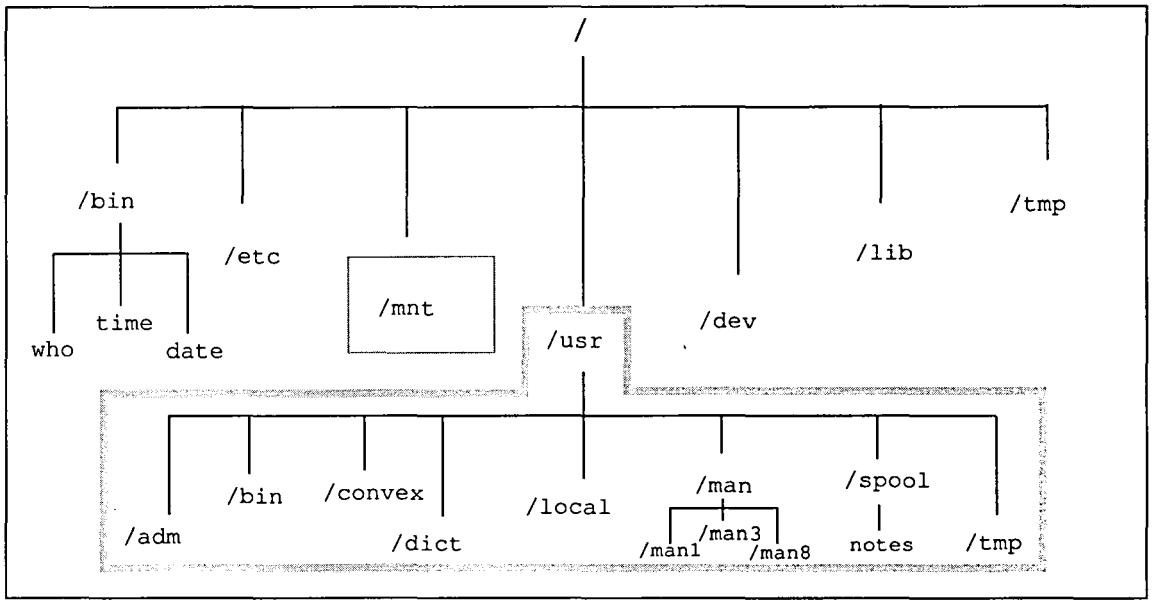
Table 5-1 describes the types of files contained in each of these directories.

Table 5-1
Content of
ConvexOS directories

Directory	Contents
/bin	ConvexOS programs
/etc	System files, essential data, and maintenance utilities
/mnt	General-purpose directory, often contains users' directories
/usr	CONVEX subdirectories and links
/usr/adm	System administration files
/usr/bin	ConvexOS programs
/usr/convex	CONVEX utilities that are extensions to Berkeley distribution
/usr/dict	Dictionary files
/usr/local	Customer-supplied and supported files
/usr/man	Online man pages
/usr/spool	Transient files generated by utilities such as mail and print
/usr/tmp	Temporary file storage
/dev	Special files associated with system devices
/lib	Linkable subroutine libraries for FORTRAN and C
/tmp	Temporary file storage

Beneath the directory /usr, listed in Table 5-1, are its subdirectories. These directories and their files are one file system. Figure 5-2 outlines the /usr file system. /mnt is outlined to show that it too is a file system.

Figure 5-2
Outlining the /usr file tree



Once a file system is part of the tree, any file in the structure from root downward is considered part of the tree. Your home directory is part of the file tree.

Special files

Each I/O device (that is, terminal, disk, tape, or printer) name is interpreted as a file name; hence, ConvexOS treats I/O devices as files. Moving data and programs to and from device files uses commands that work on files. For example, you can pipe the output of the command `ls` to a printer instead of to your screen or to an ordinary file. The command

```
ls | lpr -Prex
```

produces a hard copy listing of the files in the current directory on the printer named "rex" by sending it to the `lpr` command.

Symbolic links

A symbolic link contains a text string that is interpreted as the path name to a real file. A long listing of a directory marks these files with an "l." Another type of link is a hard link that enables you to assign multiple names to a single file. Links are created with the `ln` command. Read the ln(1) man page to learn how to use this command. Creating links is beyond the scope of this primer.

Naming files

Each file has a file name to uniquely distinguish it from another. Conventions for naming a file are minimal. The ConvexOS file system allows up to 256 characters in a name. With few exceptions, almost any character can be used in a file or directory name.

Table 5-2 lists characters to use and those to avoid. Listed as unacceptable is the slash (/) because it represents the root directory and separates directory names in path names, and the hyphen (-) because it precedes a command option.

Table 5-2
Characters in file names

Acceptable characters		Unacceptable characters	
A-Z	uppercase	/	slash
a-z	lowercase	-	hyphen
0-9		&	ampersand
_	underscore	*	asterisk
.	period	?	question mark
,	comma	\	backslash

Note

File names are case sensitive. BOOKFILE.DAT and Bookfile.dat are not the same.

Many ConvexOS users employ the convention of *filename.extension*. The extension, located to the right of the period, serves to further describe the contents of a file. Set a scheme for yourself and use it. It is good practice to assign a name that reflects a file's contents.

Some standard extensions are:

.c	C language programs
.f	FORTRAN language programs
.o	Compiled (object) programs
.ms, .me	Text documents

Employing directories and path names

When you log into your system, you are in your *home directory.* Your home directory is assigned to you when your system manager establishes your account, which includes creating your home directory, setting your password and setting up your work environment as described in Chapter 2. Each user's home directory has a name— usually the user's login (*username*).

Moving to another directory

The route that ConvexOS must travel to reach a directory or a file is known as a *path name.* The directory that you are currently in is your *current working directory.* To view the path name of your current working directory, enter

```
pwd
```

Software responds with the path name of your current working directory. Figure 5-3 is an example. Displayed on your screen would be the path name of your current working directory.

Figure 5-3
Path to current working directory

```
% pwd
/mnt/marble
%
```

What is in a path name?

A path name is the route through the directory hierarchy that ConvexOS must travel to reach a file. Two path name courses are available to you:

- *Absolute path name* is the route that begins at root.
- *Relative path name* is the route that begins at your current working directory.

You determine the course that is most appropriate for what you are doing.

Absolute path names

Absolute path names, because they begin at root, must begin with a slash. To find the absolute path name of a program or utility, use the `which` command. (If the variable PATH does not define the route to a command, `which` gives an error message.) For example, Figure 5-4 locates the `emacs` editor.

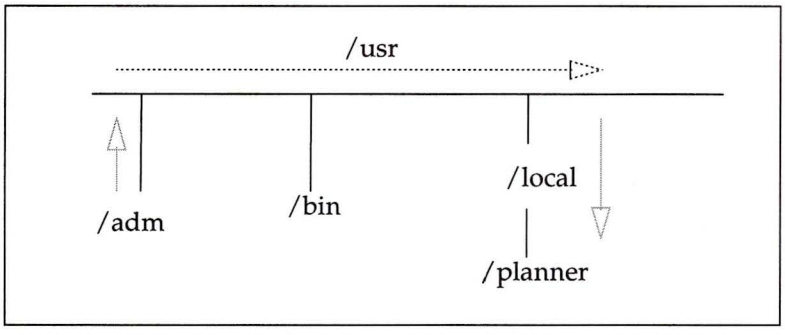
Figure 5-4
Full path name for `emacs`

```
% which emacs
/usr/convex/emacs
%
```

Relative path names

Relative path names specify the location of a file in relation to your current working directory, not in relation to root. For example, in Figure 5-4 your current working directory is /usr/adm, and the command diagramed is
`cd .. /local/planner`

Figure 5-5.
Using a dot notation in a path



Dot notations simplify the typing of path names. These notations are: . (dot) representing the current working directory, and . . (dot dot) representing the *parent directory* (the directory directly above the current directory).

`cd .. /local/planner` is a relative path name, where . . requests a move up from /usr/adm to /usr, and **`/local/planner`** requests a move down from /usr to /usr/local to /usr/planner.

Figure 5-6 shows a listing of the files in the current directory /usr/convex. The **ls -a** command lists all the files, including those files whose names begin with dot. Another requested operation is a **cd ..** up to the parent directory /usr, followed by **pwd** to verify the change.

Figure 5-6
Displaying dot files in a file listing

```
% cd /usr/convex
% pwd
/usr/convex
% ls -a
./
../
contact         notes         syspic
edt             readnotes     tpmount
emacs           rlog          whatis
% cd ..
% pwd
/usr
%
```

Identifying the types of files in a directory

Before you change to another directory, you may want to identify the types of files that belong to that directory. Chapter 3 introduced how the `ls` command lists files within a directory. Specifying `ls -F` causes directories to be marked with a trailing /, symbolic links with a trailing @, and executable files with a trailing *.

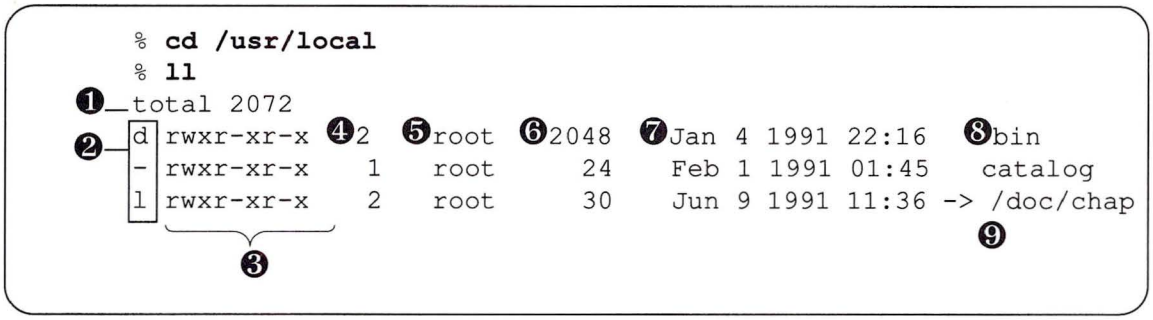
Figure 5-7 shows a directory change using a full path name and a file listing with file type identification from an `ls` with its `-F` option.

Figure 5-7
Identifying the types of files within a directory

```
% cd /usr/local
% ls -F
bin/        chap@       local/      tmp/
catalog     lib/        man/        verde*
%
```

Output from the `ls -l` and `ll` commands is the same. Enter both commands and see for yourself. Figure 5-8 identifies each piece of information provided by these commands.

Figure 5-8
Directory long listing



❶ Directory size	Number of 1024-byte blocks of storage used by the directory.
❷ Type	States whether file is a directory (d), ordinary file (-), or symbolic link (l).
❸ Access	Regulates three user types that are assigned read (r), write (w), execute (x), or denied access (-) to your file.
❹ Links	Number of names, or hard links, to this file.
❺ Owner	Creator or owner of the file.
❻ Size	Number of bytes used by the file.
❼ Date modified	Date file was last modified.
❽ Name	Name of the file or directory.
❾ Link	Path to file that this file is linked to.

Making directories

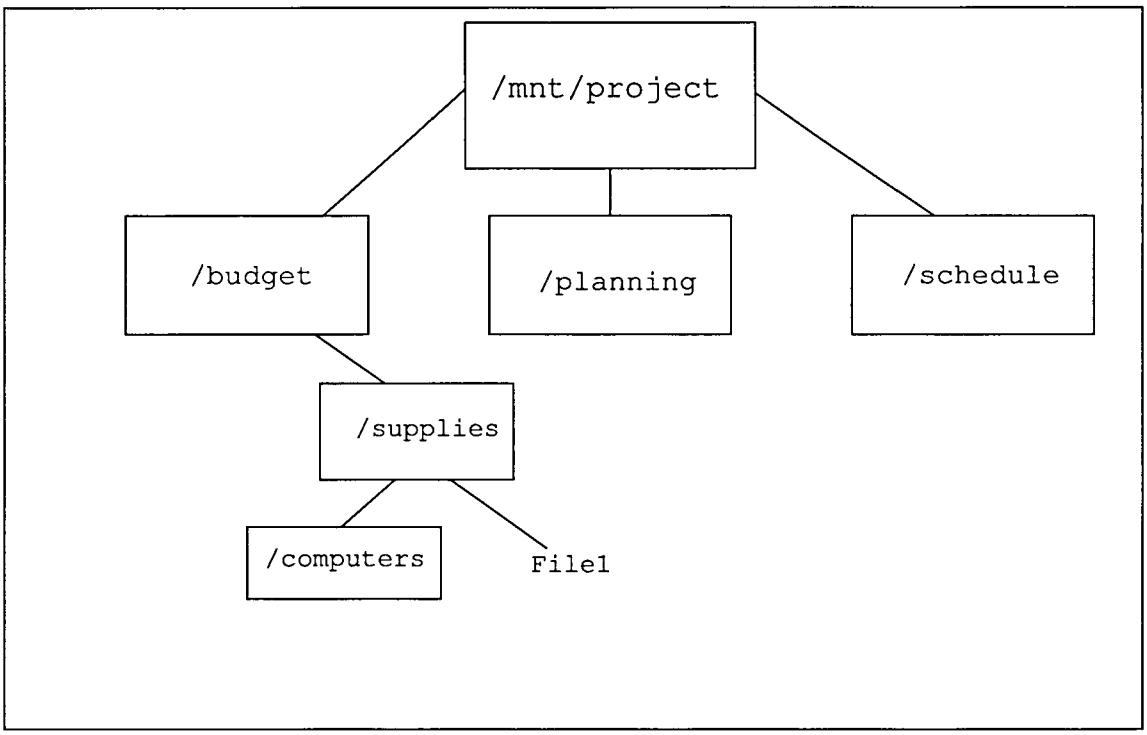
Creating a new directory creates a new level in the overall directory tree. The `mkdir` command lets you build a directory subsystem. The command syntax is:

`mkdir` *dirname*

where *dirname* is the name you select for the directory.

`mkdir` creates a directory file in the current working directory, unless you specify a path name. Figure 5-9 shows the completed directory tree for the subdirectory "project."

Figure 5-9
File tree for /mnt/project



The commands that created this tree are explained in the next section.

Creating a directory tree

The subdirectory "project" was created under the /mnt directory, by entering

```
mkdir /mnt/project
```

Three subdirectories, budget, planning, and schedule were created under /mnt/project by entering

```
mkdir /mnt/project/budget
mkdir /mnt/project/planning
mkdir /mnt/project/schedule
```

Additional directories can be created at any time under "project" or any directory. Depending on where the current working directory hangs within the directory tree determines whether you enter a full path name or a relative path name when creating a directory. Figure 5-10 lists the commands isssued to create the additional directories and files under "project."

Figure 5-10
Creating a directory tree

```
% cd /mnt/project
% pwd
/mnt/project
❶ % ls -F project
% budget/     planning/    schedule/
❷ % mkdir /budget/supplies
❸ % ls budget/supplies
%
❹ % mkdir /budget/supplies/computers
❺ % cd budget/supplies
% vi supplies/file1
❻ % ls -F supplies
computers/ file1
%
```

❶ The files contained in the directory project are listed and marked as to their type.

❷ A directory, supplies, is created under the directory budget.

❸ The `ls` command shows that the supplies directory is empty.

❹ A directory, computers, is created to hang under supplies.

❺ The current working directory is changed to supplies. File1 is created with the editor `vi`.

❻ The `ls -F` command lists the contents of the supplies directory.

Copying files

The copy command–cp—allows you to make a duplicate file under a different name within the same or another directory. Or, you can make a duplicate, using the same name, and store it in another directory. In either case, the original does not change.

The syntax for a copy is:

cp *old_filename new_filename*

where *old_filename* is the original file and *new_filename* is the duplicate.

To copy a file from another directory to the current directory, the . (single dot notation) provides a shortcut. Figure 5-11 shows an example of using dot notation to represent the current working directory, while retaining the existing file name.

Figure 5-11
Copying using dot notation

```
% pwd
/mnt/marble
% cp /mnt/jones/jones_memo .
% ls -F
file1       prog.1*      jones_memo
file2.c     prog.1.c     schedule
%
```

Moving files

The mv command enables you to remove a file from where it is and place it in another directory; or, you can rename the file and leave it where it is.

The syntax for a move is:

mv *old_filename new_filename*

where *old_filename* is its original name (which includes its location), and *new_filename* can be either its new location (destination) or its new name.

Note

Make sure the *new_filename* is unique; mv overwrites an existing file.

Figure 5-12 shows that knowing the current working directory is not always relevant. In this example, within the directory "jones," the file memo is renamed to jones_memo.

Figure 5-12
Moving a file

```
% pwd
/mnt/marble
% ls
user_list
% mv /mnt/jones/memo /mnt/jones/jones_memo
% cd /mnt/jones
% ls
budget      doc          jones_memo
catalog     employees    schedule
%
```

Deleting directories—only

When you no longer need a directory, you should delete it to free disk space. Only empty directories can be deleted with the `rmdir` command. The syntax of `rmdir` is

`rmdir` *dirname*

Figure 5-13 shows how to remove a directory.

Figure 5-13
Removing an empty directory

```
% ls -F
project1/   project2/
% rmdir project2
% ls -F
project1/
%
```

If the directory you want to delete contains even one file, `rmdir` prints an error message. `ls` and `ll` do not list files that begin with a dot (.). Use `ls` or `ll` with the `-a` option to list dot files. Figure 5-14 shows an example of the `ls -a` command.

Figure 5-14
Listing all entries

```
% ls -a
.login
.cshrc
file1
file2
%
```

Deleting all files

The `rm` command deletes files without notifying you. To avoid an accidental removal of your files, include the interactive [`-i`] option that causes `rm` to request confirmation from you before it removes a file.

The syntax is:

`rm -i` *filename*

The recursive option, `-r`, instructs `rm` to delete a nonempty directory and all of its files. The syntax is:

`rm -r` *dirname*

Protecting your files

Access to your files by other users is something that you, as owner, can restrict and adjust. Each ConvexOS file and directory has access settings that specify who can access it and the degree of access allowed. Naming one of the three possible types of users supplies the who, and one of the four possible access permissions defines the degree of access.

Access permissions

ConvexOS provides file and directory access permissions as described in Table 5-3. Access permissions affect directories differently than files.

Table 5-3
Access permissions

Access	Ordinary files	Directories
Readable (r)	Can view file contents.	Can list the contents of a directory.
Writable (w)	Can change file contents by editing or overwriting.	Can create files and directories. Files can be deleted regardless of their write permissions, if sticky bit is not set. Subdirectories, only if empty, can be deleted regardless of their write permissions.
Executable (x)	Can execute the program.	Can traverse through the directory.
None (-)	All access permission denied.	All access permissions denied.
Sticky bit (t)	Not applicable.	Only owners of files within this directory can delete them.

Note

If a directory is writable, its files can be deleted regardless of their access modes. Prevention of this is to set the sticky bit on the directory.

An octal value is held by each permission, as shown in Table 5-4.

Table 5-4
Octal values of access permissions

Access	Octal value
Readable (r)	4
Writable (w)	2
Executable (x)	1
None (-)	0 (zero)
Sticky bit (t)	1000 (for directories only)

Types of users

ConvexOS organizes users into the three types described in Table 5-5. The number preceding each user type associates the user type with its access permissions likewise numbered in Figure 5-15.

Table 5-5
Users of ConvexOS

User	Description
❶ User (owner)	File or directory owner (maybe creator) and definer of access permissions.
❷ Group	Users in the same group as the owner (often related by task or department); groups are defined in the /etc/group file by the system manager.
❸Other	Any user that is not the owner or group member.

Because changing to another user's home directory is quite easy, protecting your files is important. `cd` followed by the tilde (~) and a user's name allows you to move to another user's home directory. The syntax is

`cd` *~username*

Entering `cd` with no arguments returns you to your home directory.

Patterns of access permissions

Read, write, and execute permissions form a pattern. In Figure 5-15, the numbers ❶, ❷, ❸, and ❹ associate each permission unit with its user type described in Table 5-5 and the User column in Table 5-6, where the octal values for that user's permissions are written out.

Figure 5-15
User access permissions

```
% cd /usr
% ll
total 2072
-rwxrwxr-xt   2   root    24      Jan 4 1991 22:16   tax
drwxr-xr-xt   1   root    20484   Feb 1 1991 01:45   bin/
              ❶ ❷ ❸   ❹
```

Octal values of access permissions

For each user type, the octal values of the assigned access permissions are summed. The sum is a single octal value, giving a user type a single octal value. Table 5-6 sums the permissions shown in Figure 5-15.

Table 5-6
Octal values assigned to access

User	Access permission	Octal value
❶ User	rwx |--- |---	700
❷ Group	|---|r-x|---	050
❸ Other	|---|---|r-x	005
		755
❹ Sticky bit restricts all user types	|---|---|---t	1000
		1755

Assigning and changing file access

Now that you understand the types of access available for assignment, you are ready to learn how to control them. You use the `chmod` command to change permissions only for the files and directories that you own.

`chmod` has two modes: *symbolic mode* and *absolute mode*. The one you use is strictly a matter of preference:

- *Symbolic* changes permissions relative to the current permissions.
- *Absolute* changes permissions irrespective of the current permissions.

Symbolic mode

When using symbolic mode, you change the permission pattern by adding new privileges to or removing them from a file.

The syntax for symbolic `chmod` is

`chmod` [*who*] *op_code permission filename*

where

- *who* is the type of user identified by:
  `u` = user (owner) of the file
  `g` = group in which the owner is a member
  `o` = other (users not classified as user or group)
- *op_code* is the operation to be performed identified by:
  `+` add the specified permissions to the existing ones
  `-` subtract the specified permissions from the existing ones
- *permission* is the type of access granted identified by:
  `r` = read permission
  `w` = write permission
  `x` = execute permission
  `-` = access denied
  `t` = sticky bit (directories only)

For example, to assign only read and write access, enter

```
chmod g=rw
```

To add read to the existing permissions, enter

```
chmod g+r
```

For example, to remove read and write permissions from group and other, enter

chmod go-rw *filename*

Figure 5-16 is an example of this command. Note that the `ls -l` command shows the permissions prior to and after the symbolic `chmod` change.

Figure 5-16
Symbolic `chmod`

```
% ls -l example
-rw-rw-rw- 1 smith 1236 Jun 10 12:43 example
% chmod go-rw example
% ls -l example
-rw------- 1 smith 1236 Jun 12 11:22 example
```

Omitting the *who* parameter causes the change request to be applied to all three user types.

Absolute mode

When using absolute mode, the octal value of an access permission is assigned to a user type. (Table 5-4 lists the access permissions with their octal values.)

The format of the absolute `chmod` command is

`chmod` *octal_value filename*

where *octal_value* is the octal value of the access permission to be assigned.

Figure 5-17 shows an example of an absolute `chmod` change.

Figure 5-17
Absolute `chmod`

```
% ls -l example
-r--r--r-- 1 smith 1236 Jun 10 12:43 example
% chmod 600 example
% ls -l example
-rw------- 1 smith 1236 Jun 12 11:22 example
```

Looking at Figure 5-17 and referencing Table 5-4, the permissions `-r--r--r--` have an octal value of 444. Applying `chmod 600` to the file "example" changed the permissions to read: `-rw-------`.

Chapter highlights

Check off the topics that you feel you have mastered. Return to those that require more of your time.

- ❒ **ConvexOS file structure**
- ❒ **Organization of files**
  - -Types of files
  - -ConvexOS directories and their contents
- ❒ **Naming files**
  - -Case sensitive
  - -Length
  - -Acceptable and unacceptable characters
- ❒ **Directories**
  - -Path names: absolute and relative
  - -Long listing of directory contents
  - -Making a directory
  - -Moving (changing) to another directory
  - -Copying and moving files
  - -Deleting directories
- ❒ **Files**
  - -Restricting access: symbolic and absolute
  - -Listing by directory
  - -Deleting

Printing files

6

In addition to reading a file displayed on your terminal screen, there are times when you need a paper copy. Two terms are interchangeable when referring to a paper copy: one is *printout*, the other is *hard copy*.

This chapter explains the commands that enable you to:

- Print your files
- Check the status of a file (job) in a print queue
- Move a file from one queue to another
- Remove a file from a queue

Sending files to print

Most CONVEX systems have more than one printer. Typically, you will use the `lpr` command to send a file to a printer. Ask your system manager how the printers are set up on your system. For each printer available to you, get its name, the type of input it handles, and the type of output it produces. Some printers only accept ASCII format (program listings), while others accept PostScript format and produce typeset documents and graphics.

A single printer can serve many print requests, referred to as print jobs, but only one at a time. To service multiple print requests, each printer has a holding file called a queue, where jobs wait to be printed. Print jobs feed to a printer in the order that they enter a printer's queue.

Print jobs can go either to the default printer set with the environment variable PRINTER or to a printer that you specify. In the sections ahead, you will learn how to:

- Submit a job to a printer
- Check a printer queue to determine your job's status
- Move your print job to another queue
- Remove a job from a queue

Queuing files

The `lpr` command prints the contents of a file on the *default printer*. The syntax for the `lpr` command is

`lpr` *filename*

which submits your file (print job) to the default printer's queue. Your print command executes in foreground, but the print job executes in background mode. Any command that executes in foreground has control of your terminal keyboard; by executing a job in background, your keyboard is free for you to issue other commands.

To read the lpr(1) man page, enter

```
man lpr
```

Metacharacters are valid in file names specified in a print request. Chapter 3 discusses metacharacters. An example of metacharacters as part of a file name in a print request is

```
lpr chap*
```

This example requests all files that begin with "chap" and followed by any character, no character, or any group of one or more characters. Files with the name chapter1, chapstick, and chaps would be queued.

Directing a file to a specific printer

The `-P`*printer* option of the `lpr` command identifies the printer that is to produce the hard copy.

To send a file to a specific printer, the syntax is

`lpr -P`*printer filename*

For example, to send the file "schedule" to the printer "kraft," enter

```
lpr -Pkraft schedule
```

Queuing several files

To queue multiple files in a single print request, the syntax is:

`lpr [-P`*printer*`] [`*filename...*`]`

where the printer can be the default printer (omit printer name) or one that you specify with **-P***printer*.

For example, to print the three files—one, two, and three—on the printer "kraft," enter

```
lpr -Pkraft one two three
```

Requesting multiple copies and page breaks

You can define, by file, the number of copies to be printed and whether text is to be printed according to the predefined printed page length and width. (The defaults for where pages break are 72 characters wide by 66 lines long.) These settings are defined by the `pr` command. Electing page breaks includes a header at the top of each page stating: the current date, file name, and page number.

To queue multiple copies with page breaks, the syntax is

`lpr [-P`*printer*`] [-#`*num*`] [-p] `*filename*

For example, to print three copies, with page breaks, of the file "schedule" on the printer "star" enter

```
lpr -Pstar -#3 -p schedule
```

Checking a queue for your print job

The `lpq` command reports the following for each job within a printer queue:

- Current rank within the queue
- Owner's name
- Identification number
- File name
- File size

`lpq` has an option that specifies a specific printer:

`lpq` `[-P`*printer*`]`

where `[-P`*printer*`]` is the name of the printer.

Figure 6-1 shows the contents of the queue "kraft."

Figure 6-1
Job status within a printer queue

```
% lpq -Pkraft
kraft is ready and printing
Rank    Owner     Job    Files     Total Size
active  barnes    114    timers    4500 bytes
1st     mars      115    minutes   998 bytes
2nd     case      116    phrases   13399 bytes
%
```

A job's number is part of its queue status information, as shown in Figure 6-1 listed under the heading "Job."

A job's number is needed by the `lpmv` command to move a job from one queue to another, and by the `lprm` command to remove a job from a queue.

For example, the command that removes job 116 shown in Figure 6-1 is

```
lprm 116
```

Moving a job from one queue to another

You move one job or all your jobs from one CONVEX printer queue to another CONVEX printer queue with the `lpmv` command. Options enable you to name a specific printer when the default printer is not used and to request a move of one job or all your jobs to another printer.

The syntax for the `lpmv` command is:

`lpmv [-a] [-P`*printer*`] [`*job#...*`] `*destination_printer*

where the *-a* option requests all jobs owned by requestor. Jobs are selected by the requestor's UID and the name of the system where the `lpr` command was invoked. This option saves you from having to list job numbers, in case you want to move one or two, but not all the jobs you might have queued.

The command line shown within Figure 6-2 moves job 116 from the "kraft" queue to the queue of printer "clipper."

Figure 6-2
Moving a job to another queue

```
% lpmv -Pkraft 116 clipper
kraft is ready and printing
Rank    Owner      Job   Files    Total Size
active  barnes     114   timers   4500 bytes
1st     mars       115   minutes  998 bytes
2nd     case       116   phrases  13399 bytes
%
```

Removing a job from a queue

You can remove *only your jobs* from a print queue. The `lprm` command removes one job or several jobs from a queue. Entered without an option, `lprm` removes the currently active job, only if you own it. `lprm` announces the name of a file it is removing.

The syntax for the `lprm` command is:

`lprm [-P`*printer*`] [-] [`*job#...*`]`

The - option selects for removal all the jobs owned by the requestor, eliminating the need to list the job numbers. This selection is determined by the requestor's login name and the name of the system where the `lpr` command was invoked.

Chapter highlights

Check off the topics that you feel you have mastered. Return to those that require more of your time.

- ❒ **Entering print requests**
  - -Using metacharacters in file names
  - -Using the default printer
  - -Directing a job or jobs to a specific printer
  - -Sending several files in one print request
  - -Asking for multiple copies
  - -Asking for page breaks
- ❒ **Checking a printer queue**
  - -Displaying job numbers
  - -Checking the queue for a job's status in the queue
- ❒ **Moving a job or jobs to another queue**
  - -Identify the job by its number
  - -Identify all your jobs by your user name (as owner)
- ❒ **Removing a job or jobs**
  - -Identify a job by its job number
  - -Identify all your jobs by your user name (as owner)

Communicating with other users

7

This chapter describes how you can get information about other users connected to your system and how to contact them electronically. More specifically, the topics explained in this chapter are:

- How to retrieve information about other users
- How to send, receive, and handle mail messages
- How to send and receive system-wide messages
- How to interactively talk from your terminal with other users

Information about other users

Want to know who is currently logged in and some information about each user? Five commands—`users`, `uptime`, `who`, `w`, and `finger`—explained in the sections that follow, display such information.

Listing users logged in

`users` lists, one right after the other across your screen, the user name (login names) of each user currently logged in. Figure 7-1 shows sample output from entering

```
users
```

Figure 7-1
Users logged in

```
% users
belmont canton dedham eastham hingham
% 
```

Who is on the system

`who` lists all currently logged-in users by login name, their terminal, and the date they logged in. Figure 7-2 shows that `who` listings are sorted by terminal number.

Figure 7-2
`who` command output

```
% who
belmont     ttyp0       Aug 8 13:28
canton      ttyp1       Aug 6 22:03
dedham      ttyp2       Aug 7 18:48
eastham     ttyp3       Aug 8 12:01
hingham     ttyp4       Aug 8 08:05
% 
```

`who` has an argument [`am i`] that is particularly helpful when a terminal is available to several users. If someone has already logged in and another user comes along to use the terminal, entering **`who am i`** displays the login name of the user currently logged in.

Facts about the system and its current users

The `uptime` command displays a snapshot, as shown in Figure 7-3, of the current activity on the system, including the number of users currently logged in.

Figure 7-3
Snapshot of current system activity

```
% uptime
 5:36pm up 4 days, 20:02, 5 users, load average: 0.29, 0.21, 0.31

%
```


Displayed on one line across your screen is the:

- Current time of day
- Length of time the system has been up
- Number of users logged in
- Load averages: three values based on the number of jobs queued over 1, 5, and 15 minutes

`uptime`'s one-line summary is the headline displayed above the user information output by the `w` command.

The `w` command also summarizes current system activity, but focuses on what each user is currently doing. Figure 7-4 is a sample display.

Figure 7-4
System activity focusing on users

```
% w
 5:36pm up 4 days, 20:02, 5 users, load average: 0.29, 0.21, 0.31
User     tty    login@     idle    JCPU    PCPU     what
belmont  ttyp0  1:28pm     2:55    33      1        -csh[belmont]
canton   ttyp2  6:48pm     7:15     4      1        -csh
dedham   ttyp4  8:05am     1:18                    -csh
eastham  ttyp5  9:11am     1:01    2:23    1:16     emacs -nw
hingham  ttyp8  10:04am    2:07      14       4     -ksh[hingham]
%
```

Following the headline, for each user these facts display:

- User's login name
- Terminal's ID
- Time of day the user logged in
- Number of minutes elapsed since last input
- CPU time used by all processes (JCPU)
- CPU time used by all current processes (PCPU)
- Name and arguments of the current process

More user-specific facts

The `finger` command has two forms:

- **Without arguments**—Provides an expanded `who` listing, including each user's full name, idle time, office and phone numbers; enter

  `finger`

- **With one or more user names as arguments**—Provides facts about any authorized user. The user name you request does not have to be that of a user currently logged in; enter

  `finger` [*username ...*]

  Several users can be fingered in one `finger` command. After each user name, insert a space before typing the next user name.

Figure 7-5 shows the output for user "belmont."

Figure 7-5
`finger` output for a named user

```
% finger belmont
Login name: belmont             In real life: chadwick belmont
Directory: /mnt/belmont         Shell: /bin/csh
On since Aug 28 11:04:50 on ttyp3 from stonehedge
Project: your project here...
Plan:
no plans yet.
%
```

Mailing other users

Electronic mail, also known as *email*, refers to a system that transmits memos and messages over computer networks. At your terminal, you compose a message and send it to one or more users. With a multitasking system such as the ConvexOS, mail can be delivered and announced while the sender works on other tasks. Recipients can read their mail when it is convenient for them.

The ConvexOS electronic mail system enables you to:

- Send a message to one or more users
- Receive mail
- Delete mail
- Forward mail
- Save mail to a file for later reference
- Print mail messages

Notifying you of new mail

When you log in, by default, the `mail` system notifies you of any new mail messages received since your last terminal session. When you have new mail, `mail` displays the message `You have new mail` when you log in. If you have unread messages received during previous terminal sessions, `mail` displays the message `You have old mail`. If both new and unread messages exist when you log in, only the new mail message, shown in Figure 7-6, displays.

Figure 7-6
New mail notice

```
You have new mail.
% ■
```

Notification of incoming mail during a terminal session depends on the setting of the `biff` command. The `biff` command informs the system as to whether or not you want to be notified when mail arrives. `biff` has two settings. Generally, a `biff` setting is written in your .login file to be executed during each login. The `biff` settings are:

`biff n` This is the default. You receive the notice shown in Figure 7-6 whenever new mail arrives.The longer notification of `biff y` is disabled.

`biff y` Send immediate notification of a new message. Its header and the first few message lines display on your screen. This display interrupts output from the current process.

To view your current `biff` setting, as shown in Figure 7-7, enter

```
biff
```

Figure 7-7
Checking your `biff` setting

```
% biff
is n
%
```

Setting `biff` can be done either:

- At the shell prompt by entering `biff` and its setting
- From within your .login file using an editor

Entering mail

You can enter `mail` at any time, not just when you log in. At the shell prompt, as shown in Figure 7-8, enter

```
mail
```

Figure 7-8
Requesting the `mail` utility

```
% mail
```

`mail` first displays the summary lines of unread and new messages as shown in Figure 7-9.

Figure 7-9
Beginning a mail session

```
% mail
"/usr/spool/mail/hingham": 4 messages 3 new 1 unread
>U 1 hingham Wed Aug 10 17:03 12/316 "Project summary for July"
 N 2 hingham Wed Aug 10 17:04 13/313 "Project schedule for July"
 N 3 hingham Wed Aug 10 18:64 33/679 "Weekly status starscape"
 N 4 hingham Wed Aug 10 17:05 15/317 "System backups for July"
& 
```

These summary lines are explained later in this chapter. Beneath them displays the `mail` prompt and the & (ampersand), followed by the cursor.

Leaving mail

Enter either the **q** or **x** command to terminate a mail session. The one you choose depends on how you want `mail` to close the session.

- Entering **q** has the same effect as pressing **CTRL-d**:
  - Quits the `mail` utility
  - Returns you to the shell prompt
  - Processes deletions requested during the `mail` session
  - Writes read messages to your local mailbox (~/mbox) and shows a status of `U` for those unread
- Entering **x**
  - Exits the `mail` utility
  - Returns you to the shell prompt
  - Cancels deletions requested during the `mail` session
  - Holds messages in your system mailbox

Enter **x** when you change your mind about message processing or when something seems to be going wrong. This command restores deleted and saved messages and inserts them back into your system mailbox.

Options set in the .mailrc file in your home directory can affect what gets written and when from your system mailbox to ~/mbox. Read the mail(1) man page to learn about these options.

Processing mail

Within mail, you issue commands to read, send, reply to, delete, store, and print messages. Table 7-1 lists commands for these activities, as well as commands that are discussed later in this chapter.

Table 7-1
`mail` commands

Action	Press
Display menu of `mail` commands	**?**
Display next message	**RETURN**
Display previous message	**-** (hyphen)
Redisplay message	**p**
Redisplay headers	**h**
Respond only to sender of message	**R**
Respond to all senders and recipients	**r**
Delete message	**d**
Undelete messages	**u**
Send to *username*	**m** [*username...*]
Save message in *filename*	**s** *filename*
Write message without header to *filename*	**w** *filename*
Exit mail; restores deleted messages	**x**
Quit mail; processes deletions	**q**

To view a list of commands, each with a brief description, at the & enter

?

Sending mail

You can send mail either from:

- The shell prompt (%)
- Inside mail at the mail prompt (&)

From the shell prompt

To send mail to one or more users, as shown in Figure 7-10, enter

mail *username* [...]

Figure 7-10
Mailing from the shell prompt

```
% mail belmont canton dedham
```

From the mail prompt

To send mail to one or more users, as shown in Figure 7-11, enter

m *username* [...]

Figure 7-11
Mailing from the mail prompt

```
& m belmont canton dedham
```

Note

Enter `finger` to learn a user name.

Composing mail

A sample mail message is shown in Figure 7-12. Boldface text identifies what you type; regular text is what mail displays to you.

Figure 7-12
Sample mail message

```
& mail student
Subject: mail to user student
To tell mail that you have finished typing the
body text, press the return key to move to a
new line. In the first character position, as
the only character on that line, type a period
(.).

Mail next displays Cc:; type the user names of
those to receive a copy of this message. After
the last name, press the return key. To ignore
Cc: press the return key, which tells mail to
send the message.
.
Cc: wilson
&
```

Changing parts of a message, ending a message, and aborting a message are done with specific `mail` commands. These commands are listed in Table 7-2 .

Table 7-2
`mail` commands

Within a message	Press
Start a new text line	RETURN
Abort a message	CTRL-c CTRL-c
On a new message line	**Press**
Add names to To:	~t [*username* ...]
Replace subject text with new text	~s *new_subject_text*
Redisplay message with changes	~p
End a message	CTRL-d
End a message	.

Mailing files

Files can be mailed from within a message or at the shell's command line. You choose the method according to your need.

- **At the shell prompt**—Input redirection.

  Input redirection enables you to mail a file from the command line. You name the file and the recipient without having to retype text or enter `mail`. Figure 7-13 tells `mail` to find the file "proposal" within the current directory, then send a copy of it to the user "student."

Figure 7-13
Mailing a file from the shell

```
% mail student < proposal
% 
```

- **Within a message**—The `~r` *filename* command.

  This command tells `mail` to include the contents of a file at the cursor location. In Figure 7-14, the file to include is "week_12" from the current directory. `mail` confirms your request directly below the command. Neither your `~r` *filename* command nor the confirmation appear in the mailed message.

Figure 7-14
File included in a message

```
& mail student
Subject: including file in message
Here is the latest copy of my report.

~r week_12

"week_12" 78/230

You will receive all updates.
.
Cc:
& 
```

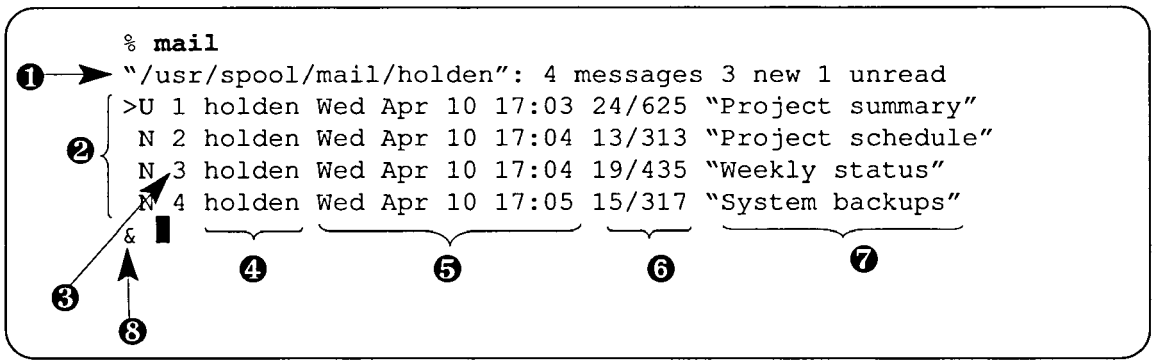
Reading mail messages

From the shell prompt, to access your mail, enter

```
mail
```

`mail` informs you of any new or unread messages. Figure 7-15 is a sample of the first display you see. An explanation of the information presented on this screen follows the figure.

Figure 7-15
Starting a mail session



❶ Header line summarizing status of all messages.

❷ Summary column for each message:

> marks the current message: the next message to be read.

Blank identifies a read message (As you read your messages, enter the **h** command to redisplay the headers. Note that the summary line of each read message beings with a blank.)

U identifies old message still unread.

N identifies new messages (received since you last read your mail.)

❸ Message number used to reference the message.

❹ Name of sender.

❺ Date and time message was received.

❻ Size of message, includes header plus text:

—First number represents number of lines.

—Second number represent number of characters.

❼ Text copied from each message's `Subject:` line.

❽ `mail` prompt (`&`) where you enter `mail` commands.

The > marks the *current message*, the next message to be read.

- To read the current message, press **RETURN.**
- To read a message other than the current message, enter *message_number* **RETURN**

  where *message_number* is identified by the ❸ in Figure 7-15.

Storing read messages

Read mail, by default, goes to your local mailbox located in your home directory (referenced as ~/mbox) unless you name a file where you want it stored.

Reading messages stored in ~/mbox

Messages saved in ~/mbox are still treated as mail.

- `mail` commands work on them.
- Brief descriptions, the headers, list just as if they were new or unread.

To read mail stored in ~/mbox, as shown in Figure 7-16, enter

`mail -f`

Figure 7-16
Reading from your mbox

```
% mail -f
"/mnt/belmont/mbox": 2 messages
>1 letra@raydon Thu May 8 9:04 38/1100 "news"
 2 map@star Tue May 28 23:17 58/2411 "xdump"
& ▮
```

Storing a message in a file

To include the header when saving the current message, use this syntax

`s` *filename*

To omit the header when saving the current message, use this syntax

`w` *filename*

Figure 7-17 shows saving the current message to the file "log," and the confirmation that this save was done.

Figure 7-17
Saving a message to a file

```
& 6
From belmont Mon Sep 16 16:38:22 1991
Received: by sun.com (5.64/1.28)
      id AA20987; Wed, 18 Sep 91 16:42:38 -0500
Date: Wed, 18 Sep 91 16:42:38 -0500
From: belmont (chadwick belmont)
Message-Id: <9109142114.AA20987@sum.com>
To: student
Subject: Saving this message to a file

Organize your messages in files to match your
projects. You can read files that contain
messages with the mail -f filename command.
.
& s log
"log" [New file] 17/414
&
```

Saving a message other than the current one adds the message number to both forms of this syntax.

For example, to save message number 2 and its header to the file "log," the command is

```
s 2 log
```

Saving a message to an existing file appends it to the contents of the named file, as shown in Figure 7-18.

Figure 7-18
Confirmation of file saved

```
& s 2 log
"log" [Appended] 14/493
```

Reading mail saved to a file

To read mail saved to a file, enter

mail -f *filename*

When you access a message file with this command, you enter `mail`. The summary line for each message stored within the file displays. You can select a file to read, and you can send mail.

Replying to mail

To reply to a message, you must be reading it. The reply command you enter defines the receiver as:

r	Senders and Cc: recipients
R	Sender only

Figure 7-19 shows the reply command to be **R**. First message **6** is read, then a reply is initiated to the sender.

Figure 7-19
Replying only to the sender

```
& 6
From belmont Mon Sep 16 16:38:22 1991
Received: by sun.com (5.64/1.28)
      id AA20987; Wed, 18 Sep 91 16:42:38 -0500
Date: Wed, 18 Sep 91 16:42:38 -0500
From: belmont (chadwick belmont)
Message-Id: <9109142114.AA20987@sum.com>
To: student
Subject: Responding to a message
Status: R

The recipient, student, uses the R command
to respond only to the sender. Using the r
command would send the reply to the sender and
the users listed to receive copies.
& R
To: belmont@sun
Subject: Re: Responding to a message
Type a response just as you do when you initally
type a message. End it just like a message,with
a period on a new line.
.
&
```

Printing messages

Perform these two steps to print a message:

1. Save the message to a file.
2. Send this file containing the message to a printer as you would send any file.

Deleting and undeleting messages

Usually when you quit `mail`, read messages are saved in ~/mbox. To prevent this file from growing too large, delete unwanted messages. Deleted messages are not saved in ~/mbox. Messages deleted during the current `mail` session can be retrieved before you leave the session.

The command name for each operation is:

d	Delete
u	Undo (undelete)

Both operations have the same command syntax. When reading Table 7-3, replace the "k" with the command name of the operation you want.

Table 7-3
Delete and undelete commands

Deleting and undeleting messages	
Current	*k*
One	*k message_number*
Range	*k message_number–message_number*
Nonsequential	*k* [*message_number*] [*message_number...*]

Mailing system-wide messages

Occasionally, you may want to send a message to every user connected to your system. `mail` provides a special feature for this purpose.

To send a system-wide message, enter

```
mail msgs
```

where **msgs** defines the distribution of this message as system-wide.

After you enter this command, `Subject:` displays. The rules and commands that apply to sending regular mail also apply to sending system-wide messages.

Figure 7-20 is an example; it announces when system backups will be done.

Figure 7-20
Sending a system-wide message

```
% mail msgs
Subject: policy
Effective Sep 1, every Thursday evening,
starting at 8 p.m., all systems will receive a
full backup.
.
%
```

Reading system-wide messages

When informed that you have a message, you read it by entering

```
msgs
```

This command displays the header of the next message to be read. Figure 7-21 shows the header of the next message waiting to be read.

Figure 7-21
Header of a system-wide message

```
Message 3616:
From belmont Fri Sep 20 11:40:03 1991
Subject: System backups
(4 lines) More? [ynq]
```

The header states:

- Message number
- Sender's name with date and time the message was sent
- Subject of message
- Number of message lines and the message handling options enclosed in brackets. The options are:

`y`	Yes, display this message
`n`	No, bypass this message; show next header
`q`	Quit; bypass this message; exit `msgs` now; start here next session

When more than one system-wide message is waiting to be read, one header displays at a time, giving you the option of either reading the message, skipping to the next message, or quitting.

When the current message is the only one or the last one, the options are [`yq`].

.msgsrc is a file in your home directory that contains the message number of the next message to be read. You can modify this number with a text editor. The next time you run `msgs`, it will read this new value.

Rereading a message

To reread a message, you need its message number, stated on the first line of its header. The command **msgs -h** prints message headers only.

To reread a message, enter

msgs *message_number*

Another method for recalling a message is to specify a number relative to the most recently read message. For example, **msgs -5** displays messages starting with the fifth message prior to the last message read. Hence, **msgs +5** displays messages starting with the fifth message after the last message read.

To learn more about system-wide messages, enter

```
man msgs
```

Talking to other users

ConvexOS provides a utility that lets you carry on a dialogue with another user, much like using the telephone. Appropriately, the name of this utility and its command is `talk`. You establish a two-way connection with another user, during which everything you type transmits to the other user.

When you request `talk`, the person you are calling is informed that you want to talk. When the person answers, a connection is established, and you can type messages to each other, sequentially or simultaneously.

When a connection is established, your screen splits into two sections: one section represents the mouthpiece, the other sectiion is the earpiece of the telephone receiver. You speak into one section and listen in the other.

Establishing a `talk` session

To establish a `talk` session, the command syntax is

`talk` *username*

Figure 7-22 shows the command for requesting a `talk` session with user "belmont."

Figure 7-22
Establishing a talk session

```
% talk belmont
```

After your screen display splits, and while you are waiting for the person you are calling to respond, at the top of your screen this message displays:

```
[Waiting for your party to respond]
```

The recipient of a `talk` call receives a notification similar to the one shown in Figure 7-23.

Figure 7-23
A user is calling

```
Message from Talk_Daemon at 10:56...
talk: connection requested by cambridge
talk: respond with: talk cambridge
```

Accepting a call

If the recipient wants to talk with the caller, the response to enter is stated on the last line of the notice. The response for this example is:

`talk cambridge`

The connection is established, and the electronic conversation can begin.

Refusing `talk` calls

You can request that the system not notify you when someone sends you a `talk` request. Notices sent by `talk` overwrite whatever is on your screen, making it difficult to read what you were doing prior to receiving the message.

The `mesg` command enables you to accept or refuse talk requests with two settings. The syntax of `mesg` is:

```
mesg [y|n]
```

where

`y`	Permits someone to talk to you.
`n`	Prevents someone from talking to you.

To check your current setting, enter

`mesg`

When you try to contact someone who has refused `talk` requests, you receive this message

```
[Your party is refusing messages]
```

Terminating a `talk` session

When you have finished your conversation, press

CTRL-c

Either party can terminate a conversation, not just the caller.

Talking at your terminal

Select someone on your system and establish a talk session. Use the `who` or `users` command to find out who is currently logged in.

The best way to understand how this utility works is to try it. Figure 7-24 shows a sample conversation.

Figure 7-24
Talking with another user

```
Hi there.
I am learning about ConvexOS and thought
I would try the talk utility.

I want to talk with you.



- - - - - - - - - - - - - - - - - - - - -

Hello, this is great; talking to you from
my keyboard...instant response. When you see
the cursor jump between the two screen sections,
don't worry. It moves from me typing to the
other section to receive your input for me to
listen.
The cursor will move between sections every few
characters because it takes turns letting us
type.
```

As the conversation continues, lines scroll off the top of each section as new lines are typed.

Chapter highlights

Check off the topics that you feel you have mastered. Return to those that require more of your time.

- ❒ **Commands that report on other users**
  - -Reports on authorized users or those logged in
- ❒ **Sending mail and messages electronically**
  - -How you control notification of mail
  - -How to enter mail
  - -How to leave mail
- ❒ **Mail commands**
  - -Processing and editing
  - -Reading
  - -Replying
  - -Sending
  - -Saving
  - -Deleting
  - -Printing
- ❒ **Mailing files**
  - -From the shell prompt
  - -From within a message
- ❒ **Mailing system-wide messages**
  - -Notification
  - -Reading and rereading
- ❒ **Talking to other users**
  - -Establishing a session
  - -Terminating a session
  - -Accepting a call
  - -Refusing calls

New to computers? Read this.

A

To understand the capabilities of a computer system, you should know what its major components are and the role of each in processing data. This chapter introduces the major parts of a computer, along with relevant terms and concepts.

The topics explained in this chapter are:

- The major components of a computer system
- The role of each component
- How you access a computer system
- How you tell the system what you want
- Your work environment on a system
- How you store and retreive data
- Definitions of computer-related terminology

These explanations are overviews. If you need or want more information, take a trip to your favorite bookstore and look around in the computer section. You will be amazed at how much is available to you.

If you are new to computers, read this chapter, then pursue your education by reading other books (refer to this primer's bibliography) and talking with experienced computer users. There really is no substitute for experience, so begin reading and start using a computer.

What is a computer system?

A computer system is made up of hardware, software, and data. It processes instructions that direct hardware and software to transform data into information. This section defines hardware and software.

Classifying computers

Several classes of computers are on the market today. With technological advancements bringing new products to market almost daily, these computer classes could look different tomorrow. Tracing the evolution of the personal computer shows how processing power is increasing while physical size is decreasing. The major classes of computers are listed below, starting with the least amount of processing power and progressing to the most powerful:

- Hand-held
- Portable
- Microcomputer systems (personal computers)
- Minicomputer systems
- Medium-scale mainframes
- Large-scale mainframes
- Supercomputers

Common to all computer systems are their "major" components. Systems differ in the level of sophistication and intelligence inherent in each component and in the number of each type of component.

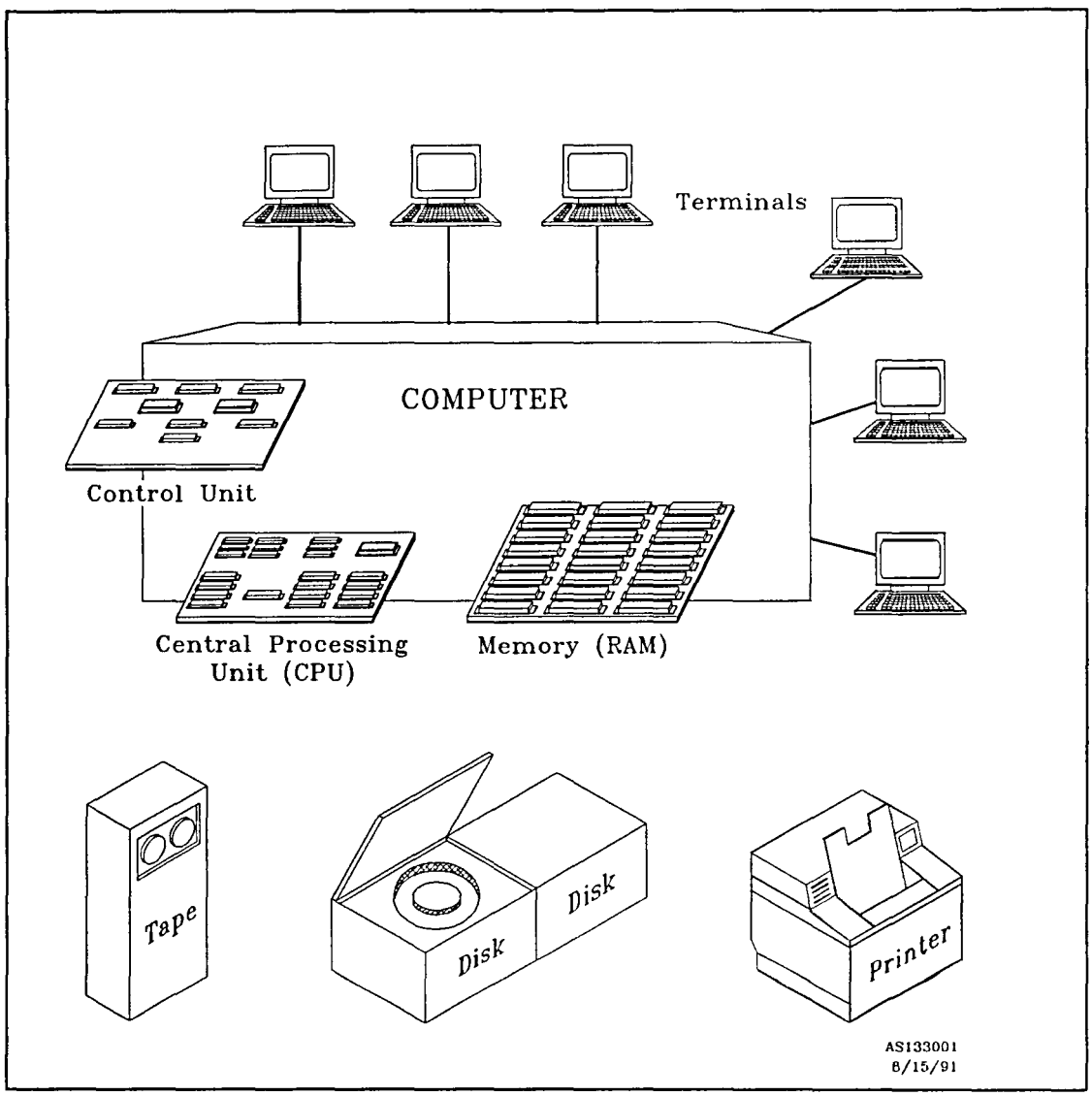
To *configure* a system means to set up (bring together) its components so that, together, they process data in a certain way. The way in which individual components connect to a system and the degree of intelligence possessed by each component define a computer as a member of a certain class. Even though computers come in different sizes and processing power, they all do the same thing: they make it easier and faster to process data into meaningful information.

The sections ahead define the major components common to all computers within all classes. Refer to the glossary at the end of this primer for definitions of unfamiliar terms.

Hardware

Hardware is the physical equipment of a computer system. Figure A-1 presents the major hardware components of a system.

Figure A-1
Computer system components



The CPU

The central processing unit (CPU) is the computer, designed to process and distribute data between the pieces of equipment connected to it. More specifically, its *control unit* interprets and executes program requests. (A *program* is a sequence of instructions that enables a computer to perform a certain task. This term is further discussed in the section "Software.") Its *arithmetic unit* performs all the logical and mathematical computations.

A CPU is a collection of *microchips* (known as chips) that process and store the data entered into the system. Chips are complex electrical conduits that organize large amounts of electronic data between the keyboard, disk drives, monitor, and printers at varying speeds.

Memory

Memory is a collection of microchips. Unlike other chips in the CPU and control units, these do not control or direct the flow of data; they store information until it is needed. The number and storage capacity of memory chips varies from system to system. Whatever the memory capacity of your system, a program does not see memory as individual chips, but as thousands or millions of storage cells, each with a unique address. Storage capacity determines the number of programs that can run at the same time and the amount of data that can be processed at a given time.

Powering up your system electrically applies voltage to the microchips. Information is then loaded into these energized microchips when the CPU runs a software program. Interrupting the flow of electricity corrupts or destroys the data set in the chips. For this reason, memory is temporary storage, technically known as read access memory or *RAM*. Permanent storage is recording information stored in RAM to a mass storage device such as a hard disk or tape.

All program execution and data processing takes place in memory. Program instructions are copied into memory from disk or tape, then moved from memory into a processing area for analysis and execution.

Bus

A bus is a shared path through which the components of a computer are attached in order to communicate. When one component passes data or control information to another, the data or control codes travel along this common path to reach its destination. Every CPU, every microchip, and every memory

address in the computer connects either directly or indirectly to some type of bus. Depending on the computer model, a computer may have multiple processors with multiple types of buses that establish paths between various parts of the system. Bus architecture varies greatly from system class to system class.

Buses carry more than data; they carry power and control information, such as timing signals (from the system clock), interrupt signals, memory addresses, and devices attached to the bus.

Control unit and controller

When physically located within the CPU, the control unit locates, analyzes, and executes each instruction in the program. When physically external to the CPU, either standing as an independent unit (controller) or as part of a peripheral device (device controller), upon signals from the CPU, a control unit performs the physical data transfers between memory and a peripheral device over a bus.

Device controllers transfer data between a device and the CPU. Where disks have much slower transfer rates than a CPU, the controller has the intelligence to handle the reading and writing of data without involving the CPU. Device controllers are becoming increasingly more intelligent and sophisticated.

Terminal

A terminal is an input and output device (peripheral) with a keyboard for typing input and a monitor (video display screen) or printer for displaying output:

- The keyboard is similar to a typewriter keyboard with some differences. Namely, it has special keys—some used alone, some in combination with other keys—that perform functions determined by the software being used. As a rule, what you type on your keyboard displays on your monitor.
- The monitor is the television-like screen on which you view what you are typing or what you request. Monitors vary in shape and size, along with their resolution. Monochrome monitors display information in one foreground color and one background color. These monitors are less expensive than their color counterparts.

Users communicate with a computer system from a computer terminal. Today many types of terminals connect to a system: a personal computer emulating a terminal, a remotely connected (networked) desktop workstation, or a directly connected terminal. In an effort to keep explanations simple and clear, *terminal* in this primer refers collectively to all these types.

Disks and disk drives

A disk drive is a peripheral storage device that holds, spins, reads, and writes magnetic disks. Disk surfaces hold magnetized data that is written and retrieved by the disk drive. Time to locate data varies according to the type of disk drive. Most computers have at least one type of disk drive. Some computers are set up without disk drives; these computers are diskless workstations that are commonly used with networks.

Disk storage comes in several forms: flexible floppy diskettes, rigid disk cartridges, and hard disks. A hard-disk drive can store larger amounts of data than a floppy diskette drive; it can also more rapidly transfer data. Functionally, all drives do the same job: they serve as a place to store information when the computer is not working on it, which is why they are sometimes referred to as auxiliary storage devices.

Tapes and tape drives

Tapes sequentially store and read magnetic data. Locating a specific piece of data on tape means reading all data in front of it. To add or delete data from tape, the contents of the tape have to be brought into the system. Typically, tape is used for data collection and offline backup storage; they are also known as auxiliary storage devices.

Printers

Many types and models of printers are on the market today. The major differences between printers are the amount of data output per minute and the quality of the copy produced. Fast printers, such as band or line printers, handle large amounts of data (lines-per-minute output), such as program listings and print them in draft quality. Laser printers handle text and graphics (characters-per-minute output), such as letters, manuals, and brochures, and print them in typeset quality. (Laser printer graphics capabilities enable printers to produce diagrams and illustrations in the same quality as text.)

Software

By itself, computer hardware can do little. *Software*, a set of instructions called programs, must be added to make the hardware carry out specific operations. To understand the difference between hardware and software, imagine the covers and pages of this manual as the hardware, while the language written on these pages is the software.

Software includes two types of programs: *system programs* and *application programs*.

System programs, usually supplied by a computer vendor, control the internal operations of a computer system. These are the programs that let you log onto a system, type commands, and write programs. Included in this group is the *operating system*, the interface between users and the hardware, and the software that enables the hardware and application programs to work together.

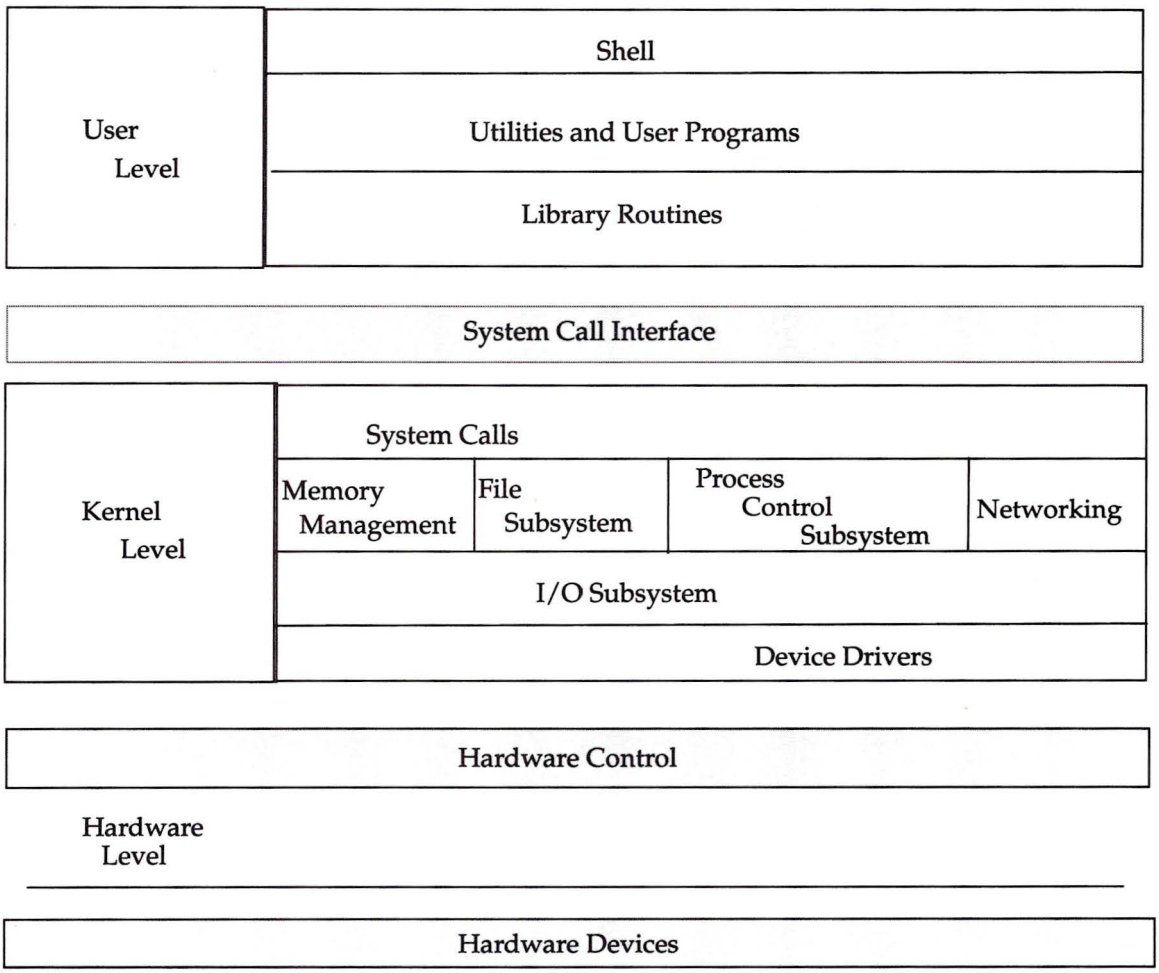
Application programs are designed to perform specific tasks. Some examples are: word processors, spreadsheets, graphics packages, and accounting programs.

Data is the information used by or produced by system or application software and stored on disks or tapes.

The operating system

The operating system is the coordinator between the computer hardware and the users. It manages the resources that enable users to store and organize files; to communicate, print, display information, and to write and run programs. On some systems, it allows many users and programs to share one system's resources. Figure A-2 shows the layers and types of software that are a part of the ConvexOS operating system.

Figure A-2
Software layers of the ConvexOS operating system



The kernel level is the part of the operating system that allocates the resources and controls program execution. Resources include peripheral devices, main memory, and system software.

Distinct types of operating systems exist. Some are dedicated to serving specific types of applications, some are proprietary, and others are like ConvexOS: general purpose, multitasking, and multiuser. ConvexOS is based on the Berkeley Software Distribution (BSD).

In summary: directing data for processing

Users send instructions and data to a computer from peripheral devices—such as terminals, disk or tape drives—for processing. The CPU interprets the instructions (commands) and calls the appropriate software and hardware to execute the request. Data transfers over buses between peripheral devices and main (RAM) memory governed by the control unit and device controllers.

Peripheral devices connect to the network and computer system through device controllers. These controllers are becoming increasingly more intelligent in an effort to relieve the internal control unit for other tasks.

The CPU uses RAM to store programs while they are being executed and data while it is being processed. Auxiliary storage offers greater capacity than main memory, even though it uses more CPU resources to access it. Current and frequently used data are stored on disk. Infrequently needed data or data to be preserved (archived) goes to tape. Hard copy output goes to a printer.

Measuring computer memory and disk capacity

Measurement is done in bytes. A byte is a group of binary digits (zeros or ones) used to encode a single character. A typical memory chip has 32,768 bytes, known as 32K.

Most computer systems use the ASCII code to translate byte values to the familiar letters, numbers, and symbols that we use. ASCII stands for American Standard Code for Information Interchange.

Connecting systems

Today computer systems talk with other computer systems. Systems can and do work as independent units, but the practice of sharing hardware devices, software programs and data is the goal of more and more businesses. The term *communications* is the electronic transfer of information from one location to another. *Networks* enable multiple computer systems to communicate, whether the physical units are in the next room or across the globe. Peripheral devices attached to a network are available to the users of the network.

Communications and networking are expansive fields, with volumes of information written on them. However, the focus of this overview is the working of a single system.

Working with your CONVEX system

Users communicate with a computer system from a computer terminal. Today, there are many types of terminals that can connect to a system: a personal computer emulating a terminal, a remotely connected (networked) workstation, or a directly connected terminal. To keep explanations clear, terminal in this primer refers collectively to all types of keyboard devices.

Commands are how you tell the system what you want it to do. For every activity supported by software, there is an associated command. As you type a command, it displays on the terminal's screen so you can see what you are typing.

A special program known as a shell accepts your commands and executes them. Shells are explained next.

Verification of your user name and password by system software confirms you as an authorized user. As a result, you are given access to the system and placed in your *home directory.* At this point, a program called the *shell* waits for you to enter *commands* (instructions) to the system. The next section explains how a shell serves users.

What is a shell?

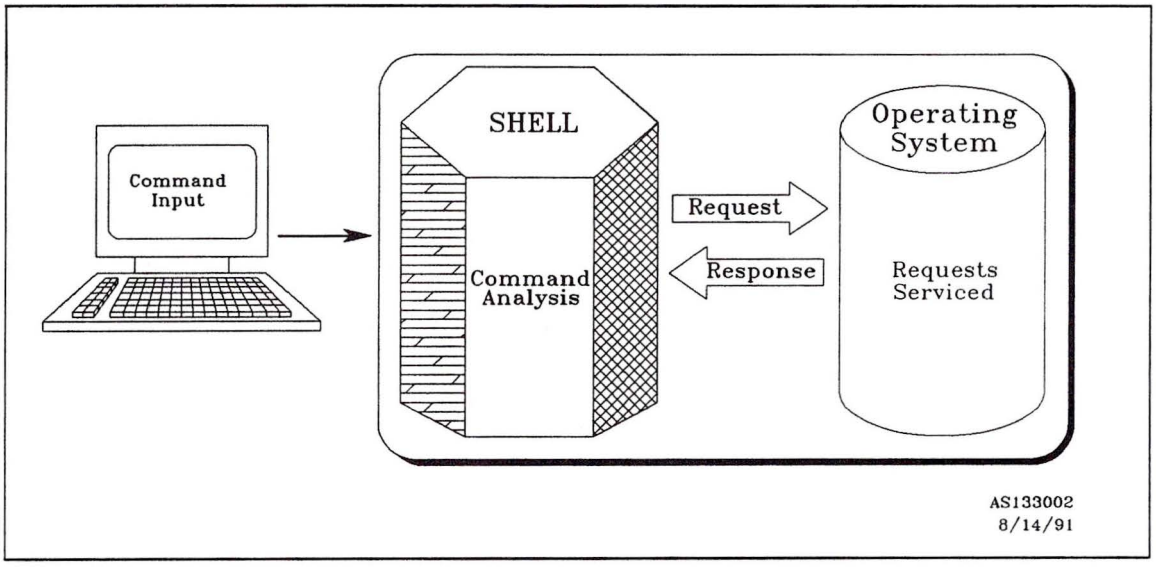
When you log on, ConvexOS starts a shell for you, and terminates it when you log out. The term shell originates from the seashell that provides an environment for a sea animal; likewise, the shell sets up a work environment from which you use your CONVEX system.

A shell is an interpretive programming language: a program that interprets the commands that you enter. As a command interpreter, the shell interactively accepts commands and arranges for the execution of requested actions. Commands tell the system what you want it to do. For every activity supported by software, there is an associated command. As you type a command, it displays on your terminal's screen so you can see what you are typing.

You can string a series of shell commands together to accomplish some task. This type of programming is called writing *shell scripts.*

Figure A-3 shows the route taken by each command that you enter.

Figure A-3
Command interpretation



Types of shells available

More than one type of shell exists, each having its own complexion. It is up to you to decide which shell best meets your needs. For ConvexOS, there are four basic shells:

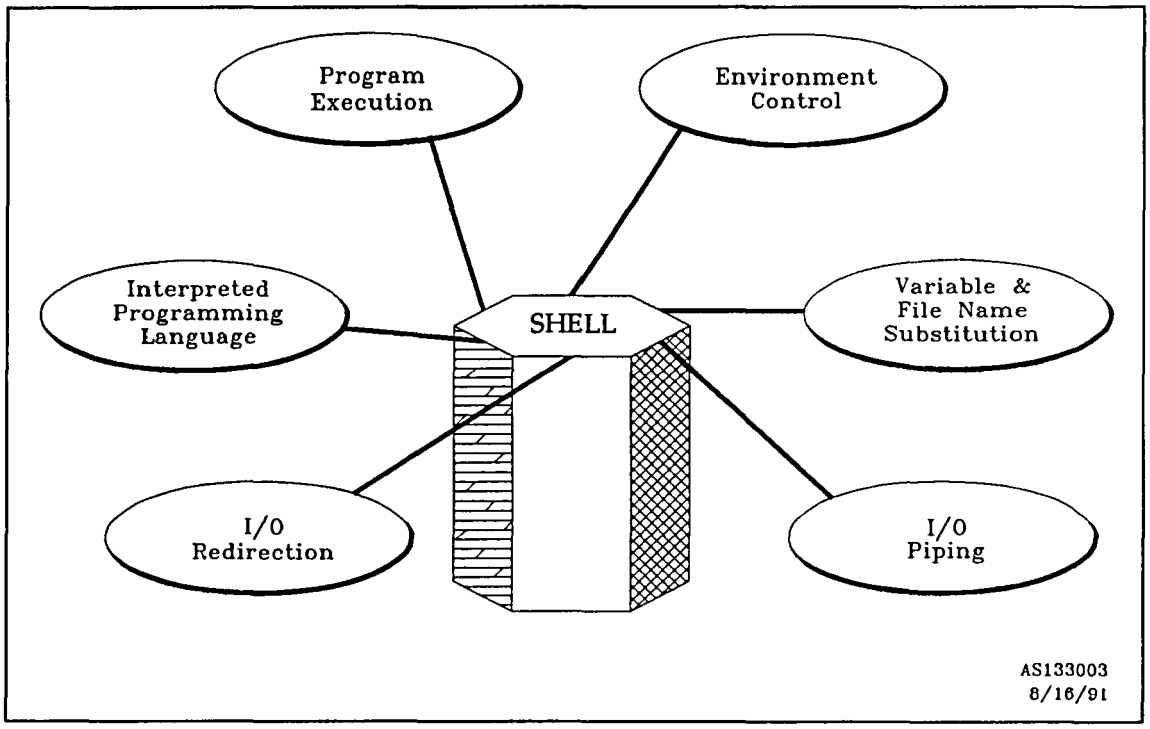
- **Bourne shell**—An easy-to-use, flexible command interpreter with a built-in programming language.
- **C shell**—A command interpreter and C-like programming language that logs previously issued commands and supports running interactive jobs in foreground and background.
- **Korn shell**—A superset of the Bourne shell with some additional features of the C shell.
- **COVUEshell**—A CONVEX-to-VAX user environment. An optional product.

Usually, you can identify a shell by its command prompt—the character, or a string of characters, that signal that the system is ready to accept input, namely commands.

Focusing on the C shell

The most widely used shell is the C shell. What it does for you is very powerful. Figure A-4 presents its responsibilities, followed by a brief description of each.

Figure A-4
Responsibilities of the C shell



- **Program execution** —The shell executes programs that you request. Each time you enter a command line, the shell analyzes the line and determines what to do. Commands are discussed later in this chapter.
- **Variable and file name substitution**—When the shell analyzes a command line, it recognizes special characters and substitutes the appropriate values.
- **I/O redirection**—The shell analyzes the command line for special characters, including characters for redirecting input, output and error messages. Discussions on standard input and output are in Chapter 3.
- **I/O piping**—As the shell scans the command line for redirection characters, it also looks for the pipe character (|). For each pipe character, the shell connects the standard output from the command preceding the pipe to the

standard output of the command following it, then executes the command line.

- **Environment control**—The shell provides commands that enable you to customize your work environment. These commands are explained in Chapter 2.
- **Interpreted programming language**—The shell has its own built-in programming language. (It is beyond this primer's scope to explain this language.)

Entering instructions to the system

A command is an abbreviation that invokes a specific program. Programs exist for hundreds of operations, each called by its command. Commands have three parts that must be typed in this order:

command_name [*option*] [*argument*]

where

command_name	Requests a specific operation.
option	Modifies how a command works.
argument	Defines the objects on which the command works, usually file and directory names.

The line on which you type a command is the line where the shell prompt sits. This line is referred to as the command line.

ConvexOS commands are usually lowercase, simple two, three, sometimes four characters long. At first they may seem too cryptic, but, with use, you will learn their convenience.

Working with files, directories, and data

A *file* is a block of information. Files can hold business reports, letters, budgets, and computer programs. The file system oversees retrieving files into memory for use, storing them to a disk or tape, or printing them, with every operation performed according to your instructions—your commands.

You create files either with an editor, such as `vi` or `emacs`, or with an application program such as a word processor. You can also write a computer program to create files, let's say in Pascal or C language for example. Some files contain *ASCII* text readable by people; other files contain *binary* code (called text files) represented by zeros (0) and ones (1) that are read by programs.

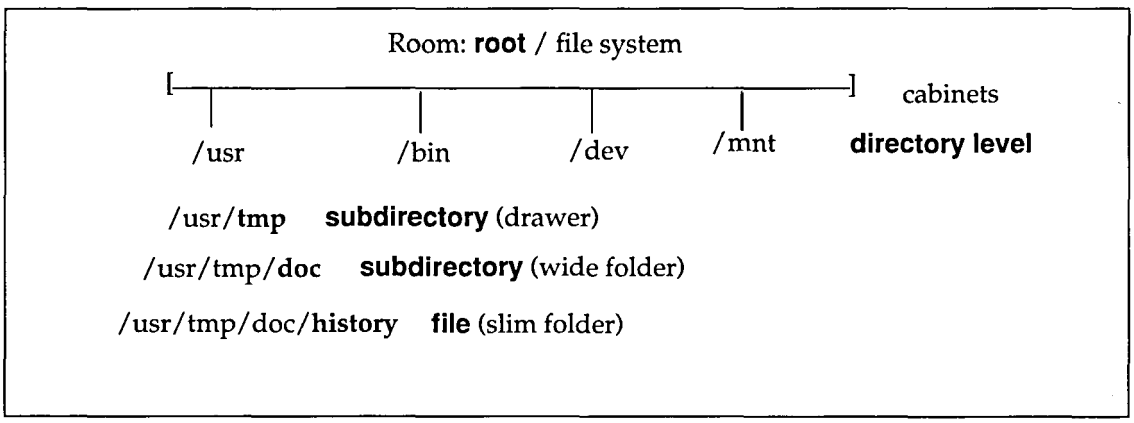
Storing files and data within directories

After you create a file, if you store it on disk, you write it to a *directory.* Directories contain the names of files and other directories. A directory that follows directly below another directory (hangs from that directory) is known as a *subdirectory.* Subdirectories can also contain files and another level of subdirectories. Each file and directory has its own unique name for identification.

Visualizing directories and files

To better understand how files and directories work together, refer to Figure A-5. The explanation that follows has you visualizing a room containing several file cabinets, all with drawers full of labeled file folders.

Figure A-5
Layout of directories, subdirectories, and files



View the room as the main (root) file system, identified by the "slash." Each file cabinet in the room is a directory, and each drawer as a subdirectory. Each drawer (subdirectory) may or may not contain file folders of varying width, where the wide folders are another level of subdirectories and the slim folders are files. Looking in the wide subdirectory folders, you may see some slim folders; yes, these are files. Some of the slim folders contain program files, while others contain document files.

The room containing all the file cabinets is called *slash,* identified by the (/) character. To uniquely identify each cabinet, each one is assigned a unique name, preceded with the slash (/) character. To associate a file drawer (subdirectory) with its cabinet (directory), each drawer name is preceded by its cabinet name. Within each drawer, the name of each folder begins with the name of the room, cabinet, and drawer where it resides. Wide folders contain slim folders, so a slim folder name will also

contain the wide folder name. If you drew these levels out, you would see that the file system is a tree structure. The path from the slash level down to the file you want is called the file's path name. Chapter 5 of this primer explains the ConvexOS file structure.

In Chapter 5 you learn why smartly organizing your directories benefits you and your CONVEX system. At the beginning of this chapter you learned about computer resources. Two major resources in a computer system are disk space (storage) and the CPU time involved with retrieving files into memory for use. A well-planned directory and file scheme saves system search time across or down your directories when you request a specific directory, as well as saving the system resources needed to store data in specific files.

Time to start the primer

With the basics you just learned, you are ready for Chapter 1. Make sure that you understand the terms and topics explained in this appendix. Consult the glossary in the back of this primer or a third-party guide that explains computer concepts.

Chapter 1 awaits you.

Reporting problems

B

The CONVEX Technical Assistance Center (TAC) is staffed by technical specialists who address diverse questions and problems arising in a supercomputing environment. The TAC recommends using the `contact` utility to inquire about hardware, software, or documentation.

`contact` is an interactive program that helps the TAC track reports and route them to the CONVEX personnel most qualified to handle them.

This appendix describes:

- Prerequisites for using `contact`
- Procedure, step-by-step, for using `contact`
- Tips for using `contact`

Prerequisites for using `contact`

`contact` requires:

- UNIX-to-UNIX Communication Protocol (UUCP) connection to the TAC
- Full path name of the program or utility about which you have an inquiry
- Version number of the program or utility about which you have an inquiry

UUCP connection

Before using `contact`, ask your system manager if your site has a UUCP connection to the TAC. A UUCP connection allows files to be copied from one UNIX-based system to another. The `uucp` (UNIX-to-UNIX copy) command relies on either a dial-up or hard-wired UUCP communication line.

Using `which` to find a program's path name

To determine the full path name of a program or utility, use the `which` command, which has the following syntax:

`which` *program*

where *program* is the name of the program whose path you need. Figure B-1 illustrates the use of `which` to find the full path name of the program called `filefix`.

Figure B-1
Using the `which` command

```
% which filefix
/bin/filefix
% ▮
```

In this example, the full path name of `filefix` is `/bin/filefix`.

If you use `csh` or `ksh`, you can use the `whence` command to find a program's path name. `whence` works the same as `which`, but faster.

For more information on the `which` or `whence` command, refer to the which(1) or whence(1) man page, respectively.

Using `vers` to find a program's version number

To determine the version number of a program or utility, use the `vers` command, which has the following syntax:

`vers` *path*

where *path* is the full path name of the program or utility whose version number you need. Figure B-2 illustrates the use of `vers` to find the version number of the program `filefix`.

Figure B-2
Using the `vers` command

```
% vers /bin/filefix
/bin/filefix: 9.0
% ▮
```

In this example, the version number of `filefix` is shown to be 9.0.

For more information on the `vers` command, refer to the vers(1) man page.

Using contact

Before using contact, refer to the previous section, "Prerequisites for using contact," which gives you important information you will need to know in order to use contact effectively.

Invoking contact

The contact utility has the following syntax:

contact *option*

where *option* can only be -r

which includes the ~/contact.dead file of your previous report with your current contact report. For more information on this option, refer to "Submitting you dead.report file" on page B-14.

contact can be invoked as shown in Figure B-3:

Figure B-3
Beginning a contact report

```
% contact
Welcome to contact version 0.12 (90/02/05)
```

After invoking contact, the system responds with a welcome message and a series of questions regarding your hardware, software, or documentation question. To use contact effectively, you must provide the following information:

1. Your name, title, phone number, and corporate name.
2. Name of the product with which you are having a problem.
3. Version number of the product with which you are having a problem.
4. One-line summary of the problem.
5. Detailed description of the problem.
6. Priority of the problem.
7. Instructions on how to reproduce the problem.
8. Comments about the problem.
9. Comments about the documentation relating to the problem.
10. Whether files are included in the contact report and, if so, the full path names of these files.
11. How you would like to complete your contact session.

Providing `contact` information

So that the TAC can help you to the best of their ability, you must complete each of the following inquiries as best as you can. The following sections are step-by-step examples on how to respond to each inquiry prompted by `contact`, as shown in Figure B-4.

Figure B-4
Beginning a `contact` report without a .contact file

```
% contact
Welcome to contact version 0.12 (90/02/05)

Enter your name, title, phone number, and corporate name (^D to
terminate)
> Chris Smith
> Programmer
> (214) 900-2000
> Jupiter Corporation
> ^D
```

Step 1 Entering information about yourself

After invoking `contact` you are asked for some information about yourself as shown in Figure B-4, unless you have a `.contact` file in your home directory. If you have a `.contact` file in your home directory, `contact` skips this inquiry. (Refer to "Creating a contact file" on page B-12.)

Figure B-5 illustrates how to invoke `contact` and how the system responds if you have a `.contact` file in your home directory. If you have a .contact file, go to Step 2.

Figure B-5
Beginning a `contact` report with a `.contact` file

```
% contact
Welcome to contact version 0.12 (90/02/05)

Enter the name of the product involved
> filefix
```

Step 2: Entering the product name

The `contact` utility next prompts for the name of the product with which you are experiencing a problem, as shown in Figure B-5. Enter the name of the product.

Step 3: Specifying a program version number

The `contact` utility prompts for the version number of the product with which you are experiencing a problem as in Figure B-6.

Figure B-6
Prompt for product version

```
Enter the version number (in the form X.X or X.X.X.X) of the product
> 9.0
```

If you do not know the version number, press **CTRL-Z** to suspend the session and refer to the section "Using `vers` to find a program's version number" in this appendix. After you have determined the proper version number, use the `fg` command to return to the `contact` session and enter the version number in the form *X.X* or *X.X.X.X*, such as

`9.0`

or

`9.0.0.1`

Step 4: Entering a one-line problem summary

The `contact` utility prompts for a one-line summary of the problem, as shown in Figure B-7.

Figure B-7
Prompt for a short, problem summary

```
Enter a short (1 line) summary of the problem
> Trouble suspending filefix
```

This summary is the subject header in any further correspondence regarding your problem. Please make this summary as descriptive as possible in one line.

Step 5: Entering a detailed problem description

The `contact` utility prompts for a detailed description of the problem, as shown in Figure B-8.

Figure B-8
Prompt for a detailed description of the problem

```
Enter a detailed description of the problem (^D to terminate)
> After entering filefix, I have trouble suspending the session if I
want to do other tasks.
> ^D
```

Enter a detailed description of the problem. When you have completed your description, press **CTRL-d**.

When writing your problem description, please make it as complete as possible. Include source code and a stack backtrace when possible. (Refer to the adb(1) or csd(1) man pages for information on obtaining a stack backtrace.) The more information you provide, the quicker the TAC can isolate and solve your problem.

Step 6: Entering a problem priority

`contact` prompts for the priority of your problem. The priority of a problem indicates the impact your problem has on your work. Figure B-9 shows how `contact` prompts for priority levels. Select your problem priority by entering the number associated with the priority.

Note

You must enter the number associated with your problem priority.

Figure B-9
Prompt for problem priority

```
Enter a problem priority, based on the following:
1) Critical-work cannot proceed until the problem is resolved.
2) Serious-work can proceed around the problem, with difficulty.
3) Necessary-problem has to be fixed.
4) Annoying-problem is bothersome.
5) Enhancement-requested enhancement.
6) Informative-for informational purposes only.
> 4
```

Step 7: Giving instructions on how to reproduce the problem

`contact` prompts for instructions on how to reproduce the problem, as shown in Figure B-10.

Figure B-10
Prompt for how to reproduce problem

```
Enter instructions by which the problem may be reproduced
  (^D to terminate)
> 1. Enter filefix myfile.c
> 2. Suspend by entering CTRL-s
```

Please include the command syntax and options you used and anything else you did to make the program run.

Step 8: Giving applicable comments

The `contact` utility prompts for any other pertinent comments, as shown in Figure B-11. Please include all relevant information.

Figure B-11
Prompt for supportive comments

```
Enter any comments that are applicable(^D to terminate)
> Perhaps I am using the wrong command?
> ^D
```

Step 9: Offering suggestions for documentation and support

The `contact` utility prompts for suggestions regarding documentation supporting the product, as shown in Figure B-12:

Figure B-12
Prompt for suggestions or comments

```
Do you have any suggestions or comments on the documentation that
 you referenced when you were trying to resolve your problem
(for example, additions corrections, organization, accessibility)?
 (^D to terminate)
> A command summary or quick-reference for filefix would be helpful.
```

Please indicate whether the documentation could be revised to address the problem.

Step 10: Indicating additional files

The `contact` utility prompts for names of files necessary to reproduce the problem.

If you have files that can be included with your report, enter "yes" and enter the related file names, as shown in Figure B-13.

Note

List file names one per line. After entering your last file name, press CTRL-d. Tilde-escape sequences are not recognized in your file listing. A tilde (~) in this section indicates your home directory.

Figure B-13
Including related files

```
Are there any files that should be included in this report (yes | no)?
> yes
Please enter the names of the files, one to a line (^D to terminate)
> myfile.c
> ^D
```

If you do not have any files to include with your report, enter "no," and you will be prompted for the next response.

If files specified are small text files, they are automatically included in the `contact` report. If the files are too large to be included in this report, `contact` gives further instructions on how to submit these files.

To specify a directory, combine directory files into a single file using the `tar` command (Refer to the tar(1) man page for further information) or enter each file name in the directory on a single line in the `contact` report.

Step 11: Finishing your report

To finish your `contact` report, you are given the choice to review, edit, submit, or abort the report, as shown in Figure B-14.

Note

You must enter a number associated with your selection.

Figure B-14
Prompt to review, edit, submit, or abort your `contact` report

```
Please select one of the following options:
1) Review the problem report.
2) Edit the problem report.
3) Submit the problem report.
4) Abort the problem report.
> 3
```

The options listed in Figure B-14 indicate the following:

Review	Review the text of the `contact` report. You are then prompted again to select an option.
Edit	Edit the text of the `contact` report. If you choose to edit the report, `contact` opens your default text editor.
Submit	Send the report to the CONVEX TAC. The TAC notifies you within 48 hours that your report has been received. Choosing this option exits the `contact` utility and returns you to your shell.
Abort	Save the text of the report in a file named ~/dead.report (in your home directory). Choosing this option exits `contact` and returns you to your shell.

Tips for using `contact`

This section lists tips to help you use `contact` efficiently. In particular, this explains:

- Creating a .contact file
- Suspending a `contact` session
- Moving within `contact` from one prompt to another
- Using tilde-escape sequences within `contact`
- Aborting your `contact` report
- Submitting your aborted report

Creating a .contact file

When you invoke `contact`, it first prompts for your name, title, phone number, and company name. You can, however, create a `.contact` file to skip this first prompt. `contact` will look for a `.contact` file and use the information in that file automatically.

Follow these steps to create a .contact file:

1. Create a .contact file in your home directory.
2. Enter your name, job title, phone number, and company name, each on a new line.

Figure B-15 is an example of a .contact file viewed using the `cat` command.

Figure B-15
Example of a .contact file

```
% cat .contact
Chris Smith
Programmer
(214) 900-2000
Jupiter Corporation
% ▮
```

Suspending your contact session

Sometimes it is necessary to suspend your contact session and return to your shell (for instance, to find your program path name or version number). To suspend your contact session, press **CTRL-z**.

To return to the contact session, type **fg**. Using **CTRL-z** and the fg (foreground) command, you can switch between contact and your shell. You cannot, however, use **CTRL-z** and fg to switch back and forth if you are using the Bourne shell (sh).

Moving to another prompt

The contact utility prompts for information pertinent to your hardware, software, or documentation question. Some prompts require one-line responses; to move to the next prompt, press **RETURN**. Other prompts require more than a one-line response; to move to the next prompt, press **CTRL-d**.

Tilde-escape sequences

The contact utility treats input beginning with a tilde (~) as a special sequence. The character following the tilde is considered a request for a special function.

Note

You cannot use tilde-escape sequences when listing file names to include in your report, as described in "Step 10: Indicating additional files."

Any other time you can use the following tilde sequences within contact:

~e	Edit your report using your default editor (defined in your EDITOR environment variable).
~h	Help by displaying a list of available tilde-escape sequences.
~p	Print your contact report to the terminal screen.
~r *filename*	Read the contents of a specified file into a response to the current prompt, where *filename* is the name of the file you wish to specify.

Some prompts require only a one-line response. This tilde-escape sequence works only for prompts that allow more than a one-line response.

~~ Insert a single tilde as the first character in the line. This is in case you need to use a tilde as part of your response and not as an escape sequence.

Aborting your report

To abort a `contact` report, either press the interrupt key (usually **CTRL-c**) or choose the `abort` option when prompted by the `contact` utility as described in "Step 11: Finishing your report." Using **CTRL-c** to abort does not save the contents of the report. Using the `abort` option saves the contents of the report in a file named ~/dead.report.

Submitting your dead.report file

After you abort a `contact` report (as shown in "Step 11: Finishing your report"), `contact` saves the report in a file named ~/dead.report. If you specify the `-r` option when you next invoke `contact`, it automatically merges the contents of your ~/dead.report file from your previous report into your current `contact` report. For more information on how to use the `-r` option refer to the section "Invoking `contact`."

Bibliography

This annotated bibliography focuses on third-party documentation that provides technical information useful to users of CONVEX systems and software. To facilitate subject look up, entries are organized into the following topic groups:

- Introductory material
- Text editing and document preparation
- Communications
- Support tools
- General UNIX
- Miscellaneous material

Introductory material

Bourne, Philip E. *UNIX for VMS Users*. Digital Press, 1989.
Presentation of fundamental concepts and basic command procedures. Covers the use of high-level languages, programming the operating system, text processing, and networked communications. Appendixes cover command and file summaries.

Lamb, Linda. *Learning the vi Editor*. O'Reilly and Associates, 1988.
A complete guide to editing with `vi`. Covers basic editing, moving around quickly, beyond the basics, greater power with `ex`, global search and replacement, customizing `vi` and `ex`, command shortcuts, and quick reference to `vi` and `ex` commands.

Loukides, Mike. *UNIX for FORTRAN Programmers*. O'Reilly and Associates, 1989.
An introduction to UNIX and these UNIX tools: the FORTRAN compiler (f77); UNIX interactive command languages; `vi`; object library management tools (`ar` and `ranlib`); `adb` and `dbx`; `prof`, `gprof`, and time-profiling tools; `make`; and `rcs`, a source-code management system for large projects.

The Waite Group. *UNIX Primer Plus*. 2nd ed. Howard W. Sams and Company, 1990.
Introduction to basic and advanced features. Topics include logging in and out, using shell functions and entering commands, sending and handling electronic mail, managing files and the directory structure, using editors, using programming languages, printing, and some advanced editing techniques such as abbreviations, command macros, plus advanced search and replace options.

Todino, G.; Strang, J. *Learning the UNIX Operating System*. O'Reilly and Associates, 1988.
Introduction for new UNIX users. Topics include logging in and out, managing UNIX files and directories, sending and receiving mail, redirecting input/output, pipes and filters, background processing, and customizing your account.

Text editing and document preparation

Dougherty, Dale; O'Reilly, Tim. *UNIX Text Processing*. O'Reilly and Associates, 1988.
Covers basic editing to writing complex `troff` macro packages. Topics include editing with `vi` and `ex`, using basic `nroff` and `troff` requests, using ms and mm macros, formatting with `tbl`, typesetting equations with `eqn`, drawing pictures with `pic`, shell programming for writers, `awk` programming language, `sed` stream editing, writing `troff` macros, indexing with `troff`, `sed`, and `awk`, and managing book production with `make`.

Gehani, Narain. *Document Formatting and Typesetting on the UNIX System*. Silicon Press, 1986.
Contains a general discussion of formatting and an extensive discussion of the UNIX system document formatting tools. Covers mm macros, `tbl`, `pic`, `eqn`, and `troff`. Contains templates for preparing a variety of documents and descriptions of the UNIX system typesetting commands.

Communications

Comer, Douglas. *Internetworking with TCP/IP*, Second Edition. Prentice Hall, 1991.
Discusses protocols, addressing, address resolution (ARP), routing, and gateways. Handy for anyone who needs to manage a TCP/IP network, design network applications, or write network programs.

Frey, Donnalyn.; Adams, Rick. *A. Directory of Electronic Mail Addressing & Networks*. O'Reilly and Associates, 1990.
Lists more than 130 networks around the world and, for each one, shows general description, address structure and format, architecture, connections to other networks or sites, facilities available to users, contact name and address, cross-references to other networks, future plans, and the date of last update.

Kochan, Stephen; Wood, Patrick. *UNIX Networking*. Howard W. Sams and Company, 1989.
Comprehensive look at the major aspects of networking in a UNIX system.

Tanenbaum, Andrew. *Computer Networks*, Second Edition. Prentice Hall, 1989.
Contains general networking information, including network architecture, protocols, and applications. Covers TCP/IP, OSI. X.25, NFS, LANs, WANs, FDDI, and Ethernet.

O'Reilly, Tim; Todino, Grace. *Managing UUCP and Usenet*. O'Reilly and Associates, 1989.
Written for system administrators, this book explains how to install and manage UUCP and Usenet software. Covers how UUCP works, RS-32 cabling, talking with modems, setting up a UUCP link, security considerations, UUCP administrations, introduction to Usenet, installing Netnews, administering Netnews, and working files.

Todino, Grace; Dougherty, Dale. *Using UUCP and Usenet*. O'Reilly and Associates, 1988.
Shows how to use `cu` and `tip` to communicate with both UNIX and non-UNIX systems. For beginners and experienced users. Covers introduction to UUCP communications, copying files with UUCP, remote command execution, sending mail to distant users, checking on UUCP requests, remote `login` with `cu` or `tip`, extending the UUCP network, and reading and posting news.

The Waite Group. *UNIX Communications*. Howard W. Sams and Company, 1989.
For novice and expert UNIX users, covers mail, networking, and file transfer tools.

General UNIX

Christian, Kaare. *The C and UNIX Dictionary.* John Wiley and Sons, 1988.

Contains basic computer and computer hardware terminology, terms relating to system maintainence, technical terms relating to programming, and profiles of individuals who have significantly contributed to the development of the UNIX system.

Computer Systems Research Group, Computer Science Division, Department of Electrical Engineering and Computer Science, University of California. *UNIX Programmer's Manual Supplementary Documents 1.* 1986.

One of the sources for the *ConvexOS Tutorial Papers.* It is programmer-oriented and contains papers on languages and programming tools.

Foxley, Eric. *UNIX for Super-Users.* Addison-Wesley, 1988.

Describes powering up and powering down the system, creating new `login` names, maintaining file security, monitoring user resource usage, and machine performance considerations.

Kernighan, Brian; Pike, Robert. *The UNIX Programming Environment.* Prentice Hall, 1984.

Covers file systems, using the C shell, filters, shell programming, programming with standard I/O, UNIX system calls, program development, and document preparation.

Loukides, Mike. *System Performance Tuning.* O'Reilly and Associates, 1990.

Topics covered include real and perceived performance problems, tricks to improve keyboard response, how to manage your system's load, how to survive without a lot of memory, how to configure your I/O system for the best throughput, how to detect an overworked or malfunctioning network, and how to build a more efficient kernel.

Nemeth, Evi; Snyder, Garth; Seebass, Scott. *UNIX System Administration Handbook.* Prentice Hall, 1989.

Covers typical system administration duties, booting and shutting down, superuser privileges, file systems, controlling processes, adding new users, devices and drivers, configuring the kernel, installing terminals, printing under ATT, printing under BSD, adding a disk, hardware maintenance tips, networking, `mail` and `sendmail`, `uucp`, news, backups and transportable media, accounting, daemons, and periodic processes.

Quarterman, John S., et al. *The Design and Implementation of the 4.3 BSD UNIX Operating System.* Addison-Wesley, 1988.
Covers the internal structure of the 4.3 BSD system and the concepts, data structures, and algorithms used in implementing the system facilities. Kernel discussion includes system process management, memory management, the I/O system, the file system, the socket IPC mechanism, and network-protocol implementations.

Sage, Russell G. *Tricks of the UNIX Masters.* Howard W. Sams and Company, 1987.
Covers how UNIX works, task management, security and personal security, UNIX communications, useful tricks (powerful one- and two-line commands)

The Waite Group. *UNIX Papers.* Prentice Hall, 1990.
Collection of papers on a broad range of advanced topics such as security, communications, and standards for the operating system.

Support tools

Aho, Alfred; Kernighan, Brian; Wienberger, P. *The awk Programming Language.* Addison-Wesley, 1988.
This book documents `nawk`, the "new" version of `awk`. Users need `nawk` — part of the optional CONVEX Toolbox product — to use this book effectively.

Anderson, Gail; Anderson, Paul. *The UNIX C Shell Field Guide.* Prentice Hall, 1986.
Explains how UNIX works. Focuses on file management and personal security. Covers UNIX communications, reveals powerful one- and two-line commands, and discusses shell programming and debugging.

Curry, Dave. *Using C on the UNIX System.* O'Reilly and Associates, 1988.
Written for intermediate to experienced C programmers who want to be UNIX programmers, this book covers I/O using stdio, manipulating files and directories, device I/O control, getting information about users, telling time and timing functions, processing signals, creating processes and executing programs, job control, interprocess communication, and networking (Internet clients, servers).

Darwin, Ian F. *Checking C Programs with lint.* O'Reilly and Associates, 1989.
Contents include an overview of using `lint`, casting and delinting, `lint` comments, command line options, using `lint` with `make`, rolling your own `lint` library, public domain programs, a look inside `lint`, and future directions.

Dougherty, Dale. *sed & awk*. O'Reilly and Associates, 1990.
Covers the original and the new `awk`. Includes how to write a `sed` script, using both basic and advanced commands, how to use `awk`'s built-in functions as well as how to write user-defined functions that are available with `nawk`. Also includes miscellaneous `sed` and `awk` scripts that demonstrate a wide range of scripting styles and techniques.

Gircy, Gintaras R. *Understanding and Using COFF*. O'Reilly and Associates, 1987.
Contents cover the basics of COFF, assembly code relocation process, COFF file headers, relocation structures, the linking process, the COFF system in UNIX, magic numbers, the COFF symbolic debugger, and COFF and shared libraries.

Loukides, Mike. *UNIX for FORTRAN Programmers*. O'Reilly and Associates, 1989.
An introduction to UNIX and these UNIX tools: the FORTRAN compiler (f77); UNIX interactive command languages; `vi`; object library management tools (`ar` and `ranlib`); `adb` and `dbx`; `prof`, `gprof`, and time profiling tools, `make`; and `rcs`, a source code management system for large projects.

Mason, Tony; Brown, Doug. *lex and yacc*. O'Reilly and Associates, 1989.
Part one provides a basic understanding of language theory, the function of compilers and interpreters, `lex` and `yacc`, and presents tutorial examples. The second part provides detailed information for advanced use of `lex` and `yacc`. Appendixes cover GNU, Flex, and Bison. This book assumes some knowledge of UNIX and C.

Strang, John. *Programming with curses*. O'Reilly and Associates, 1989.
Curses is a UNIX library of functions for controlling a terminal's screen display. This handbook covers Ken Arnold's original Berkeley implementation of curses, not the System V version. Topics include windows, screens, and images; multiple windows, the WINDOW structure, terminal independence, the curses functions, and sample.

Strang, John; Mui, Linda; O'Reilly, Tim. *termcap & terminfo*. O'Reilly and Associates, 1988.
This book documents hundreds of capabilities of `termcap` and `terminfo`. It also covers terminal independence, reading `termcap` and `terminfo` entries, capability syntax, initializing the terminal environment, writing `termcap` and `terminfo` entries, converting between `termcap` and `terminfo`, and detailed descriptions of each capability and how it is used.

Talbott, Steve. *Managing Projects with make.* O'Reilly and Associates, 1987.
Topics include writing a simple makefile, shell variables, internal macros, suffix rules, special description targets, maintaining libraries, and invoking `make` recursively.

Miscellaneous material

Dougherty, Dale; O'Reilly, Tim. *DOS Meets UNIX.* O'Reilly and Associates, 1988.
Describes the solutions available for integrating DOS and UNIX. Includes DOS, UNIX, and departmental computing, strategies to integrate DOS and UNIX, introduction to UNIX, DOS-UNIX networking, the virtual DOS environment, hardware issues, and where to buy products.

Dougherty, Dale. *UNIX in a Nutshell for HyperCard.* O'Reilly and Associates, 1989.
Contains a series of HyperCard stacks for Macintosh users of UNIX systems. Includes the main stack, introductory stack (demonstrates applicable HyperCard techniques), command stack for both System V and Berkeley commands, text editor stack, shell stack, programming tools stack, freeform searching capabilities, glossary, and topic index.

Kochan, Stephen G.; Wood, Patrick H. *UNIX System Security.* Macmillan, 1990.
Addresses each of the major security issues concerning UNIX users; includes information about administering passwords, auditing security, checking file permissions, data encryption, and setting up a restricted environment.

Glossary

This glossary defines terms, acronyms, and symbols that appear in this primer. Also included are terms that are a part of the everyday use of computers.

Symbols

&
The ampersand within the C shell causes the command preceding it to be run in background. Within the `mail` utility, it is the `mail` prompt.

$
The dollar sign within the Bourne shell indicates that the system is waiting for input; it is the default shell prompt. Within the C shell, it indicates variable names.

%
The percent sign in the C shell is its default shell prompt. Its presence on the screen indicates that the system is waiting for input.

*****

The asterisk is a metacharacter that represents a variable number or sequence of characters.

.
This symbol is called "dot." When referring to directories, it represents the current or working directory.

..
Similar to dot, the "dot dot" symbol stands for the parent directory of the current, or working directory.

/
This character, called slash, separates directories in a path name. It also represents the root directory when used by itself or when it is the first character in a path name.

****
This character is the backslash. On the command line, when it precedes a character, that character loses its special meaning. When it is the last character on a command line, it signals that the command line continues onto the next line.

<
The less than sign redirects command input. By default, input comes from your terminal; redirection can define input to come from a file instead. Users activate input redirection by placing the < on the command line followed by the name of the source file.

>
The greater than sign redirects command output. By default, output goes to your terminal; redirection can define output to go to a file, a printer, or wherever you specify. Users activate output redirection by placing the > on the command line followed by the name of the destination file.

>>
Two greater than signs append command output to a file. Use this symbol to avoid overwriting the contents of an existing file.

?
The question mark is a metacharacter that substitutes for any single character in file names.

|
The pipe symbol instructs the shell to pass the output of the command on the left as input to the command on the right. A pipeline is a group of commands joined by pipe connections.

!
The exclamation point reexecutes a command from the C shell history list. It is also referred to as bang.

!!
This command reexecutes the last command in the C shell history list.

;
The semicolon separates one C shell command from another on the command line, enabling the user to enter multiple commands at one time.

A

absolute path name
States the route to a file, starting from the root (/) directory. This path name always begins with slash (/).

access
To use the contents of a file or the services of a computer or computer network. Pertaining to one's ability or right to use the contents of a file.

access mode
One of the three modes of read, write, and execute permissions assigned to the three types of users (owner, group, and other). Access modes display in positions two through ten of a directory long listing initiated by the `ls` command.

account
A user's resources on a system. Usually, the system manager assigns these resources, such as creating a home directory, making an entry in the password file, and assigning certain system rights.

alphanumeric
A general term for alphabetic letters A-Z and a-z; numeric digits 0-9; and special characters (for example, @, #, $, %, &, *, and +) that can be processed by the system.

application program
Any data entry, update, query, or report program that processes data for users.

archive
To back up data onto a disk or tape.

argument
Technically, anything that follows the command name is an argument. Arguments pass information to a command to further direct its operation. An example of command syntax is:

command_name [*option*] [*argument*]

An option is an argument that alters the operation of a command. Options usually are single characters preceded by a hyphen (-), also called a flag ora switch. Next, an argument is the file, directory, or whatever upon which the command (directed by any options) is to operate.

arithmetic unit
The portion of the central processing unit (CPU) where arithmetic and logical operations are performed.

ASCII
American Standard Code for Information Interchange. A character code, of seven or eight bits, that represents letters, numbers, punctuation, and control characters. ASCII character representation is a standard used by many computers for data transfer and storage.

assembly language
A programming language directly related to the native instruction set (0s and 1s) of a computer.

B

background
A process that runs unattended, freeing the terminal keyboard so other programs can be initiated. Within the C shell, placing an ampersand at the end of the command line instructs the shell to run the process in the background.

bang
In the C shell, this term refers to the exclamation point's function of reexecuting a command from the history list.

backslash
The character \. On a command line, when it precedes a character, the character loses its special meaning. When placed at the end of a command line, it signals the shell that the command line continues onto the next line.

batch job
An noninteractive program. A program that is submitted by the user for subsequent execution. The submitting user does not have to be logged in for a batch job to execute.

binary
Meaning two: a zero (0) or a one (1). All input to a computer is converted into machine language, which is made up of zeroes and ones.

bindings.
See *key binding*.

bit
A binary digit. A single digit in a binary number, either 1 or 0. Groups of eight bits form bytes, which in turn represent characters and words.

block
A group of bits or characters transmitted as a unit.

Bourne shell
A shell available under ConvexOS. See *shell.*

BSD
Berkeley Software Distribution. A generic term used to refer to the version of the operating system created at the University of California at Berkeley. ConvexOS is based on BSD.

buffer
A reserved area in memory used to hold data while it is being processed. In a program, buffers hold some amount of data from files to be read or written.

bus
A common channel or pathway between hardware devices, either internally between components in a computer or externally between stations in a communication network, on which data is sent.

byte
A grouping of adjacent binary digits (bits) operated on by the computer as a unit. The most common size contains bits. One byte encodes one character.

C

C
A high-level programming language developed at Bell Laboratories that can manipulate the computer at a low level, much like assembly language. Its programs can be compiled into machine languages for most computers.

central processing unit
The circuitry that controls the interpretation and execution of instructions. It includes the arithmetic-logic and sometimes the control unit. Also known as the processor or CPU.

character string
A group of one or more alphanumeric characters considered as a single unit.

chip
An integrated circuit that holds anywhere from a few dozen to several million components such as transistors and resistors. Also known as a microchip.

command
Characters interpreted by the operating system as an instruction to execute. What you type following the command prompt is

interpreted as a command. Commands typically have three elements: command name, command options, and arguments.

command interpreter
See *shell*.

command line
The line on which the command prompt sits. Usually refers to the line where you type commands directly following the command prompt.

command mode
An operational state within the `vi` editor. In this mode, all characters typed are interpreted as editing commands; therefore, they do not go into the editing buffer.

command option
For ConvexOS, options affect how a command executes. They immediately follow the command name and usually begin with a hyphen (-).

command prompt
The symbol at the left of the screen signaling that the system is waiting for you to give it instructions—commands. In the C shell, the default command prompt is the `%`.

command syntax
The rules governing how a command and its arguments must be ordered on the command line.

communications
The electronic transfer of data from one location to another. Data communications refers to digital transmissions, and telecommunications refers to all forms of transmission, including analog voice and video.

configure
The act of connecting (physically and electrically) system components so they operate as a system, processing data in a certain way.

communications
The electronic transfer of information from one location to another.

computer
A device capable of solving problems of manipulating data by accepting data, performing prescribed operations (mathematical or logical) on the data, and supplying the results of these operations.

CPU
An acronym for the central processing unit. See *central processing unit*.

COVUEshell
A shell that is part of the CONVEX- to-VAX user environment that supports VMS-like commands, lexical functions, and qualifiers on CONVEX systems.

CRT
An acronym for cathode- ray tube, originating from the early versions of video and television screen that used cathode-ray tubes to generate screen displays. Another term for computer screen.

C shell
A shell created by William Joy at the University of California at Berkeley. Its programming constructs look much like the C programming language. Also called `csh`.

current directory
See *directory, current*.

cursor
A movable symbol on a display terminal that indicates the position where the next character will appear. The cursor usually is a solid blinking rectangular box or underline.

D

data
Basic elements of information to be processed or produced by a computer.

data bus
An internal pathway across which data is transferred to and from the CPU.

data communications
The digital transmission of information from one location to another.

data processing
The capturing, storing, updating, and retrieving of data and information.

device
See *peripheral*.

device name
The name assigned to a hardware device that represents its physical address.

digital
The use of numbers, originating from the word digit, or finger.

directory
A type of file that contains indexes to files that are physically scattered all over a disk or several disks. Also called directory file.

directory, current
Your current location within the ConvexOS file system. Also called the working directory. Files in the current directory are accessible without specifying a full path name.

directory, home
The directory into which a user is placed at the immediate completion of the login process.

directory, parent
The directory above the current directory in the hierarchical file system.

directory, root
The directory from which all other directories in that file system branch, either directly or indirectly. The slash (/) represents this directory. Absolute path names start in the root directory.

disk
A device consisting of magnetic surfaces upon which data and programs are stored, known as a mass storage device.

disk drive
A peripheral storage device that holds, spins, reads, and writes magnetic or optical disks.

display unit
A device that shows text and graphics on a video screen, such as a video display.

driver
A program that reformats data for transfer to and from a particular peripheral device. The electrical and mechanical requirements differ from one kind of device to another. Software drivers are used to standardize the format of data between devices and the CPU.

E

ed
The original text editor on UNIX systems. This line editor gave rise to both the `ex` and `vi` editors. See also *editor, line.*

edit
To check the correctness of data. To modify data by adding or deleting certain characters.

edit buffer
The temporary storage area used by an editor, such as `ed` or `vi`.

editor
A computer program designed to make it easy to review and alter a file or program. Also called a text editor.

editor, line
A text editor that creates and modifies files by displaying and operating on one line at a time or on a group of lines. Line editors are powerful because they can easily make a given change to many lines in a file. However, they are considered harder to use and less intuitive than screen editors. `ed` and `ex` are line editors.

editor, screen
A text editor that maintains a representation of a region of a file on the terminal screen. `vi` and `emacs` are screen editors.

editor, text
A general-purpose program used to prepare and manipulate text files. A text editor does not manage margins, headers, footnotes, typefaces, and point sizes.

electronic mail
The process of sending, receiving, replying, storing, and forwarding messages in digital format over telecommunications facilities. Also called email.

email
See *electronic mail.*

EMACS
(Editor macros) A programmable text editor developed at MIT. See *editor, screen.*

end user
See *user.*

enter
To type input and send it to the system by pressing the **RETURN** key.

environment
Characteristics of a user's account, such as the home directory, the group affiliation, and the environment and shell variables.

environment, variable
See *variable, environment*.

error message
Information about a problem that prevented successful completion of a program's or command's execution.

event number
A sequential number assigned by `csh` to each command that you issue. `csh` keeps a log (history) that shows each command and its event number.

ex
The line editing counterpart of the `vi` text editor.

execute
To carry out the instructions in a program. Same as run.

executable
A program in machine language that is ready to run in a particular computer environment.

exit
To leave the current mode or quit the program. Not always the same as quit.

external storage
Peripheral storage external to the computer, such as tape. Also called auxiliary storage or peripheral storage.

F

file
Any collection of data that is treated as a single unit. In word processing—a document; in programming—the source program and machine language programs; in data management—a collection of related records.

file access mode
The protection information for a ConvexOS file. Files may be read, written, or executed by three classes of users: the file's owner, members of the file owner's group, or others. The owner of the file sets the type of access allowed to each of the three classes. Also known as file permission mode.

file name
A name assigned by the user or programmer that identifies a file.

file protection
Operating system protection enables files to be assigned any combination of read, write, or execute access as determined by the file's owner. See *file access mode.*

file system
A method of cataloging files in a computer system. See *hierarchical file system.*

fork
One of the key system calls. Duplicates an existing process, thereby creating a new process.

G

G
Gigabyte. Actually, 1,024,000,000 bytes.

general-purpose computer
A computer designed to solve a wide class of problems. Refers to most computers, from micro to mainframe.

general-purpose controller
A peripheral control unit that can service more than one type of peripheral device, for example, a printer and a disk drive.

graphics
Creating and managing pictures using computers.

group
A set of users associated by a group ID (GID) number.

H

hands-on
The process of physically using a computer system.

hard copy
A printed copy of machine output in readable format.

hardware
All physical equipment, such as electronic, magnetic, and mechanical devices. Contrast with *software*.

hardware resources
CPU time, internal storage (memory), disk and tape devices, terminals, printers, and other peripheral devices.

header
The first record in a file that identifies the file. It includes the name of the file, date of last update, and other status data.

help
On-screen instruction regarding the use of a program.

hidden file
A disk file that has been given a status that prevents it from being viewed, changed, or deleted. Also used to prevent unauthorized access.

hierarchical file system
A scheme for storing files on a disk in which information is stored at different levels. Files are stored in a top-to-bottom, multilevel structure. Access to files starts at the top and proceeds down through the levels of the hierarchy.

history
A recorded list that shows the order in which you issued commands during a terminal session. Each command in a history list is recorded with its event number. A history list can be continued from one terminal session to another.

home directory
See *directory, home*.

information
Technically, data are raw facts and figures that are processed into information.

input
Any data ready for entry into a computer. Or, the act of entering data into a computer.

input device
A peripheral device that generates input for the computer such as a keyboard, scanner, disk, or tape.

input/output (I/O)
A transfer of data between the CPU and a peripheral device. Every transfer is an output from one device and an input into another.

input queue
A reserved segment of disk or memory that holds messages that have been received or job control statements describing work to be done.

input redirection
Reassignment by the shell of a program's standard input, changing the input source from a user's terminal to a file.

instruction
A statement in a programming language.

intelligent controller
A peripheral control unit that uses a built-in microprocessor for controlling its operation.

interactive processing
A dialog between the user and a computer. Contrast with batch processing.

interface
A connecting point between two pieces of hardware, or software, or between two systems.

internal storage
Addressable storage directly controlled by the CPU to store programs while they are being executed and while they are being processed. Also called internal memory, main memory, and primary storage.

interpret
To run a program one line at a time, where each line is translated into machine language, then executed.

interpreter
A high-level programming language translator that translates one program statement into machine language and executes it, then proceeds to the next statement.

invoke
To activate a program, routine, function, or process.

iteration
One repetition of a sequence of instructions or events.

J

job
A unit of work running in the computer. A job can be a single program or a group of programs that are required to run together.

job number
An identification number assigned to a job.

job queue
A line-up of programs ready to execute.

K

kernel
The memory-resident programs in an operating system, containing the system functions that are needed immediately and frequently.

key binding
The assignment of one or more keyboard keys as the initiator of a function.

keyboard
A set of input keys. Keyboards on terminals and workstations contain the standard typewriter keys and a number of specialized keys and features.

kill
A software signal process that terminates another process.

Korn shell
This shell is an extension of the Bourne shell and the standard for System V UNIX systems.

L

language
A set of symbols and rules used to convey information. See *programming language.*

laser printer
A printer that uses the copy machine electrophotographic method to print a page at a time.

letter quality
The print quality of an electric typewriter.

library
A collection of programs and data files.

library routines
A collection of prewritten functions or subroutines linked into the main program when compiled.

load
(1) To copy a program from disk or tape into memory for execution. (2) To fill a disk with data or programs. (3) To insert a disk or tape into its drive. (4) The current use of a system as percentage of total capacity.

local area network (LAN)
A communications network that services users within a confined geographical area. Ethernet is the most common LAN hardware.

local variable
A variable that is used only within the shell, routine, or function in which it is defined.

log
A record of computer activity.

login
A multi-step procedure used by the system to verify that the requesting user is an authorized system user. During this procedure the user is asked to enter a login name and password.

login directory
See *directory, home.*

login name
The name entered by a user during login. Your login name identifies you to the system. A user's home directory is often the same as their login name.

login session
Usage of a computer from the time of logging in to the computer system and logging out.

login shell
The shell given to you when you log in. See also *shell*.

logout
The process of ending your login session, which terminates the connection between your terminal and the computer system.

M

machine code, machine instructions
The computer's native language that tells it what to do and where to do it. In order for a program to run, it must be in the language of the machine executing it. Also called machine language.

machine-dependent
Programs that run on only one kind of computer.

mailbox
The computer storage assigned to a user for electronically transmitted mail.

mainframe
A large computer offering high-speed operation and extensive storage facilities. Reference may be to the main housing that holds the CPU, or to the small, medium, and large-scale systems that handle a few to hundreds of online terminals, varying sizes (megabytes) of main memory, and disk storage (gigabytes).

main memory
Same as main storage or memory. See *memory*.

main storage
Same as main memory or memory. See *memory*.

man pages
An online program (the `man` utility) that displays reference information as requested by the user.

megabyte
A megabyte is 1024 kilobytes, approximately one million bytes. Often written as 1 Mbyte or 1 Mb.

memory
The computer's working storage, which is a collection of RAM chips. It determines the size and number of programs that can

run at the same time, as well as the amount of data that can be processed instantly.

memory chip
A chip that holds programs and data either temporarily or permanently. The major categories of memory chips are RAMs and ROMs.

menu
A list of available options and commands displayed on a screen in an interactive program.

menu-driven
A program that is commanded by selecting options from a list.

metacharacter
A symbol interpreted by the shell. For example, while entering shell commands, the asterisk can be used to match any sequence of characters in a file name.

microcomputer
A small, low-cost computer that performs input, processing, storage, and output operations following a set of instructions. Uses a microprocessor for a CPU. Personal computers are microcomputers.

minicomputers
A computer that is distinguished from a microcomputer by higher performance, a more powerful instruction set, a higher price, and a greater selection of available programming languages and operating systems.

mode
An operational state that is set. Hardware and software can switch from one mode to another.

modem
A device that adapts a terminal or computer digital signal to a frequency that can be transmitted long distances over telephone lines. It also converts a transmitted signal back into a digital signal on the receiving end.

monitor
A high-resolution screen that displays output from the computer and input from the user.

multiprocessing
Simultaneous processing using two or more processors in one computer.

multitasking
The running of two or more programs in one computer at the same time. Multitasking is controlled by the operating system, which loads the programs and manages them until finished. The number of programs that can be effectively multitasked depends on the available system resources: CPU speed, memory, speed and capacity of peripheral devices.

multiuser
Many users working on a system at one time.

N

network
A group of computers able to communicate at high speed over long distances via special hardware and software connections. Two commonly used networks are local area networks (LANs) and wide area networks (WANs).

network file system
A file system that is present on the disk of one computer but made accessible to other systems via a network.

numeric data
Pertaining to numerals or to representation by means of numerals.

O

offline
Not connected to or not installed in the computer. A terminal not powered on, but physically connected, is offline. Disks and tapes unmounted and stored are considered offline. Contrast with *online*, meaning ready to go.

online
A peripheral device, such as a terminal or printer, that is ready to work. An online computer system implies data entry and updating. A transaction processing system updates the appropriate files when work is done, such as an order processing system. A realtime system is an online system that provides immediate response to input.

operating system (OS)
A set of programs designed to control the input and output operations of the computer, communicate with users, and schedule system resources (including hardware) to maintain a continuous operation of the computer system.

option
An argument that alters the operation of a command. Also called a switch or a flag. For example, in the command `ls -l`, the option is `-l`, which asks for the long format of the list of files within a directory.

ordinary files
Files used to store data and instructions: programs, documents, letters, databases, and other types of information. Contrast with *directory* (directory file) and *special file*.

output
Any computer-generated information transferred from memory to some auxiliary storage such as a disk or tape, or to a peripheral device such as a printer or terminal screen.

output redirection
Reassignment by the shell of a program's standard output to a file or device other than the terminal. The user can reassign standard output with the shell's > character.

overwrite
To place data in a location so as to destroy the data previously contained in that location.

owner
A file's creator or the user with control over the file's access privileges.

P

parallel processing
An architecture within a single computer that performs more than one operation at the same time.

parent directory
The directory immediately above the current directory.

parent process
A process that initiates another process (called a child). This initiation of a process is called forking. For example, the shell forks a child process to execute another command or a command external to the shell's language.

password
A personal code that validates an individual's identity or a code entered during login.

path name
A route through the file system that leads to a file. A path name consists of a list of directory names separated by the slash (/) character and a final file name. Path names are either absolute or relative.

path name, absolute
A route to a file starting from the root directory. This path name always begins with a slash (/).

path name, relative
A route to a file starting from the current working directory.

PC
A personal computer; a computer designed to be used by one user at a time. Typical PCs have a terminal screen, display, disk, processor, and memory.

peripheral
Any hardware device connected to a computer, such as a screen display (monitor), keyboard, printer, plotter, disk or tape drive, plotter, scanner, or mouse.

physical link
The electronic connection between two devices.

PID
See *process identification number.*

pipe
A connection between the standard output of one program and the standard input of another. The symbol for a pipe connection is the vertical bar (|).

pipeline
A group of commands joined by pipe connections.

printer
A device that converts computer output into printed pages.

print queue
Space reserved on disk to hold output designated for the printer until the printer is ready to receive it. See *spool.*

priority
A number associated with a process that the kernel's scheduler uses when determining which process should execute next. High-priority processes (those with lower priority numbers in ConvexOS) get larger and more frequent processing time.

privilege
See *file access mode.*

procedure
A group of statements or commands called (activated) as a unit.

process
(1) A program that is being executed on ConvexOS.
(2) Transforming raw data into useful information.

process identification number
A unique number assigned by the ConvexOS kernel to identify each process. Also called a PID.

process status
The current description of a ConvexOS process.

processor
See *central processing unit.*

program
A sequence of instructions that cause a machine to perform some function.

programming language
A language used to write instructions for a computer. The instructions written by a programmer are called source code are translated into the machine's language by a special program, either an assembler, compiler, or interpreter.

prompt
A message from the software that indicates that it is ready for input from the user.

Q

queue
A temporary holding place that indicates the order in which entries will be processed.

quit
To leave the current program.

RAM
Random-access memory is the same as memory.

read
To input data into the computer from a peripheral device, such as a disk or tape. Memory is said to be read when data is transferred from it to another memory location.

real time
Occurring in the computer at about the same time as an external event, keeping pace with that event.

recover
To retrieve the contents of a file from a backup or from an archive.

redirection
See *input redirection* and *output redirection*.

register
A CPU storage location that can be accessed rapidly.

relative path name
The route to a file that starts in the current working directory.

ROM
Read Only Memory is a memory element whose information can be read but not altered or written over.

restore
To recover archived or backed up data.

retrieve
To call up data that has been stored in a computer system.

root directory
The base directory of the ConvexOS file system.

routine
A set of instructions that perform a task.

scheduler
The part of the kernel that initiates and terminates programs in the computer according to their priority.

scrolling
To move information up or down, sometimes to the right or left, on the screen. To add a new line of information at the bottom of the screen, causing everything else on the screen to move up one line.

serial
One after the other.

session
The period of interactive computer use between login and logout.

shell
A command language that provides an interface to the operating system. As a command interpreter, the shell interactively accepts commands and arranges for the requested actions to occur. ConvexOS supports these shells: Bourne (`sh`), C (`csh`), Korn (`ksh`), and COVUEshell.

shell prompt
The character that identifies the type of shell you are using. When it displays , it signals the user that the system is ready for input.

single-user
Pertaining to a system that is usable by just one person at a time. Contract with *multiuser.*

software
A set of instructions called programs that make the hardware carry out specific operations.

software tools
Programs that aid in the development of other software programs.

sort
To resequence data, typically in alphabetical or numerical order.

special file
A file used as an interface to an I/O device. Special files usually reside in the /dev directory and come in two types: block special files for devices that can support file systems, and character special files for everything else. ConvexOS has a special file for each I/O device. Contrast with *ordinary files* and *directory* (directory file).

split screen
The display of two or more sets of data on screen at the same time, where one set of data can be manipulated independently of the other. These are created by the operating system or application software, not by hardware.

spool
To place a process in a queue where it waits its turn for some action. Spooling often refers to placing a print job in the queue.

standard input
A predefined source of input, which by default is your terminal. Also known at stdin. See also *input redirection*.

standard output
A place to which programs direct their output, which by default is your terminal. Also known at stdout. See also *output redirection*.

start-up
A routine that executes when a system is started. Its purpose is e alsoto customize the environment for its associated software.

status line
An information line displayed on the screen showing current activity.

sticky bit
A protection setting placed on a directory so that the files within can only be deleted by their owners. If a directory is writable, its files can be deleted regardless of their access modes.

storage device
A hardware unit that holds data, such as disk storage (for example, floppy diskettes, disk cartridges, and hard disks) and tape storage.

string
A contiguous set of alphanumeric characters.

subdirectory
A directory that hangs from another directory.

supercomputer
The fastest computer available. Typically used for simulations in petroleum exploration and production, structural analysis, physics and chemistry, and realtime animated graphics.

syntax
The rules for writing valid statements in a command language or programming language. It specifies how words and symbols must be put together.

syntax error
An error that occurs when a program cannot understand a command line, because it has been constructed using invalid words or symbols.

system
A computer system made up of the CPU, operating system, and peripheral devices.

system manager
The person in charge of operating and maintaining a system.

system program
Software that is either part of the operating system or control program.

system software
A category of programs used to control the computer and run application programs.

T

tape drive
The physical unit that stores, reads, and writes data on magnetic tape.

task
A program that runs as an independent unit. See also *multitasking*.

telecommunications
The communication of all forms of information, including voice and video.

terminal
An I/O device consisting of a keyboard and either a video screen or a printer. Terminals enable users to interact with a computer.

terminate
To end a process.

text
Data that consists primarily of letters, digits, and punctuation symbols.

text editor
See *editor, text*.

text file
A file composed solely of ASCII characters, usually structured into lines.

time-sharing
A technique for sharing a computer and its resources among several tasks, so that each task appears to have exclusive use of the computer. See also *scheduler.*

translate
To change one language into another; for example, assemblers, compilers, and interpreters (shells) translate their source language into machine language.

tree
A hierarchical file system in which directories branch off like tree branches, creating directory levels.

tutorial
An instructional book or program that takes the user through a prescribed sequence of steps in order to teach a product or subject.

type
In data or text entry, to press the keys on a keyboard.

U

UID
User Identification Number assigned to a user. Software uses these numbers when determining if a user has access rights to a file.

UNIX system
A portable, multiuser, time-sharing operating system first developed by Ken Thompson and Dennis M. Ritchie at Bell Laboratories in the early 1970s.

umask
An octal representation of the access permission associated with each file.

user
Anyone who uses a computer system.

user interface
The part of a software package that accepts commands from a person and displays information.

user name
See *login name.*

utility
A program that supports the operation of the computer. Programs that come under this category provide sort, backup, archive, and diagnostic routines.

V

variable
A variable holds an assigned value until a new value is assigned. Environment variables are available to programs and new shells; local variables are not inherited by subsequent shells.

variable, environment
Values set by the shell and available to subsequent shells.

variable, local (predefined and user-defined)
A value available only within the shell, routine, or function in which it is defined.

vi
A text editor (more specifically a screen editor) that operates under ConvexOS.

video terminal
A data entry device that uses a keyboard for input and a display for output.

W

wide area network (WAN)
A connection of computers that are physically far apart. This connection is usually through a public telephone system or some other long-distance information carrier.

wild card
See *metacharacter.*

working directory
See *directory, current.*

workstation
A computer system designed for use by one user. Typically, these systems have a powerful CPU, a large memory, multitasking software, and network interfaces. Some also have powerful graphics capabilities.

window
A region of a screen used for a particular function. Workstations usually support multiple windows, each used for a separate task. Windows are provided through a main windowing software package.

write
To send data to a file or I/O device.

write protect
Prohibits the erasing or changing of a disk file.

Index

Symbols

A

B

C

D

E

M

N

O

P

Q

R

S

T

U

V

W

X

Order Number
DSW-133

Document Number
710-012130-000