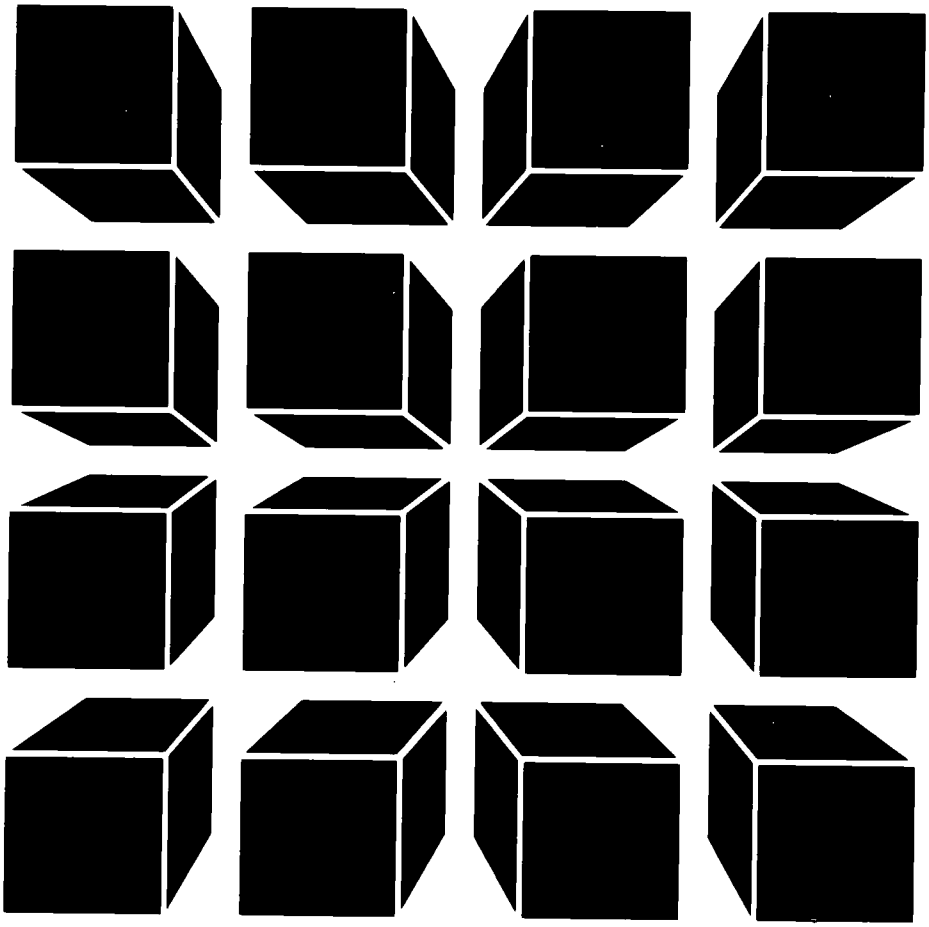
AT&T VIDEOTAPE LIBRARY

FUNDAMENTALS OF THE UNIX® SYSTEM - BASIC LEVEL



WORKBOOK

AT&T

CONTENTS

INTRODUCTION i

VOLUME 1—The Structure of the UNIX System

What You'll Be Able To Do After Volume 1 1-2
What Is an Operating System? 1-3
Two Major Components of a Computer 1-4
UNIX System Programs 1-5
Some UNIX System Characteristics 1-6
Terminal Features 1-7
What You Need to Get Started 1-8
Log In Procedure 1-9
Ending Your Session - Logging Off 1-10
What's New in the News? 1-11
The Shell and Your Commands 1-12
Command Line Format 1-13
What's a Valid Password? 1-14
Changing Your Password 1-15
Freezing and Scrolling Command Ouput 1-16
Listing Files 1-17
Displaying the Contents of ASCII Files 1-19
Volume 1 Exercise 1-20
Volume 1 Exercise - Answers 1-22
Volume 1 Summary 1-23

VOLUME 2—Basic User Communication

What You'll Be Able To Do After Volume 2 2-2
Reading Your Mail 2-3
Sending Mail 2-7
Reading Your Mail Using mailx 2-9
Sending Mail Using mailx 2-11
Mailx Commands 2-12
Volume 2 Exercise 2-14
Volume 2 Exercise - Answers 2-16
Volume 2 Summary 2-17

VOLUME 3—The File Structure

What You'll Be Able To Do After Volume 3 3-2
UNIX System Files 3-3
Every File Has a Name - The Rules 3-4
The UNIX File System 3-5
Some Standard Files 3-6
Your HOME Directory in the UNIX File System 3-7
Listing Files in the Current Directory 3-9
Path Names 3-12
Fully Qualified Path Names 3-14
Relative Path Names 3-15
Changing Directories—cd Command 3-16
Determining File Type—The file Command 3-17
Finding Files—The find Command 3-18
Finding Files—Recursive Listing 3-19
Making a Directory 3-20
Removing Regular Files 3-21
Copying, Moving and Renaming Files 3-23
Copying or Moving Directories 3-24
Volume 3 Exercise 3-27
Volume 3 Exercise - Answers 3-29
Volume 3 Summary 3-30

VOLUME 4—The vi Screen Editor

What You'll Be Able To Do After Volume 4 4-2
What Is an Editor? 4-3
Editing a File 4-4
Invoking vi 4-6
User to vi Dialog 4-7
Adding Text 4-8
Moving the Cursor 4-9
Displaying Buffer Contents 4-10
Deleting Text 4-11
Exiting vi and Saving the Buffer 4-13
Volume 4 Exercise 4-14
Volume 4 Exercise Answers 4-15
Volume 4 Summary 4-16

AT&T COMPUTER SYSTEMS
EDUCATION PROGRAM
INTRODUCTION TO STUDENT WORKBOOK

To the Student,

Welcome to the AT&T Videotape Library. We hope you will enjoy using the videotape lessons and this workbook. You will find the material both practical and easy to follow. You will be using what you learn, in your daily work, almost immediately.

If you haven't taken a class by videotape before, you are in for an exciting and fun experience, because this medium gives you control over the pace of your learning. Each videotape lesson is divided into segments which are clearly marked with colored panels, so you can find them easily while using fast forward or rewind. When you encounter new terms or concepts, you can immediately review and reinforce them by using the videotape player's rewind feature. If you find some material that is already familiar to you, simply use the player's fast forward button to jump ahead to the next segment. You can even stop the tape completely to take a break. The lesson will pick up right where you left off, and you won't miss a word.

This course, on **Fundamentals of the UNIX® System, Basic Level**, is designed to get you up and running on the UNIX System. You will learn how to create and access files, how to write files using the vi (TM) screen editor, and how to use basic communication tools on the UNIX System.

This workbook has been prepared as a support tool for this videotape course. Each chapter in the workbook follows one lesson. At any point in the lesson, you can stop the tape and use the workbook to study examples or descriptions. This workbook also contains summaries and reviews of the material you learned in each videotape lesson, and a job aid referring to new commands or concepts. At the end of each lesson, there are exercises that test your knowledge of the material you have just studied. If you have trouble with any of the exercises, it would be a good idea to go back and review that section of the lesson.

The videotapes, combined with this workbook, are designed to be complete and sufficient. However, if you have technical questions on the course contents that you cannot resolve after viewing the tape or reading this workbook, dial the **AT&T Telephone Support Line** at **1-800-247-1212, Extension 1001**, and one of our subject matter experts will return your call within twenty-four hours. The Telephone Support Line is available to you during the first six weeks following your purchase of the AT&T Videotape Library. If you are leasing the course, you have access to the Telephone Support Line for the duration of your lease.

Welcome to the world of the UNIX System, and the AT&T Videotape Library. Enjoy yourself.

VOLUME 1

The Structure of the UNIX System

WHAT YOU'LL BE ABLE TO DO AFTER VOLUME 1

- What is an Operating System?
- Logging on
- The command lines
- Basic UNIX System commands

WHAT IS AN OPERATING SYSTEM?

An operating system is the set of programs that manages the resources of the computer and coordinates its operations.

A typical operating system performs several functions, including:

- **Scheduling and executing processes** — Allocating portions of CPU time among the different tasks that are ready to run.
- **Managing storage** — Allocating main memory and auxiliary storage for running processes as required.
- **Performing I/O (input and output)** — Communicating with diverse types of peripheral equipment. This is the function of operating system programs called "device drivers."
- **Interrupt handling** — Taking appropriate action when the CPU is interrupted by requests for I/O, completion of I/O, timeouts, error conditions, etc.
- **Monitoring the system** — Maintaining current status information involves constant updating of special "data structures," or tables and lists.

TWO MAJOR COMPONENTS OF A COMPUTER

HARDWARE —physical devices—	circuit boards wires printers terminals tape drives disk drives
SOFTWARE —programs—	system programs utility programs application programs user-developed programs

UNIX SYSTEM PROGRAMS

The UNIX System programs are categorized into three basic groups:

1. The **kernel** program is the heart of the UNIX Operating System. The kernel's functions include monitoring the system, scheduling tasks, and managing data storage. Since it is always executing (running), it is always resident in main memory.

2. You communicate, or interface, with the UNIX System through the **shell**. The shell is a special utility program that interprets your command lines. After the shell does some processing on your commands, it hands them over to the kernel. It is the kernel that then gives the commands to the hardware where they are run. Every user gets his/her own shell to work in. In addition to an interface and a command interpreter, the shell is a programming language.

   The standard UNIX System V shell is called the Bourne shell (**sh**). However, there are other shells; for example, a restricted shell (**rsh**) and the Korn shell (**ksh**).

3. The **utility or application programs** include the many commands that are available on the UNIX System.

SOME UNIX SYSTEM CHARACTERISTICS

INTERACTIVE	responds to your input
MULTIUSER	many users can share resources
MULTITASKING	performs more than one task, called a *process*, at one time
PORTABLE	can run on a variety of computers
FLEXIBLE	the building block approach

TERMINAL FEATURES

- Full Duplex
- ASCII
- Even or no parity
- Correct baud
- UPPER and lower-case
- On-line

WHAT YOU NEED TO GET STARTED

Terminal	Type:
Connection	- Direct - Dial-up Phone number: - Local area network Access:
Login	
Password	* * * * * * * * * * * *

See your System Administrator for this information.

LOG IN PROCEDURE

Step 1: Turn your terminal on.

Step 2: See *login:* on your screen.

- Make phone connection?
- Access network?

Step 3: Enter your login and password.

```
Welcome to UNIX System V.
Have a great day!

you have mail

news: passwords downtime

Character erase is backspace.

$
```

ENDING YOUR SESSION -- LOGGING OFF

1. Hold down <CTRL> and type the letter 'd'. (Called a 'control d')

2. If using a dial-up connection, disconnect the phone.

3. Turn the terminal off or turn down the screen brightness.

WHAT'S NEW IN THE NEWS?

news: _____

To read all your new news:

```
$ news<CR>
```

To read one news item:

```
$ news _____ <CR>
```

To read two news items:

```
$ news _____ _____<CR>
```

To read all the news (old and new):

```
$ news -a <CR>
```

THE SHELL AND YOUR COMMANDS

The **$** is the shell's way of asking you what you want to do. Remember that you communicate with the UNIX System through the shell. Whenever a **$** prompt is on your screen, the shell is waiting for a command.

The **date** command is useful if you don't have a watch or aren't near a clock. After the UNIX System runs the **date** command, you'll see the **command output** from **date** on your screen.

```
$ date<CR>
Mon Mar 31 11:29:49 EST 1986        ← the command output from date
↑   ↑      ↑        ↑    ↑
day date   time          year
                    timezone
                (Eastern Standard Time)
```

If you type a command incorrectly, you will receive an error message or **diagnostic output** on your screen:

```
$ datte<CR>
datte: not found      ← diagnostic output
```

If you notice a typing mistake before you enter a <CR>, you can erase the individual characters OR you can erase the entire line. Typing @ will erase an entire line. It also produces an automatic <CR>. Therefore, the new line you type will be under the erased line with NO $ prompt. Here's an example:

```
$ datte@ ← @ erases the entire line
date<CR>          ← you type the correct line
Mon Sep  9 15:26:43 EDT 1986        ← the output from date
```

COMMAND LINE FORMAT

$ *news*<CR>	A *command* is a program that tells the UNIX System to do something.
$ news *downtime*<CR>	An *argument* tells the command on what to perform its action.
$ news *-a*<CR>	An *option* modifies the command.

General Command Line Format:

$ command ±option argument(s)<CR>

WHAT'S A VALID PASSWORD?

- MUST contain a minimum of 6 characters
- Only 8 characters are recognized
- MUST use at least two letters (upper or lowercase) and one numeric or punctuation type character

Valid passwords:	Invalid passwords: WHY?
?frogs	tara1
Mexico85	ryebread99
-crunch-	12-18-85
	Bubbles

CHANGING YOUR PASSWORD

THE passwd COMMAND

	user typed:
$ **passwd**<CR>	
Changing password for tmm	
Old password:<CR>	*!crunch2*<CR>
Sorry.	
$	

$ **passwd**<CR>	
Changing password for tmm	
Old password:<CR>	*!Crunch2*<CR>
New password:<CR>	*blimp*<CR>
Password is too short - must be at least 6 digits	
New password:<CR>	*blimp5*<CR>
Re-enter new password:<CR>	*blimp5*<CR>
$	

FREEZING AND SCROLLING COMMAND OUTPUT

$ cal 1752<CR>

```
                                1752

      Jan                   Feb                   Mar
S  M Tu  W Th  F  S   S  M Tu  W Th  F  S   S  M Tu  W Th  F  S
                1  2  3  4                  1   1  2  3  4  5  6  7
 5  6  7  8  9 10 11   2  3  4  5  6  7  8   8  9 10 11 12 13 14
12 13 14 15 16 17 18   9 10 11 12 13 14 15  15 16 17 18 19 20 21
19 20 21 22 23 24 25  16 17 18 19 20 21 22  22 23 24 25 26 27 28
26 27 28 29 30 31     23 24 25 26 27 28 29  29 30 31
<CTRL s><CTRL q>
      Apr                   May                   Jun
S  M Tu  W Th  F  S   S  M Tu  W Th  F  S   S  M Tu  W Th  F  S
                1  2  3  4                  1  2      1  2  3  4  5  6
 5  6  7  8  9 10 11   3  4  5  6  7  8  9   7  8  9 10 11 12 13
12 13 14 15 16 17 18  10 11 12 13 14 15 16  14 15 16 17 18 19 20
19 20 21 22 23 24 25  17 18 19 20 21 22 23  21 22 23 24 25 26 27
26 27 28 29 30        24 25 26 27 28 29 30  28 29 30  <CTRL s><CTRL q>
                      31
      Jul                   Aug                   Sep
S  M Tu  W Th  F  S   S  M Tu  W Th  F  S   S  M Tu  W Th  F  S
                1  2  3  4                  1         1  2 14 15 16
 5  6  7  8  9 10 11   2  3  4  5  6  7  8  17 18 19 20 21 22 23
12 13 14 15 16 17 18   9 10 11 12 13 14 15  24 25 26 27 28 29 30
19 20 21 22 23 24 25  16 17 18 19 20 21 22
26 27 28 29 30 31     23 24 25 26 27 28 29
                      30 31  <CTRL s><CTRL q>
      Oct                   Nov                   Dec
S  M Tu  W Th  F  S   S  M Tu  W Th  F  S   S  M Tu  W Th  F  S
 1  2  3  4  5  6  7                  1  2  3  4            1  2
 8  9 10 11 12 13 14   5  6  7  8  9 10 11   3  4  5  6  7  8  9
15 16 17 18 19 20 21  12 13 14 15 16 17 18  10 11 12 13 14 15 16
22 23 24 25 26 27 28  19 20 21 22 23 24 25  17 18 19 20 21 22 23
29 30 31              26 27 28 29 30        24 25 26 27 28 29 30
                                            31
```

<CTRL/s> - Freezes your screen.

<CTRL/q> - Scrolls your screen.

LISTING FILES

When you logged on the UNIX System you landed in your home directory. Every login has a home directory (you'll find out more about this later in this course). When you first receive a login from your system administrator, there probably won't be any files in your home yet. Later in this course you'll learn how to create files. These files can be listed using the **ls** command.

```
$ ls<CR>
bin
budget
calendar
friends
fun
letters
misc
parts
poem
project
why.unix
work
$
```

LISTING FILES

Some of these files are ASCII files and some are other directories. To find out the files that are directories, use the **ls** command with the **-p** option.

```
$ ls -p<CR>
bin/
budget/
calendar
friends
fun/
letters/
misc/
parts
poem
project/
why.unix
work/
$
```

DISPLAYING THE CONTENTS OF ASCII FILES

$ cat calendar<CR>
11/28 Thanksgiving DAY OFF! ! ! !
7/4 JULY IV -- DAY OFF! ! ! !
12/25 XMAS Day
1/1 New Years Day
2/14 Happy Valentines Day!
11/27 Thanksgiving
9/4 Labor Day -- DAY OFF! ! !

$ **cat ppoem**<CR>
cat: cannot open ppoem

$ **pg poem**<CR>
(prints poem by screenfuls)

Note: To read a directory, use ls
DO NOT use cat or pg

VOLUME 1 EXERCISE

1. List tasks performed by an operating system.

   _____

   _____

   _____

   _____

   _____

2. List several characteristics of the UNIX Operating System.

   _____

   _____

   _____

   _____

   _____

VOLUME 1 EXERCISE

3. What is a command and what is an option?

4. What is the format of a command line?

5. How can you change your password?

VOLUME 1 EXERCISE - ANSWERS

1. Page 1-3

2. Page 1-6

3. Page 1-13

4. Page 1-13

5. Page 1-15

VOLUME 1 SUMMARY

The Structure of the UNIX System

COMMUNICATING WITH OTHERS		
Command	Description	Example
news	Displays specific news items	$ news shutdown<CR>
	Displays all news items old and new	$ news -a<CR>
DISPLAYING INFORMATION		
Command	Description	Example
cal	Displays a calendar for a month or year	$ cal 12 1985<CR> December 1985 S M Tu W Th F S 1 2 3 4 5 6 7 8 9 10 11 12 13 14 . . . . .
date	Displays the current date and time	$ date<CR> Mon Jan 15 09:35 ...
who	Tells who is currently on the system	$ who<CR> joe tty10 11:24 bob tty14 10:12

VOLUME 1 SUMMARY

FILES		
Command	Description	Example
ls	Displays the names of files.	$ ls<CR> notes plans screeninfo
	Displays the names of files. Directories are followed by a /	$ ls -p<CR> notes plans/ screeninfo/
cat	Displays the contents of a file.	$ cat letter<CR> Dear Janet, Hello There... .....
pg	Displays the contents of a file in screenfuls.	$ pg letter<CR>

VOLUME 1 SUMMARY

PROTECTING YOURSELF		
Command	Description	Example
passwd	Changes your password; you must know your old password to change it. New password must be 6-8 characters; 2 alphabetic and 1 numeric or punctuation character.	$ passwd<CR>
SPECIAL KEYS		
Key	Description	Example
<BS>	Most UNIX Systems are already set up so that you can use the <BACKSPACE> key to erase a character.	
@	Standard line erase. Gives automatic <CR>.	$ datte@ date<CR> Mon Jan 15 09:35 ...
<CTRL d>	Logs you off the UNIX System.	

VOLUME 2

Basic User Communications

WHAT YOU'LL BE ABLE TO DO AFTER VOLUME 2

- Reading UNIX Systems mail
- Sending UNIX Systems mail
- Reading mail with mailx
- Sending mail with mailx

READING YOUR MAIL

Mail is a way for all the users of the UNIX System to communicate with each other. How do you know when you have mail? You'll receive a message when you log on. Also, while you're logged on, the UNIX System periodically checks to see if any new mail has arrived for you. If there is new mail, you will see the message *you have mail*. The UNIX System will wait until it's safe to notify you about this new mail; it will not interrupt anything you're doing.

Any mail that you receive gets delivered to your own personal UNIX System mailbox. Just like the U.S. mail, your UNIX System mail remains in your mailbox until you go and get it. To read your mail type:

```
$ mail<CR>
From michael Fri May  3 11:27 EDT 1985      ← header
Meeting on Monday.
See you there,                              ← Mike's message
Mike
?                                           ← mail prompt
```

After you type **mail**, the first message in your mailbox will be displayed. The first line will be a header. The header contains the login of the person who sent you the message and the date and time the message was sent. The mail message is displayed below the header. Finally, you will get a ? prompt. This prompt is the **mail** command's way of asking what you want to do with this message.

READING YOUR MAIL

```
$ mail<CR>
From janet Mon Mar 10 11:41 EST 1986
Meeting tomorrow at 2PM
to discuss the project.
See you there,
Janet
? _____
     ↑
```

s<CR>	**saves the message in the file** *mbox*
d<CR>	**deletes the message from your UNIX mailbox**
q<CR>	**quits**
<CR>	**skips the message; leaves message in your mailbox**
m login<CR>	**forwards the message to another user**

READING YOUR MAIL

Here are some things you might want to do with a mail message:

d<CR> You may just want to 'throw away' the message after you've read it. Typing "**d<CR>**" to the "?" prompt will delete the message from your mailbox.

s<CR> If the message is important, you may want to save it. Typing "**s<CR>**" to the "?" prompt will save the mail message in a file named *mbox*. The message can be saved in another file by simply typing a filename after the "s". For example, "**s notes<CR>**" will save the message in a file named *notes*. There MUST be a space between the "s" and the filename.

q<CR> If you don't feel like reading all your mail messages, type a "**q<CR>**". This lets you quit early and get the shell prompt back. Otherwise, after you 'tell' the mail command what to do with the message you just read, the next message in your mailbox will be displayed (or you'll get the shell prompt back if there are no more messages).

x<CR> A "**x<CR>**" response to the "?" prompt, like the "q<CR>", will exit mail and return the shell prompt. However, the "x<CR>" also forgets any changes made to your mailbox. For example, any messages that you deleted would be put back into your mailbox. The "x<CR>" exits mail, leaving your mailbox intact.

READING YOUR MAIL

p<CR>	A "**p<CR>**" will print the current message again. This might be helpful if the message scrolled off your terminal screen before you could read it. After the message is printed again, you will get another "?" prompt, asking what should be done with the message.
w<CR>	A "**w<CR>**" is exactly the same as a "s<CR>" except the message is saved without the header.
<CR>	A "**<CR>**" will display the next message in your mailbox. If there are no more messages in your mailbox, you will get the shell prompt. Any messages you pass by with a "<CR>" will remain in your mailbox. The next time you type the mail command to read your messages, you will again see the messages you passed by.
m login<CR>	A "**m login<CR>**" will forward the message to another user (login). For example, "m harry<CR>" will forward the current message to login harry. Leave a space between "m" and the login. The message will not only be forwarded to the specified login but also be deleted from your mailbox.
h<CR>	If you forget this list of possible responses, "**h<CR>**" will display a brief summary.

SENDING MAIL

```
$ mail janet kal root<CR>          ← To whom
I can't make the meeting.
I'm in a UNIX class.
How about next week.               ← Message

Talk to you later,
Joe
.<CR>                              ← Send it!
$
```

Either a dot (.) or a <CTRL/d> on a line by itself will send your mail.

If the mail can't be delivered, for example, you misspell a login or try to send a message to a login that doesn't exist, you'll receive a *dead.letter* message from the **mail** command. Look at the following sequence:

```
$ mail debbie jmg bob<CR>
I completed the program today.  I'll send you a copy<CR>
to review.<CR>
.<CR>
mail: can't send to jmg
Mail saved in dead.letter
$
```

Notice that you are told which login did not receive the mail message. It was sent to debbie and bob, but not jmg. In addition, the mail message was saved in a file called *dead.letter* so that you don't have to type it again if you want to resend it. Later, you will learn how to resend a mail message using the *dead.letter* file.

READING YOUR MAIL USING mailx

```
$ mailx<CR>
mailx version 2.14 11/10/83 Type ? for help.
"/usr/mail/ep": 3 messages 3 new
>N 1 djg  Mon Dec 1 15:36   6/307 Meeting
 N 2 sar   Tue Dec 2 11:45  22/531 New Parts
 N 3 jmg  Tue Dec 2 12:23  34/953 Project
? _____
   ↑
```

n<CR> **displays message number** *n*

d<CR> **deletes current message from your UNIX mailbox**

s<CR> **saves the current message in the file** *mbox*

<CR> **skips the message; next message is displayed**

READING YOUR MAIL USING mailx

```
$ mailx<CR>
mailx version 2.14 11/10/83 Type ? for help.
"/usr/mail/ep": 3 messages 3 new
>N 1 djg   Mon Dec 1 15:36   6/307 Meeting
 N 2 sar   Tue  Dec 2 11:45 22/531 New Parts
 N 3 jmg   Tue  Dec 2 12:23 34/953 Project
? _____
   ↑
```

m logins<CR>	**mails a message to other users**
q<CR>	**quits**
h<CR>	**displays the message summary**
R<CR>	**sends a response to the sender of the current message**

SENDING MAIL USING mailx

```
$ mailx tct kal<CR>
Subject: Meeting<CR>
I can't make the meeting.
I'm in a UNIX class.
How about next week.

Talk to you later,
Joe
~.<CR>
```

SOME mailx COMMANDS

When using **mailx** to send mail interactively, you will be prompted to enter a subject. After entering a subject and pressing <CR>, you are in what's called **mailx** *input mode*. In input mode, you can either type your mail message or execute **mailx** commands that add to its power and flexibility. Some of these commands are discussed below; however, for a complete list, refer to the **mailx** manual page in the **UNIX System V User's Reference Manual**.

Printing the Message:

The ~**p** command displays (prints) the message on the terminal screen. This is useful to display the message at any time before sending it.

Quitting:

The ~**x** command quits without sending the message. This is useful if you begin a mail message and then decide that you really don't want to send it after all.

Entering a Carbon Copy List:

The ~**c** *names* command adds a carbon copy list to the mail message. For example:

```
$ mailx jwh dwg<CR>
The report is in the mail!<CR>
~c fjp docs<CR>
Lisa<CR>
.<CR>
$
```

Above, login *fjp* and *docs* are on the carbon copy list. The message will be sent to the logins entered on the **mailx** command line AND entered after the ~**c** command.

Reading In A File:

The ~**r** *file* command reads the contents of a file into the mail message. For example, this might be useful to mail some text along with the contents of a file:

```
$ mailx sar dwg<CR>
Here's the report I promised you!<CR>
~r report1<CR>
"report1" 599/18219
Lisa<CR>
.<CR>
$
```

After the file is read in, the ~**r** command responds with the name of the file and the number of lines/characters in the file. In the example above, 599 lines containing 18219 characters were read into the mail message.

VOLUME 2 EXERCISE

1. How can we send the same message to several people on the same system?

2. How can we read our message and save it in a file called mymesg?

3. What does ~r commands do on mailx?

4. What is the difference between exit and quit in mail?

5. How can you quit in the middle of writing a mesg in mailx?

VOLUME 2 EXERCISE

6. How can you get help in the middle of a mail session?

7. What is dead.letter file?

VOLUME 2 EXERCISE - ANSWERS

1. Page 2-7
2. Page 2-5
3. Page 2-13
4. Page 2-5
5. Page 2-12
6. Page 2-6
7. Page 2-8

VOLUME 2 SUMMARY

Communicating on the UNIX System

Command	Description	Example
mail	Sends mail from one user to another. End input with a dot (.) on a line by itself or <CTRL/d>	$mail julie<CR> meeting at 2PM <CR> .<CR>
	Read your mail. Type "h<CR>" at "?" prompt to see possible responses	$ mail<CR> From jrs ... see you after work ? h<CR>
mailx	Sends mail to other user(s). Type "?<CR>"at "?" prompt to see possible responses End input with a ~. or <CTRL/d>	$ mailx bob<CR> Subject: Meeting meeting at 2PM ~.<CR>
	Read your mail.	$ mailx<CR>

VOLUME 3

The File Structure

WHAT YOU'LL BE ABLE TO DO AFTER VOLUME 3

- Distinguish between a regular file and a directory.
- Choose names for files.
- Use path names to identify files.
- Find "where you are" on the UNIX System. (your current directory)
- List the files that are in a directory.
- Move around and explore the UNIX System.
- Locate a file and print its path name.
- Copy, move, remove, and rename regular files.
- Organize files by creating, removing, and renaming directories.

UNIX SYSTEM FILES

A *file* is a named place to store information

regular or *ordinary* files	store text (e.g., letters, memos), data, programs
directory files	• store the names of other files • act as table of contents • used to group related files
special files	used for devices (e.g., printers, terminals)

EVERY FILE HAS A NAME — THE RULES

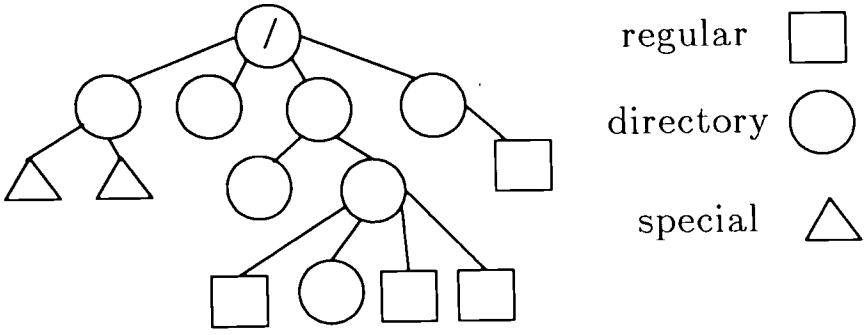
- Should describe the contents
- Maximum of 14 characters
- Recommend upper or lower-case letters, numbers, underscore(_), or dot(.)

Valid file names:	What's wrong with these?
notes	Sue{1
new_mail	John_rubenstein
Update.86	Mail-Box
README	AT+T
.profile	

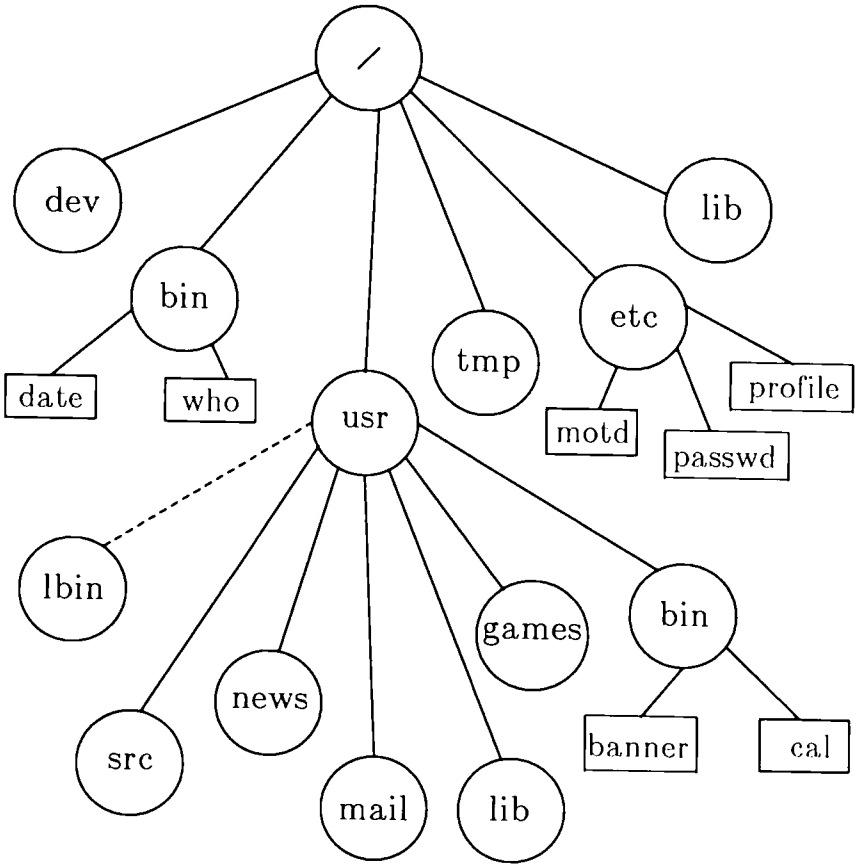
THE UNIX FILE SYSTEM

A *file system* is the organization of files

- Hierarchical
- An upside-down *tree*
- Begins at the root directory named /



SOME STANDARD FILES



YOUR HOME DIRECTORY IN THE UNIX FILE SYSTEM

your *HOME directory* is your position in the file system when you first log in

your *current directory* is your position in the file system at any given time

While logged onto the UNIX System, you are always positioned in a directory, called your *current directory*. Think of your current directory as a "you are here" marker. When you first log on, your current directory, or position in the file system, is your *HOME* or *login directory*. The **pwd** (**p**rint **w**orking **d**irectory) command prints the *fully qualified path name* of your current directory. The **pwd** command is especially useful once you begin moving around the file system since it always tells where you are.

The system administrator decides where your HOME directory will be. One of the fields of the *passwd* file in the /etc directory (*/etc/passwd*) specifies the full path name of your HOME. Typically, the HOME directories of users doing similar work are grouped together under one parent directory.

YOUR HOME DIRECTORY IN THE UNIX FILE SYSTEM

Your HOME directory is your own **subsystem** or **subtree**. Users are free to make directories, create files, rename files, remove files, etc. in their HOMEs. Depending on file permissions, users may or may not have permission to do these things in other directories. You can have as broad and deep a tree structure in your subsystem as the auxiliary storage can accommodate. Periodically, unnecessary files should be removed. In fact, the system administrator may ask users to clean up old files when disk space is low.

A directory under another directory is sometimes called a **subdirectory**. If you haven't created any files in your HOME directory, your HOME directory is probably *empty*.

LISTING FILES IN THE CURRENT DIRECTORY

The **ls** command will list files in the current directory. Remember, when you first log on, your current directory is your HOME directory. The **-p** option will print a slash (/) after files that are directories.

```
$ ls -p<CR>
bin/
budget/
calendar
friends
fun/
letters/
misc/
parts
poem
project/
why.unix
work/
$
```

Notice that the files are displayed in ASCII order. In ASCII, upper case come before lower case letters.

If an **ls** does not list any files, it means that there aren't any files in the directory, i.e., an *empty* directory.

```
$ ls<CR>
$
```

LISTING FILES IN THE CURRENT DIRECTORY

The **-C** option will list the file names in columns. The **-C** and **-p** options can be combined to list file names across and print a slash (/) after those files that are directories.

```
$ ls -Cp<CR>
bin/    calendar  fun/    misc/  poem     why.unix
budget/ friends   letters/ parts  project/ work/
$
```

The **ls** command permits multiple options to be grouped together, e.g., **-Cp**.

The **-l** option, provides a "long listing" of a directory:

```
$ ls -l<CR>
total 12    (a)
drwx------    2 jrs   other   128 Jun  9 12:58    bin
drwx------    2 jrs   other   128 Jun  9 13:58    budget
-rw-r--r-     1 jrs   other   173 Jul  7 14:01    calendar
-rw-r--r-     1 jrs   other   613 Jul 22 15:56    friends
drwxr-xr-x    2 jrs   other   112 Jul  7 15:20    fun
drwx------    2 jrs   other   160 Jul  9 13:48    letters
drwxr-xr-x    3 jrs   other   288 Aug  6 08:29    misc
-rw-r-----    1 jrs   unixc   355 Jul  8 14:35    parts
drwxr-x---    4 jrs   unixc    96 Jul 22 09:38    project
-rw-r--r--    1 jrs   other   157 Jun 12 14:34    why.unix
drwx------    2 jrs   other   240 Jul 28 08:32    work
$
(b) (c)       (d)(e)  (f)     (g)  (h)            (i)
```

LISTING FILES IN THE CURRENT DIRECTORY

The information that is displayed from a long listing is described below:

(a) Total disk blocks occupied by all the files listed. This is NOT the number of files in the directory.

(b) The first character in each line below the total line indicates the file type. A **d** means the file is a directory. A — means the file is a regular file.

(c) The next 9 characters specify the permissions of the file. Permissions will be discussed later.

(d) Links to the file. The link count is the number of directories that a file is listed in. At minimum, the link count for directories is two, since every directory is listed in itself and listed in its parent. There is an additional link for each subdirectory. For regular files, the link count will always be one unless the **ln** command has been used.

(e) Login id of the owner of the file.

(f) Group id of the file.

(g) Bytes (characters) in the file.

(h) Date and time the file was last modified.

(i) Name of the file.

PATH NAMES

A *path name* describes the route that the system must follow to locate a file.

A *fully qualified path name* describes the location of a file from the root.

A *relative path name* describes the location of a file from "where you are" (your current directory).

A path name consists of zero or more directory names, separated by slashes (/), ending with the target file name. There are two types of path names: **fully qualified** and **relative**:

- A *fully qualified path name* describes the location of a file from the root. This path name always begins with a /. Following the first / is a list of directories, each separated by a slash, ending with the target file name. A fully qualified path name is also called a **full** path name or an **absolute** path name. The **pwd** command prints the full path name of your current directory.
- A *relative path name* describes the location of a file from "where you are" (your current directory). This path name never starts with a /. The shortest relative path name is just the file name itself. For example, in the command line:

  $ **cat calendar**<CR>

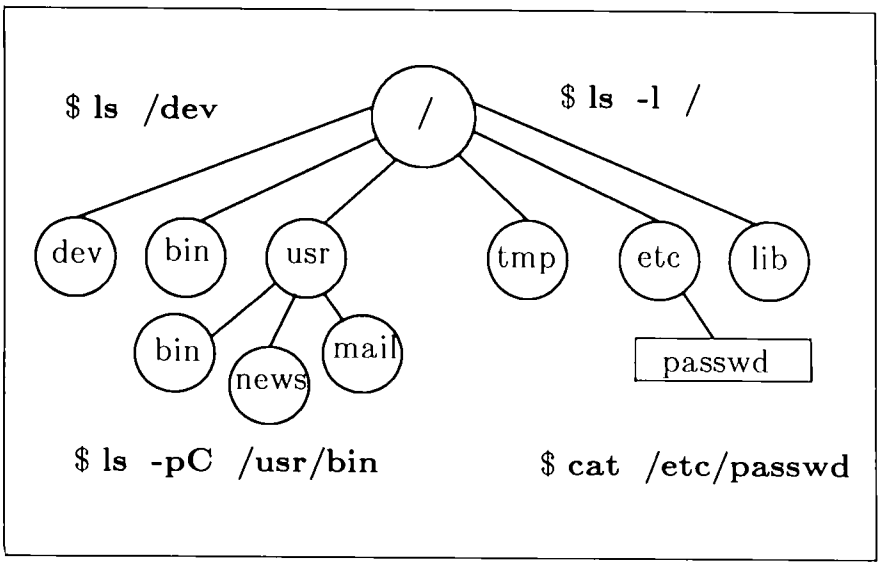
  the file name, *calendar*, is a relative path name that means the file is in the current directory. A relative path name of a file located in a directory other than the current is a list of directories, each separated by a slash, ending with the target file name. A relative path name is also called a **partial** path name.

PATH NAMES

The proper path (relative or full) to a file must always be specified or you will receive an error message that the file can't be found. Both relative and full path names are slash-separated lists of directories ending with the target file name. The difference is that full path names start with a / and relative path names don't. So why use one type of path name over the other? When working with files close to your current directory, the relative path name will probably be shorter. Otherwise, a full path name may be shorter.

FULLY QUALIFIED PATH NAMES

- always begin with /
- search begins at root



RELATIVE PATH NAMES

Relative path names never start with /. They always specify where a file is in relation to the current directory. If you forget your current directory, use the **pwd** command to print the full path name.

To specify a relative path name of a file below the current directory, just start listing directories, separated by slashes, ending with the target file name. The command to list the contents of the *notes* directory using a relative path name is:

```
$ ls project/notes<CR>
```

When specifying a relative path name of a file above the current directory, the special notation .. (pronounced "dot dot") must be used to name the parent directory. For example, **../..** means parent of the parent (2 directories above). Remember,

- when moving down in sub-directories, use file names
- when moving up in directories, use .. ("dot dot")

For example, suppose your current directory is *notes*. The command to **cat** the file *air.let* (located in the *letters* directory) using a relative path name is:

```
$ cat ../../letters/air.let<CR>
```

The first .. refers to the parent directory *project*, the next .. refers to the parent of the *project* directory. Then use the file name *letters* to go down, ending with the target file name, *air.let*.

No matter what directory you are in, current directory is always "dot" and parent directory is always "dot dot". Each time you change directories (with the **cd** command), your current and parent directories also change.

CHANGING DIRECTORIES — cd COMMAND

When you first log into the UNIX System, your current directory is your HOME directory. At times, however, it is necessary to move around in the file system to access and work on other files in other subdirectories. To change directories, use the **cd** command followed (optionally) by the path name to the new directory. The path name to the new directory can be a full or relative path name, whichever is shorter or easier. For example, to change to the directory called */usr/games*, the command is:

```
$ cd  /usr/games<CR>
$
```

The absence of error messages indicates success! However, to confirm your current directory, use the **pwd** command.

```
$ pwd<CR>
/usr/games
$
```

So why change directories? It makes using the files in the directory easier. If the file you want to use is in the current directory, the relative path name is just the file name.

Fortunately, there is an easy way to return to your HOME, or login directory, from anywhere in the file system. It's just **cd** without any path name.

```
$ cd<CR>
$
```

DETERMINING FILE TYPE — THE file COMMAND

```
$ file friends<CR>
friends:     ascii text
$

$ file poem<CR>
poem:        English text
$

$ file /bin/who<CR>
/bin/who:    3b20 executable
$
```

FINDING FILES — THE find COMMAND

```
$ find . -name poem -print<CR>
./poem
./misc/poem
$
```

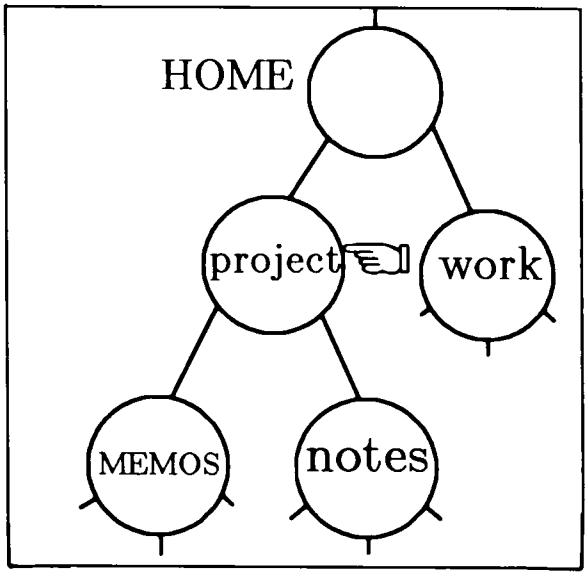
```
$ find / -name resume -print<CR>
find: cannot chdir to /usr/dump
/instr/spw/resume
find: cannot open /class/docs
find: cannot chdir to /crsdev/comm
/class/ucdir/uc1/resume
find: cannot chdir to /local/admin
          .
          .
          .
$
```

FINDING FILES — RECURSIVE LISTING

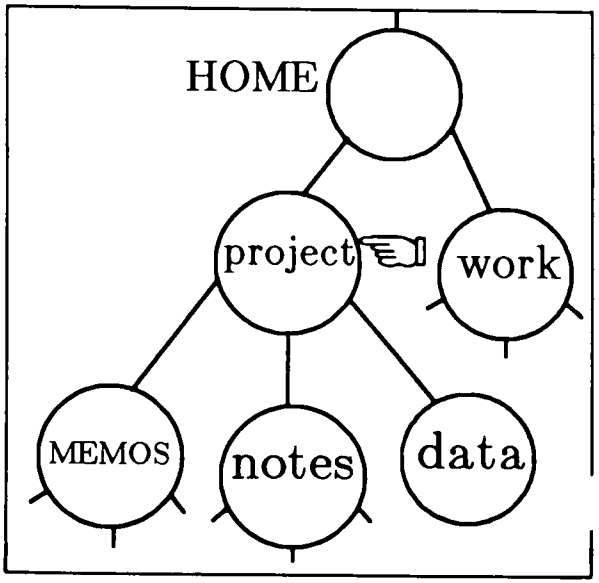
```
$ ls -CpR<CR>
bin/     calendar fun/    misc/ poem     why.unix
budget/ friends  letters/ parts project/  work/


bin:
addx    myls    nprint1  pass      timecard
lookup  nfiles  nprint2  searcher  whoall


budget:
budget87 budget89   budget90 budget92 budget9tot
budget88 budget8tot budget91 budget93 budgettot


fun:
ATTlogo  dictionary  heart  thanks  twilight.unix

                .
                .
                .
$
```

MAKING A DIRECTORY

```
$ ls -p<CR>
MEMOS/
notes/
$
```

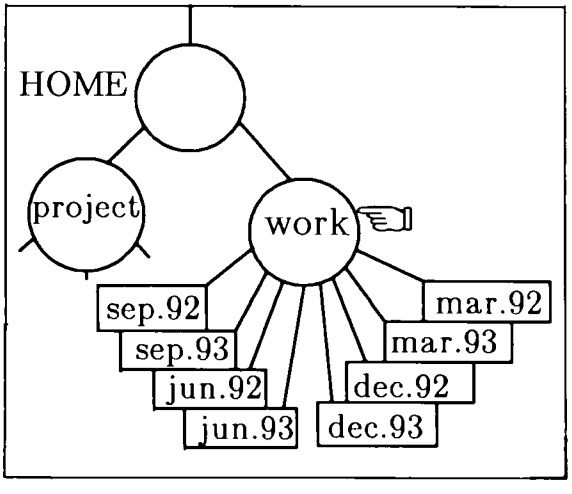



```
$ mkdir data<CR>
$ ls -p<CR>
MEMOS/
data/
notes/
$ ls data<CR>
$
```



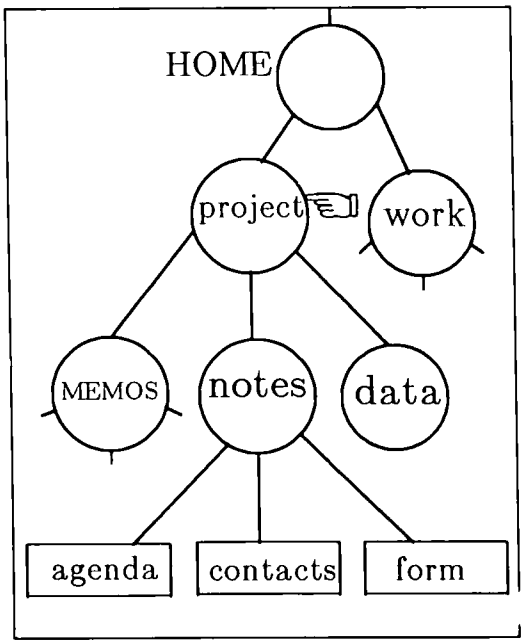
REMOVING REGULAR FILES

```
$ cd ../work<CR>
$ ls<CR>
dec.92
dec.93
jun.92
jun.93
mar.92
mar.93
sep.92
sep.93
$ rm mar.92<CR>
$ rm -i *.93<CR>
dec.93: ? y<CR>
jun.93: ? n<CR>
mar.93: ? n<CR>
sep.93: ? y<CR>
$ ls<CR>
dec.92
jun.92
jun.93
mar.93
sep.92
$
```



REMOVING DIRECTORIES

```
$ cd ../project<CR>

$ rmdir data<CR>
$


$ rmdir MEMOS<CR>
rmdir: MEMOS not empty

$ rm MEMOS<CR>
rm: MEMOS directory

$ rm -r MEMOS<CR>
$


$ rm -ri notes<CR>
directory notes? y<CR>
notes/agenda: ? y<CR>
notes/contacts: ? n<CR>
notes/form: ? n<CR>
notes: ? y<CR>
rmdir: notes not empty
$
```



COPYING, MOVING, AND RENAMING FILES

cp *file(s) target*
mv *file(s) target*

- **cp**
  - — Copies regular file(s).
  - — Cannot be used to copy a directory.
- **mv**
  - — Moves regular file(s).
  - — Cannot be used to move a directory.
  - — Renames regular file or directory.

_____

target:

— Can be regular file or directory.

— If directory, file(s) copied/moved with same name.

— Must be directory if copy/move more than one file.

Directories cannot be copied with **cp**:

```
$ cp misc stuff<CR>
cp: <misc> directory        ← you will receive this error message.
$
```

Directories can ONLY be renamed with the **mv** command; they cannot be moved to another directory on the tree.

```
$ mv misc stuff<CR>
$ ls -Cp<CR>
budget/   friends   letters/   poem     stuff/   work/
calendar  fun/      parts      project/  why.unix
$
```

Notice that the *misc* directory is now called *stuff*. If **mv** is used to try to move a directory, the following error message is displayed and the directory is not moved:

```
$ mv fun misc/funfiles<CR>
mv: directory rename only
$
```

COPYING OR MOVING DIRECTORIES

So how are directories and all the files below copied? Follow these steps:

Step 1: Create the new directory (the directory to copy files to) with **mkdir**.

Step 2: Change to the directory that is to be copied or moved.

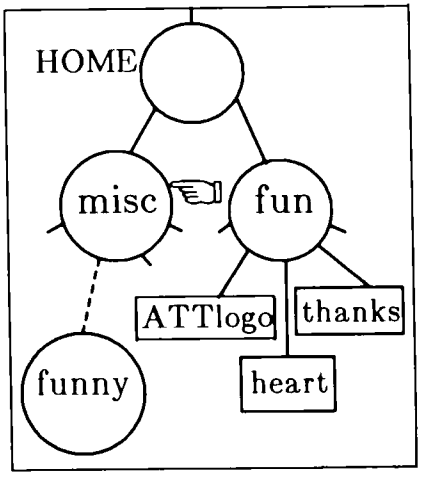
Step 3: Execute: **$ find . -print | cpio -pvd** *dir*<CR>

The **find** command above lists all the path names of the files under the current directory (which is the directory to be copied or moved). This list of path names is passed to the **cpio** command using a pipeline. The **-p** option allows path names to be passed to **cpio** and those files to be copied to *dir*, which must be the path name to the new directory. The **-v** option, for verbose, lists the names of the copied files so you can see what's happening. The **-d** option creates directories as needed, in case there are subdirectories to be copied.

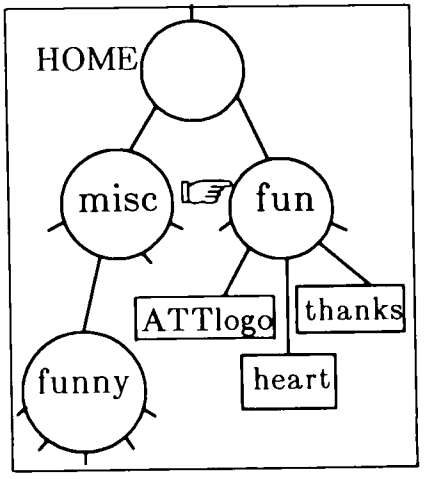
To move a directory, follow steps 1-3. Then remove the old directory. To remove the old directory, change to its parent and recursively remove (**rm** with the **-r** option) the old directory.

COPYING OR MOVING DIRECTORIES

```
$ mkdir funny<CR>
$ cd ../fun<CR>
$
```



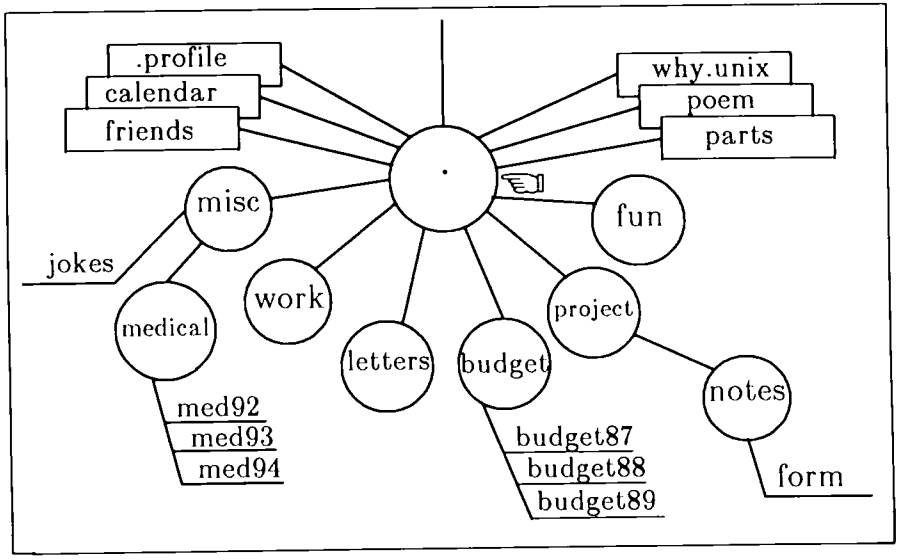

```
$ find . -print | cpio -pvd ../misc/funny<CR>
../misc/funny/ATTlogo
../misc/funny/twilight.unix
../misc/funny/thanks
../misc/funny/dictionary
../misc/funny/heart
30 blocks
$
```




```
$ cd ..<CR>
$ rm -r fun<CR>
$
```

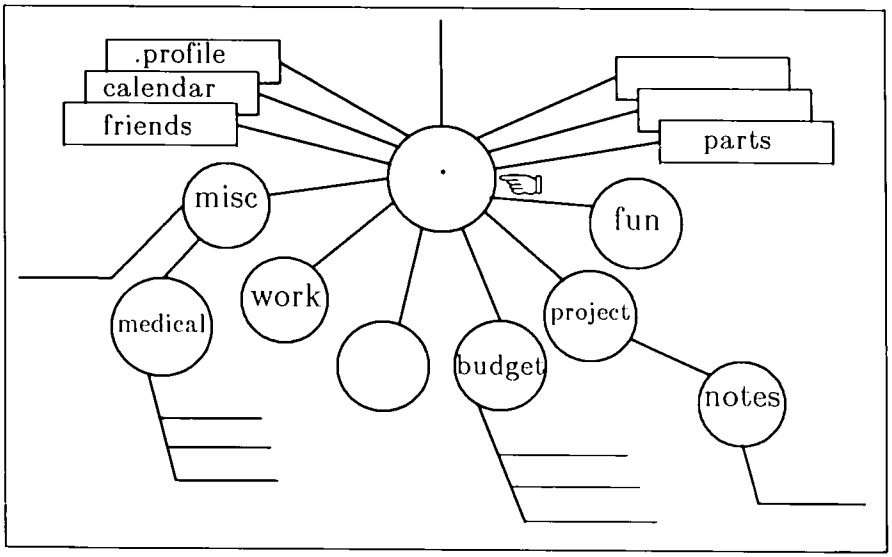
VOLUME 3 EXERCISE

How do these commands change this picture?



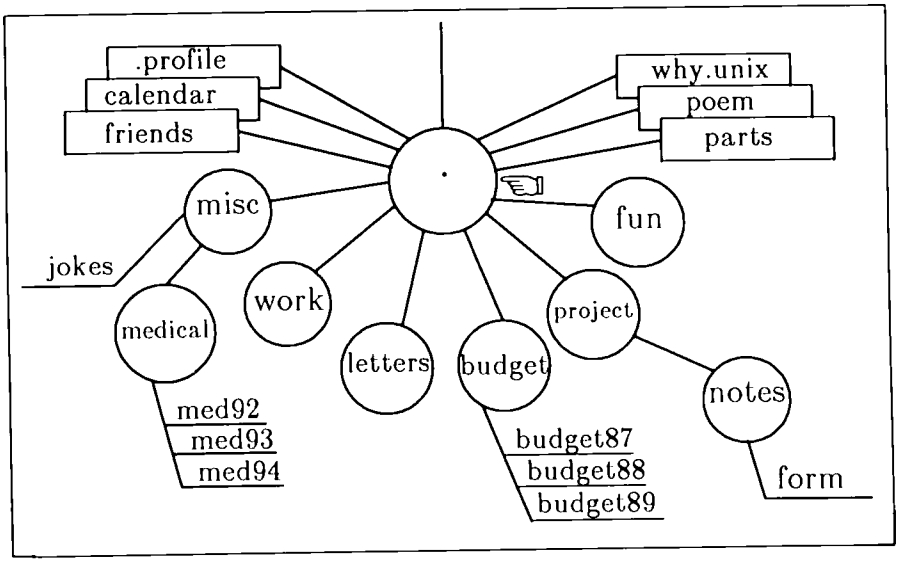
1. **$ cp .profile old.profile**
2. **$ mv poem fpoem**
3. **$ cp project/notes/form**
4. **$ mv why.unix misc**
5. **$ cp calendar misc/reminders**
6. **$ mv misc/jokes riddles**
7. **$ mv letters oldletters**
8. **$ mv misc/medical/med9* misc**
9. **$ cp budget/budget?? project/notes**

WORK SHEET



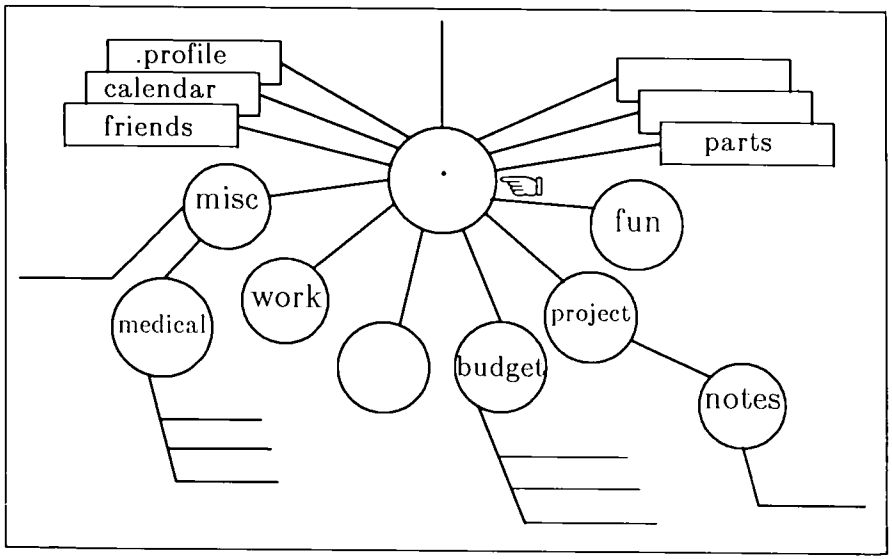
VOLUME 3 EXERCISE

How do these commands change this picture?

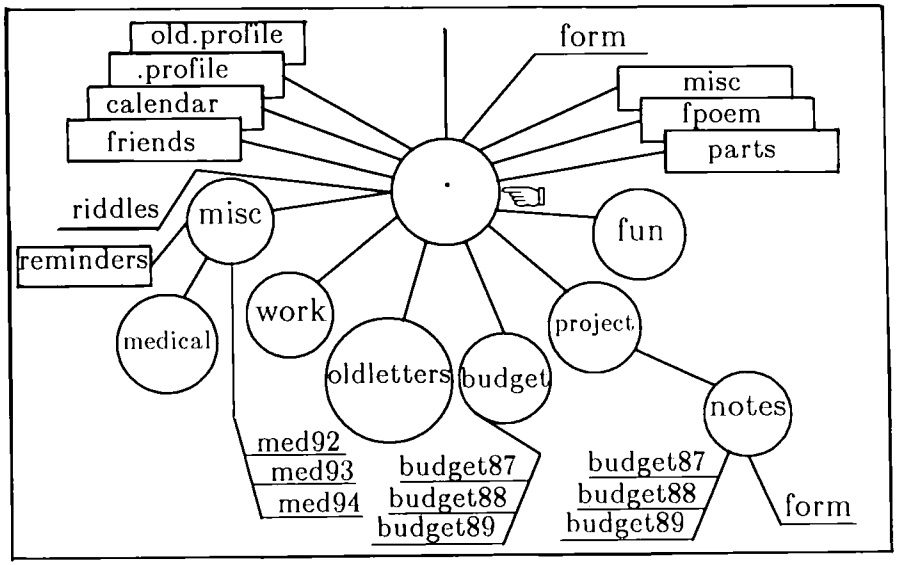


1. **$ cp .profile old.profile**
2. **$ mv poem fpoem**
3. **$ cp project/notes/form**
4. **$ mv why.unix misc**
5. **$ cp calendar misc/reminders**
6. **$ mv misc/jokes riddles**
7. **$ mv letters oldletters**
8. **$ mv misc/medical/med9* misc**
9. **$ cp budget/budget?? project/notes**

WORK SHEET



VOLUME 3 EXERCISE - ANSWERS



VOLUME 3 SUMMARY
UNIX File System

Command	Description	Example
pwd	Prints full path name of the current directory	$ pwd<CR> /a1/sda/new06
ls	Lists files names in columns	$ ls -C<CR> friends ... poem
	Lists the names of files. Directories are followed by a /	$ ls -p<CR> notes plan/ screeninfo/
	Long listing.	$ ls -l<CR> -rw-r--r-- ... notes drwxr-x-- ... plan
	Recursive - Lists the contents of all subdirectories	$ ls -R memo<CR> ode poem memo/ode: appendix
cd	Changes directory	$ cd memo<CR>
file	Reports on type of file(s)	$ file memo<CR> memo: directory
find	Prints path(s) of file(s) matching a given name in the directory subtree	$ find . -name poem -print<CR> ./poem ./misc/poem
	Prints path(s) of all files in the directory subtree	$ find . -print<CR>

VOLUME 3 SUMMARY

Path Name	Definition	Example
fully qualified	A fully qualified path name describes the location of a file from the root.	/usr/games /etc/passwd
relative	A relative path name describes the describes the location of a file from "where you are" (your current directory)	project/notes ../misc/stew

ORGANIZING FILES		
Command	Description	Example
mkdir	Makes a directory.	$ mkdir documents<CR>
cp	Copies regular files to same or different directory.	$ cp poem newpoem<CR>
mv	Renames files. Moves regular files to another directory.	$ mv poem misc<CR>
rm	Removes regular files.	$ rm poem<CR>
	Removes directory and all files in the directory.	$ rm -r misc<CR>
	Removes files interactively.	$ rm -ri misc<CR>
rmdir	Removes an empty directory.	$ rmdir documents<CR>
cpio	Copies a directory.	$ find . -print ¦ cpio -pvd *newdir*

VOLUME 4

The vi Screen Editor

WHAT YOU'LL BE ABLE TO DO AFTER VOLUME 4

- Entering and exiting vi
- General characteristics
- Creating text
- Cursor movement
- Deleting text

WHAT IS AN EDITOR?

- A program used to *edit* (create and modify) files
- Line- or screen-oriented

ed - original UNIX System line editor

vi* - UNIX System full screen editor

ex* - line-oriented version of **vi**

---

* The visual editor is based on software developed by The University of California, Berkeley, California; Computer Science Division, Department of Electrical Engineering and Computer Science, and such software is owned and licensed by the Regents of the University of California.

EDITING A FILE

TERM is used by the **vi** editor, **pg**, and other commands that control cursor motion and other terminal attributes. TERM must be set to the type terminal you are using such as **vt100, 5420,** etc. If you can't recognize your terminal type, ask the system administrator.

If, for example, your terminal type is 5425, before using vi you should do these two steps:

```
TERM=5425
export TERM
```

EDITING A FILE

A text file is a named collection of ASCII characters. These characters may comprise a memorandum, a computer program, data, etc.

Like most editors, **vi** does not edit an actual disk file, but rather a copy of the file placed in a temporary storage area in the computer's memory called a *buffer*. **vi** editing commands affect only the buffer contents and not the disk file itself. However, there are commands that can be invoked from within **vi** to write the buffer contents to a permanent file.

Because of this buffering scheme, there is a limit to the size of a file that can be edited with **vi**. This limit is dependent on your computer and what version* of **vi** you are using. Typical limits (for 3B computers) are:

- Line length: maximum 1023 characters
- File size: approximately 225,000 characters

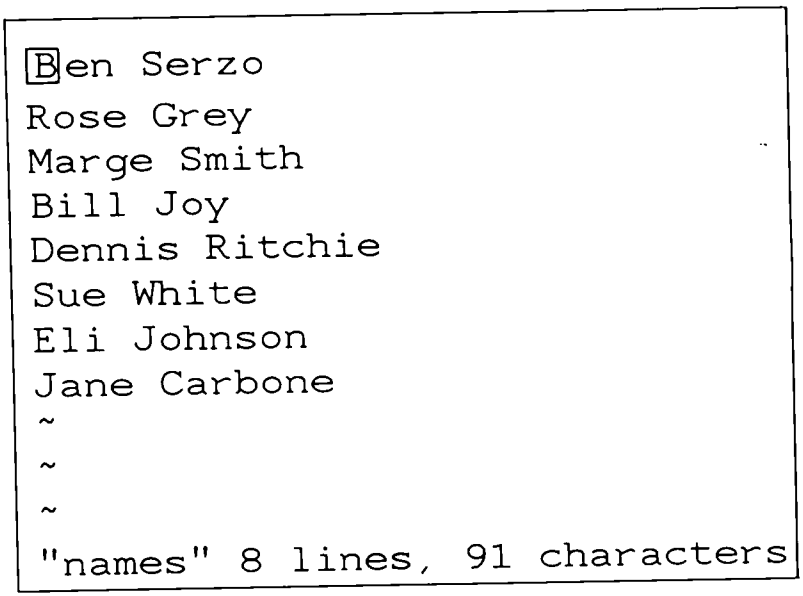
Ask your system administrator for the limits in effect on your machine. If necessary, you can use the UNIX System **split** command (documented in Section 1 of the UNIX System V User Reference Manual) to subdivide large files.

---

* Like most large software packages, **vi** has been re-released a number of times to include new features and improvements, etc. Older versions of the UNIX System may be running older versions of **vi**.

INVOKING vi
YOUR SCREEN ON ENTERING

$ vi names<CR>



- *window* — size depends on baud
- *cursor position* — upper left corner of buffer
- *entry message* — file name and size
- ~ *lines* — beyond end of file

USER TO vi DIALOG

When you "talk" to the **vi** editor, you are in either *command mode* or *text input mode*.

Command Mode:

Command mode is used to enter **vi** commands. You are always in command mode on beginning your **vi** editing session.

In contrast to commands invoked at the shell prompt, many **vi** commands do not appear on the screen as you type them in and do not end with <**CR**>. There are, however, some commands, such as those that perform file manipulation, pattern searching and substitution, setting options, etc., that are displayed at the bottom of the screen as you type them in and end with a <CR>. Most of the commands that end with a <CR> are also preceded with a colon (:).

To cancel a partially typed command, press <**ESC**>.

If you enter an invalid command you may hear a bell, your screen may flash, or you'll receive some other terminal-dependent indication.

Text Input Mode:

Text input mode is used to type new text directly into the buffer. To enter text input mode, you must type one of the input commands, which will be discussed shortly. To end text input mode, press <**ESC**>, which always returns you to command mode.

ADDING TEXT

Step 1:	a	append after cursor
	i	insert before cursor
	o	open new line below cursor
	O	open new line above cursor
Step 2		type new text
Step 3:	<ESC>	return to command mode

Note:	<BS>	erases character in input mode
	@	erases line in input mode

MOVING THE CURSOR

Movement	Command
right	→ <SP> l
left	← <BS> h
up	↑ k
down	↓ j
beginning of next line	<CR>
beginning of previous line	–
last line	G
line *n*	*n*G
end of line	$
beginning of line	^
forward beginning of word	w*n*wW*n*W
end of word	e*n*eE*n*E
back beginning of word	b*n*bB*n*B

DISPLAYING BUFFER CONTENTS

Scrolling Down:

The <**CTRL d**> command will scroll down in the buffer, displaying on the screen the lines that are immediately below the current window. That is, the screen display moves to show more lines below.

Scrolling Up:

<**CTRL u**> scrolls up in the buffer, showing the lines immediately above the current window. The default number of lines scrolled up or down is about half your window size.

Paging Forward:

When editing a large file, you may wish to move ahead in the buffer a screenful at a time. The <**CTRL f**> command will "repaint" your screen with the next screen-sized "page." The last line of the previous screen will be repeated at the top of the screen for continuity.

Paging Backward:

<**CTRL b**> works like <**CTRL f**>, but in the reverse direction.

DELETING TEXT

Deleting a Character:

The **x** command will delete the character at the current cursor position. "x" is mnemonic for the operation of "crossing out" characters on a typewriter.

Deleting Multiple Characters:

Multiple characters may be deleted by either repeating the "x" as many times as necessary, or by prefixing it with a number.

Replacing a Character:

The **r***c* will replace the character under the cursor with the character entered.

Deleting Lines:

The **dd** command will delete the current line (the line on which your cursor is). Multiple lines may be deleted by prefixing the command with a number. For example, **4dd** will delete the current line and the 3 lines below.

DELETING TEXT

Deleting Text Objects:

The *d* (delete) command may be followed by a text object. The effect of the resulting command is to delete the characters starting at the current cursor position up to, and possibly including, the specified text object.

Some combinations and their effects are listed in the following table:

COMMAND	MEANING
d$	Delete remainder of line
dw	Delete up to beginning of next word
de	Delete remainder of word
db	Delete backwards through beginning of word
dL	Delete through last line on screen
dG	Delete through last line in buffer

Undoing Changes to Buffer:

The **u** command will undo the effects of the last command that changed the contents of the buffer. ANY command that changed the contents of the buffer can be "undone." For example, the **u** command can undo the effects of the **d**, **a**, **i**, and **o** commands. It can be "toggled," allowing you to alternate between two versions of the buffer contents.

If you have made multiple changes to a line and have not moved the cursor from it, the original contents of the line can be restored with an upper case **U**.

EXITING vi AND SAVING THE BUFFER

:wq<CR>	write and quit (return to $)
:w<CR>	write
:w file<CR>	write to another file
:w! file<CR>	overwrite existing file
:q<CR> **:q!**<CR>	quit without writing

Commands following a : are echoed at bottom of screen

VOLUME 4 EXERCISE

1. What is vi?

2. How can we create a new file with vi?

3. How can we delete six lines from our current line?

4. What is the command to restore the last change in vi?

5. How do you quit vi without saving your changes?

6. How can you write to another file while in the buffer?

VOLUME 4 EXERCISE - ANSWERS

1. Page 4-3
2. page 4-6
3. Page 4-11
4. Page 4-12
5. Page 4-13
6. Page 4-13

VOLUME 4 SUMMARY
vi Screen Editor

---

1. *Invoking and exiting* **vi**

$ vi file	invoke vi
:wq	write and quit
:w	write
:w file	write to file
:w! file	overwrite existing file
:q	quit
:q!	unconditional quit

2. *Display Text*

<CTRL d>	scroll down
<CTRL u>	scroll up
<CTRL f>	page forward
<CTRL b>	page back

VOLUME 4 SUMMARY
vi Screen Editor

3. *Cursor Movement*

→ <SP> l	next character
← <BS> h	previous character
↓ j	character below
↑ k	character above
<CR>	
-	beginning of previous line
G	GOTO last line
*n*G	GOTO line *n*
$	end of line
^	beginning of line
w *n*w W *n*W	forward beginning of word
e *n*e E *n*E	end of word
b nb B *n*B	*back beginning of word*

4. *Text Creation*

atext<ESC>	append after cursor
itext<ESC>	insert before cursor
otext<ESC>	open line below
Otext<CR>	open line above

5. *Delete Text*

x	delete character
*n*x	delete *n* characters
r*c*	replace character
dd	delete current line
*n*dd	delete *n* lines
dw	delete word
d$	delete to end of line
u	undo last editing command

6. *Variables*

```
TERM=type
EXINIT="set opt1 opt2 ..."
export TERM EXINIT
```