Beginning ConvexOS Training Course

Version 5.0

June 19, 1990

CONVEX Computer Corporation

Table of Contents

Appendices

List of Tables

Preface

Objectives and Intended Audience

The ConvexOS introductory training materials address technical professionals having no prior experience with the UNIX operating system. This document, which is intended as a training aid, contains a copy of the instructor's transparency presentation and examples of the major points. In addition, concepts/functions are reinforced with exercises.

In general, the materials introduce the major ConvexOS concepts. Simplified command formats are included—logging on and off, password management, terminal input/output, communication with other users and operation on files. These materials encompass the ConvexOS hierarchical file system—files and their use, directories, and file organization. In addition, the user is introduced to the *vi* text editor. The materials also include customization of the user's working environment.

Organization

- Chapters 1 provides general information on the ConvexOS operating system and getting started on the system.
- Chapter 2 introduces the user to the on-line help features. It also provides information about the on-line tutorial program.
- Chapter 3 covers file management. Topics include making directories, copying and moving files and directories, as well as setting permission bits.
- Chapter 4 is a self-paced tutorial on the *vi* editor.
- Chapter 5 introduces information-oriented commands.
- Chapter 6 covers the *csh* features - using *history*, creating aliases, and controlling jobs.
- Chapter 7 contains insturctions for using electronic mail.
- Chapter 8 presents the use of redirection.
- Chapter 9 covers customizing the ConvexOS environment.
- Chapter 10 illustrates how to use the notes system.
- The appendix contains a summary of ConvexOS commands and illustrates basic commands needed to get started using the *vi* editor.

Notational Conventions

The following conventions are used in this document:

- *Italics* within text indicate commands, filenames, or programs.

- Within command sequences and text, `typewriter` type indicates literals. Text examples appearing inside of screens will have literals in **boldface** type. Words appearing in `typewriter` font on transparencies should be typed just as they appear. *Italics* within command sequences indicate generic commands or filenames. Substitute actual commands or filenames for the *italicized* words. For example, the command sequence:

  `ld` [*switches*] [*object files*] [*libraries*]

  instructs you to type the command *ld*, followed by your choice of switches, object files, and/or libraries.

- For text appearing in screen examples `typewriter` font represents the text as it appears on the terminal.

Associated Documents

The following documents, available from CONVEX Computer Corporation, are recommended to the ConvexOS user:

- *ConvexOS Primer* describes and illustrates the use of simple UNIX commands.
- *UNIX Tutorial Papers* contains a section "UNIX for Beginners—Second Edition", which discusses the basic UNIX commands.

1

Getting Started with ConvexOS

Objectives for Chapter 1

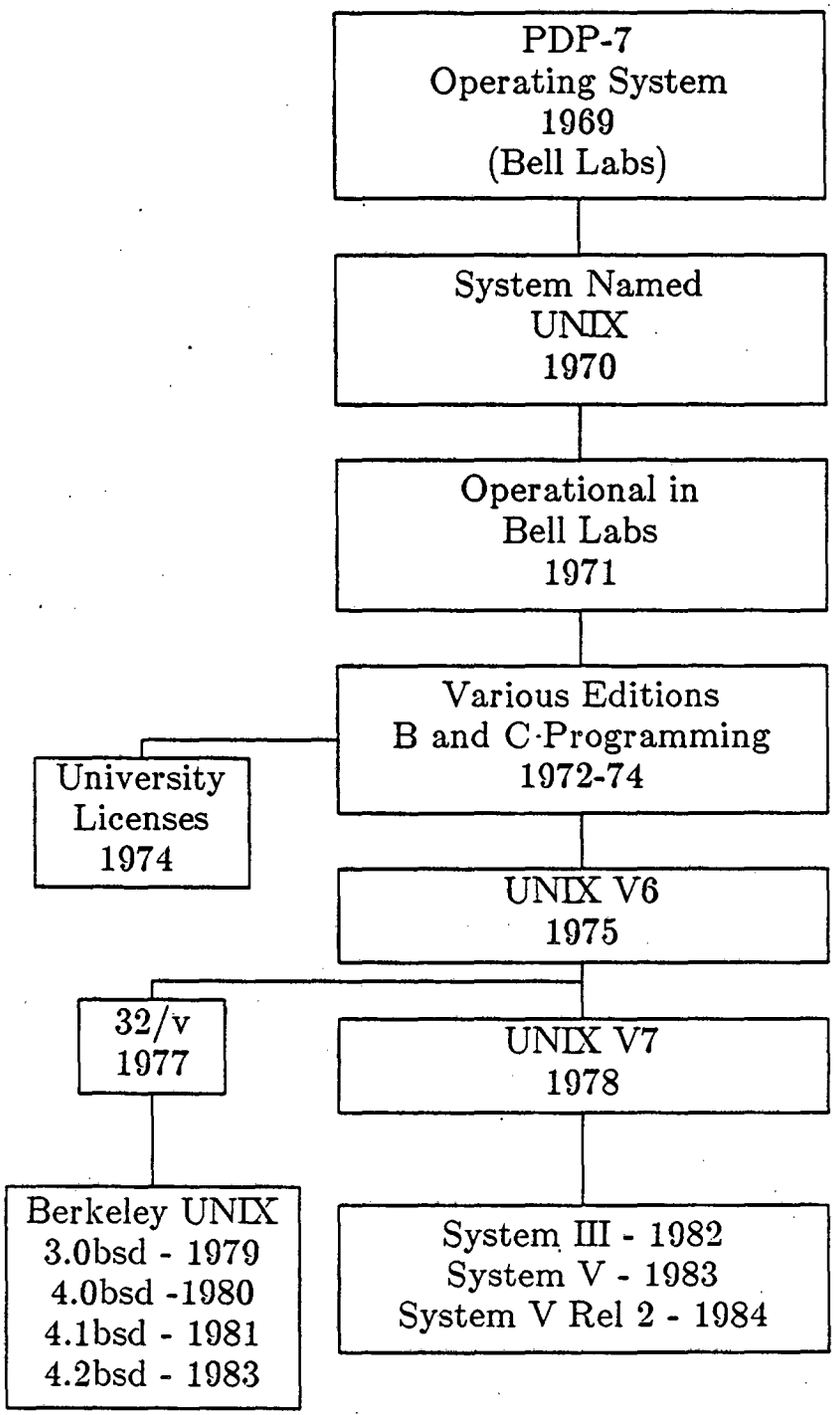
After completing this chapter, you will be able to:

1. Give a brief description of the development of Berkeley UNIX.
2. List three benefits/features of ConvexOS.
3. Be able to log on.
4. Be able to change your personal password.
5. Be able to enter a command in the correct format.
6. Be able to correct typing errors when entering a command.
7. Use control commands to suspend and resume terminal output.
8. Use control commands to interrupt (abort) commands.
9. Be able to log off.

UNIX HISTORY

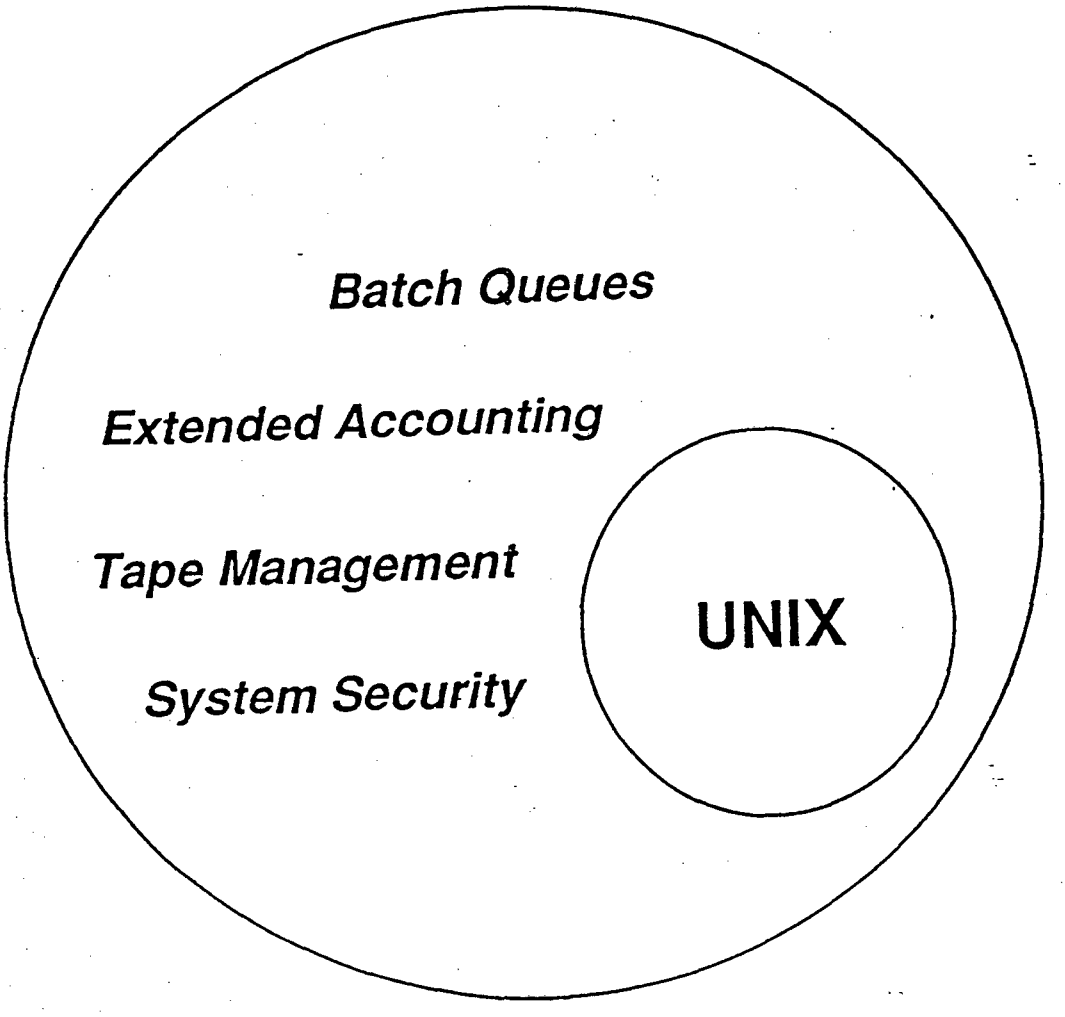
- Bell Labs Research
  - Version 6 - 1975
  - Version 7 - 1978
- AT&T UNIX Support Group (USG)
  - System III - 1982
  - System V - 1983
  - System V Rel 2 - 1984
- University of California at Berkeley (UCB)
  - 3.0bsd - 1979
  - 4.0bsd - 1980
  - 4.1bsd - 1981
  - 4.2bsd - 1983

UNIX System Genealogy



MANAGING MANY USERS

- Berkeley UNIX enhanced for a large timesharing environment



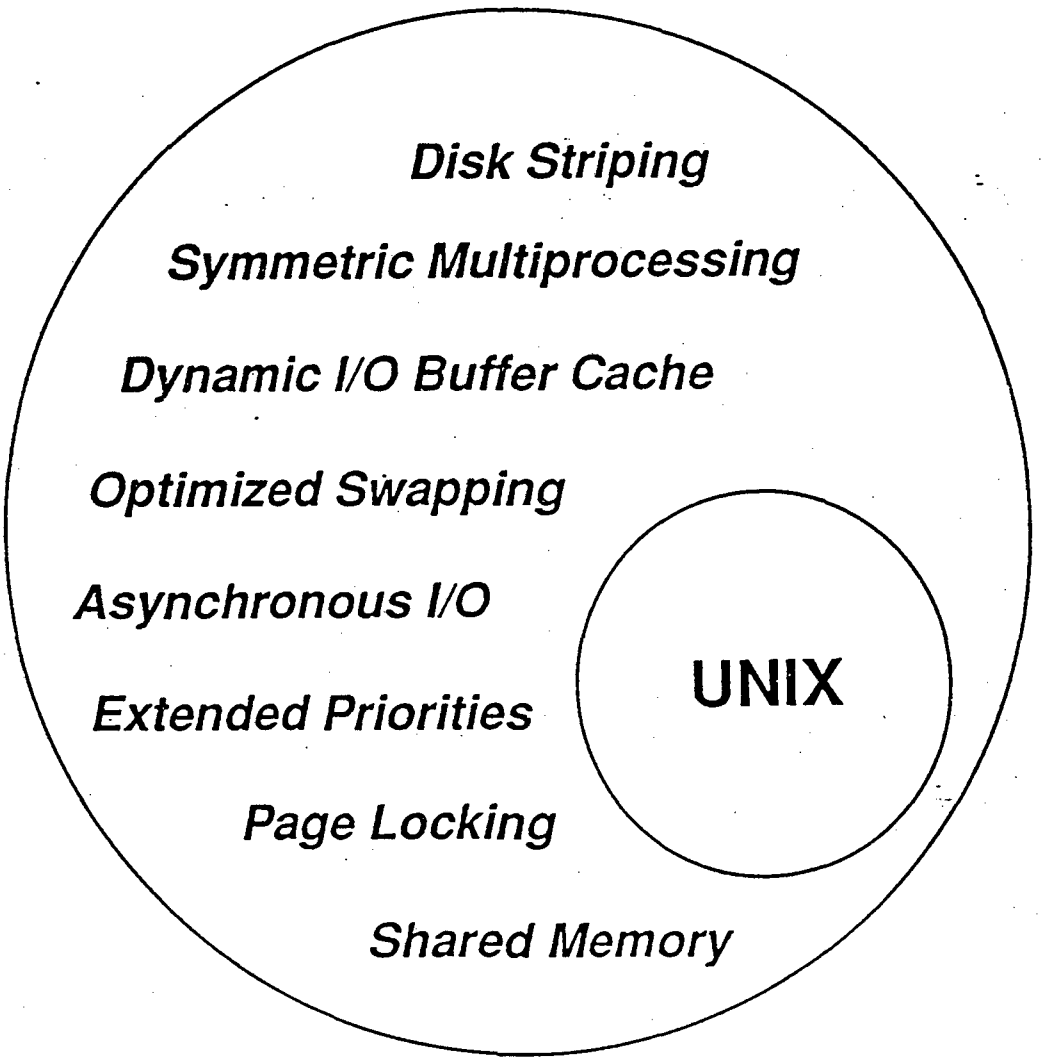
Timesharing Environment

To support large environments:

- The CXbatch management system allows queued execution of resource intensive applications.
- The accounting system supports accounting for resources and billing on a per-job basis or on a per-user basis.
- A tape allocation system with an operator interface queues tape requests and satisfies them under operator control.
- Security enhancements provide better tracking of attempted security violations, allowing the isolation of potential security problems.

SYSTEM PERFORMANCE

- Berkeley UNIX enhanced for an overhead free environment.

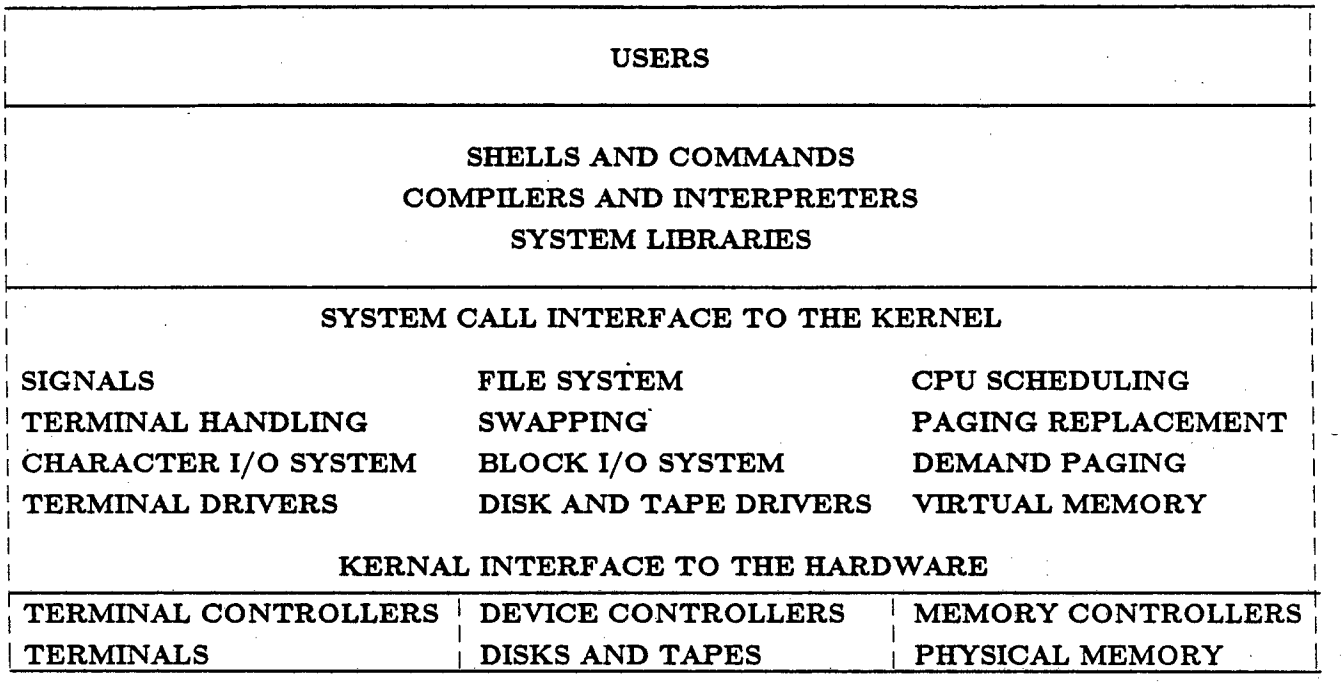


System Performance

The system is extensively tuned to minimize overhead and reduce bottlenecks.

- Important system data structures have locks associated with them to prevent simultaneous modification by multiple processors.
- The memory system is designed around a physical page manager that dynamically uses pages as either virtual memory for programs or as file system buffers for data caching. Options are provided to allows a program to be locked permanently into memory and for programs to share physical memory pages.
- Disk striping allows a single file system to be spread across multiple physical disk drives, allowing the creation of files larger than a single disk drive and providing increased performance on I/O intensive jobs by spreading individual writes across multiple drives.
- The swapping algorithm is significantly redesigned from the standard Berkeley model to more efficiently deal with very large programs unlikely to be executed on any system but a true supercomputer.

ConvexOS SYSTEM OVERVIEW



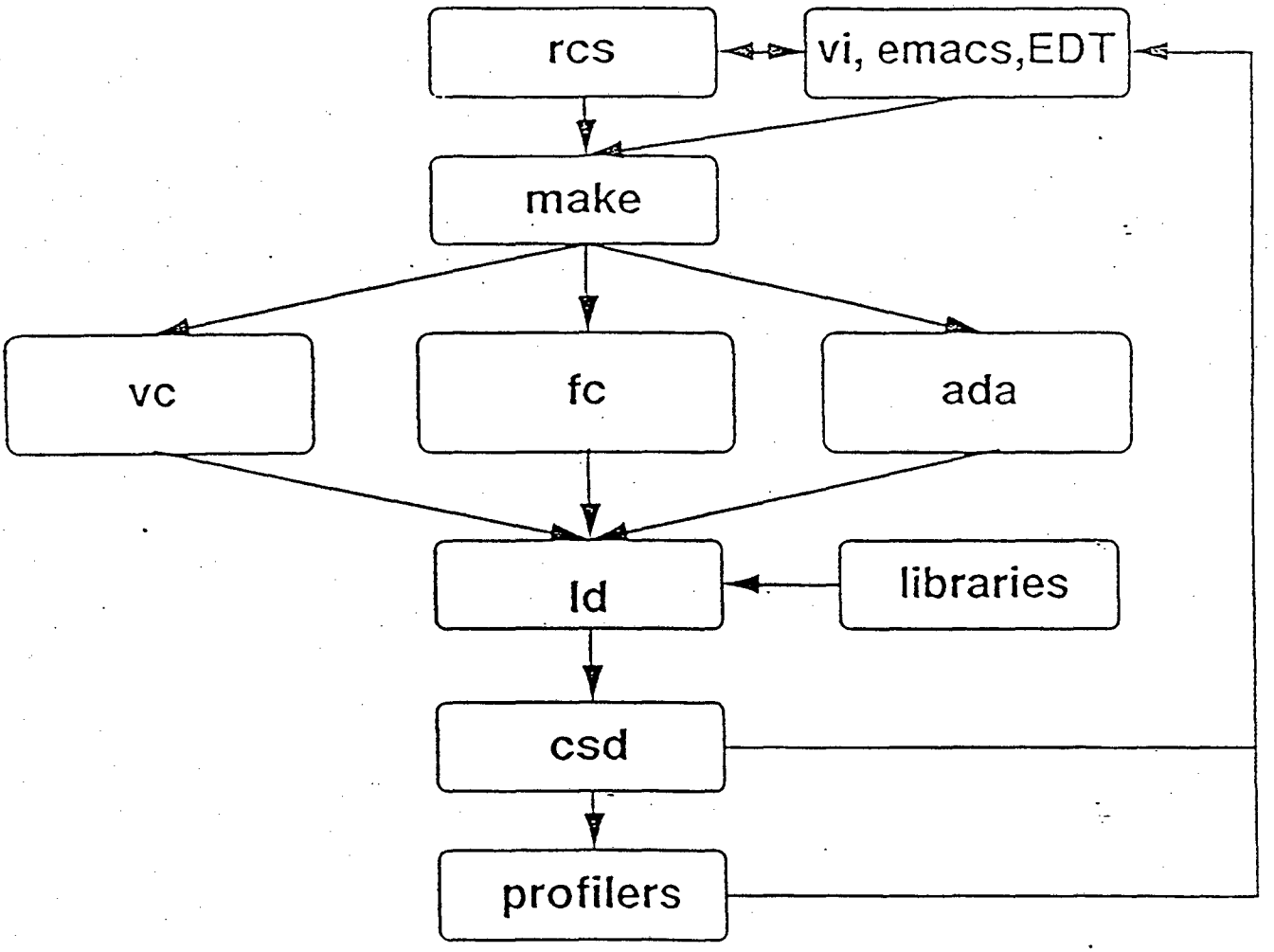
System Overview

CONVEX supports, on one base, two of the most popular user environments available today. It retains all the features that make Berkeley UNIX one of the most popular operating systems for scientific programmers. Both the Bourne and C shells are supported. The programmability of the *shells* and their built-in commands give programmers flexibility and power.

Additionally, for users more familiar with the VMS operating system from DEC, CONVEX has developed the COVUE user environment - EDT compatible editor, a DCL compatible command interpreter, and DECNet networking capabilities.

- Utilities - user-support programs (or commands) are used to make the computer carry out frequently needed tasks
- Shell - serves as interface between the kernel and user-programs
- Operating System (or kernel) - manages all of the machines' resources
- System Calls - allow interaction between the kernel and user programs
- Subroutine Libraries - supports program development
- Special Device Files - allow flow of data between the computer and peripheral devices

INTERACTIVE PROGRAMMING ENVIRONMENT



Interactive Programming Tools

The system provides a programming environment that makes it easy to manage large programming projects, debug code, and optimize and tune an application.

- ConvexOS provides a fully integrated, interactive programming environment incorporating a variety of familiar editors.
- ConvexOS provides the Revision Control System (RCS), which manages software libraries. RCS automates storing, retrieving, logging, identifying, and merging of revisions.
- ConvexOS supports major languages such as FORTRAN, ADA, and C.
- ConvexOS provides *make*, a program for maintaining computer programs. *make* automatically keeps track of files that have changed and recompiles them if required.
- Optional products are provided, such as the VECLIB library of highly optimized routines which contains commonly used routines tuned by CONVEX numerical analysts to run optimally on the C family.
- The CONVEX Consultant provides symbolic debugging and profiling tools, to speed location and correction of programs bugs, and to identify code *hot spots* as candidates for further optimization.

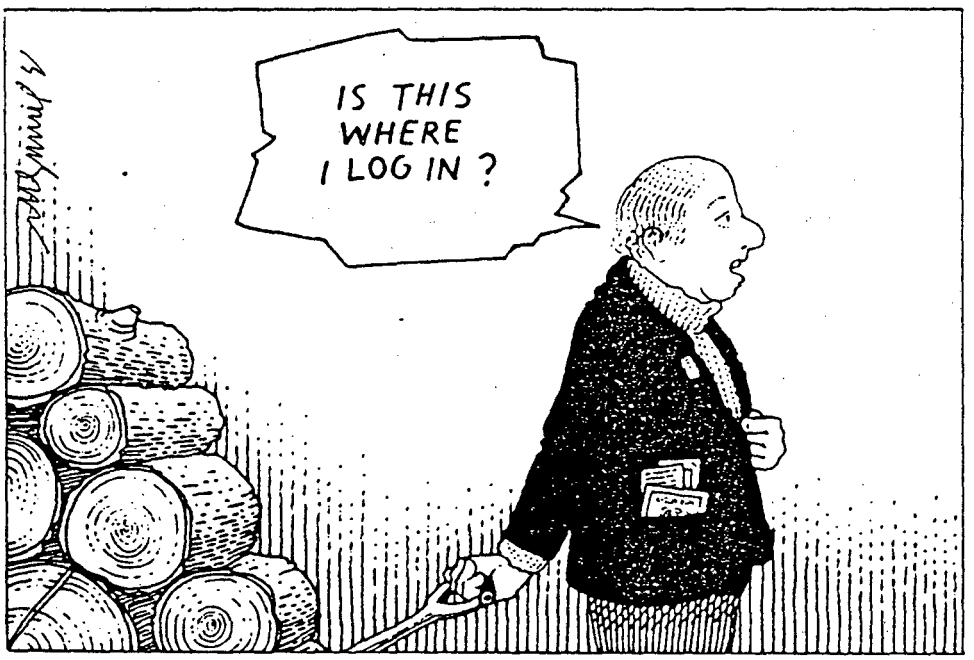
LOGGING IN

Login ID:

- Assigned by system manager
- Is usually your username
- Entered at system login prompt to access the system

Valid login ID's:

```
tsmith
joe
root
a301
```



Accessing the System

If your terminal is not on, turn it on (the switch is on the back). If your screen is blank, press the RETURN key to display the system prompt *login:*. Most likely your screen will display the name of your system and the system prompt. However, if you have several systems, a list of the systems may be displayed.

```
Convex Training/External Machine 4.0 Unix (convext)

login:
```

At the system prompt, *login:*, enter your login ID (username). (Be sure to press RETURN so the system knows that you are finished.)

```
Convex Training/External Machine 4.0 Unix (convext)

login: username
password:
```

LOGGING IN (cont.)

Password:

- Provides security with secret password
- Create password with a minimum of 6 characters; only the first 8 characters are checked
- Use a combination of uppercase and lowercase characters, digits and special characters for your password
  - `q29o4790`
  - `Salu9brious`
  - `vzc5!fads`
  - `vxCxfAds`
- Don't use passwords that are easy to discover or guess:
  - `xxxxxx -Too easy to witness`
  - `twjones -no names or initials`
  - `9520333 -no phone numbers`
  - `tsmith -don't use your username`
- Change your password frequently with the *passwd* command

Entering Your Password

After you have entered your login ID, you will be prompted for your password. (If the account does not have a password assigned to it, the system will log you in without asking for a password.) Enter your password; it will not display (echo) on the screen when you enter it. After you correctly enter your username and password, the following is displayed:

```
login: username
password:
Last login: Mon Aug 5 13:30:29 on tty2
Additional lines may appear here.
%
```

The % prompt ($ for Bourne shell) indicates that you have gained access to the ConvexOS system, and the C shell is ready to accept commands.

If the following is displayed:

```
login: username
password:
login incorrect.
login:
```

either you made an error entering your username or password, or you have no account on the system. Retype your username and password.

Changing Your Password

To change your user password, enter:

passwd

You will be prompted for your current password and a new password. Your are prompted twice for your new password to insure correctness. (The password does not echo to the screen.) Your session looks similar to:

```
% passwd
Changing password for username
Old password:
New password:
Retype new password:
%
```

If you don't remember the old password or if you mistype it, the system refuses to allow you to alter the password.

COMMAND STRUCTURES

Commands:

- Tell the system what to do
- Are usually entered in lowercase
- Use the format:

  command [options(s)] [filenames(s)]

  The *options*:

  - modify the way a command works
  - are often single letters prefixed by a dash
  - can usually be combined and prefixed by one dash

  The *filename(s)*:

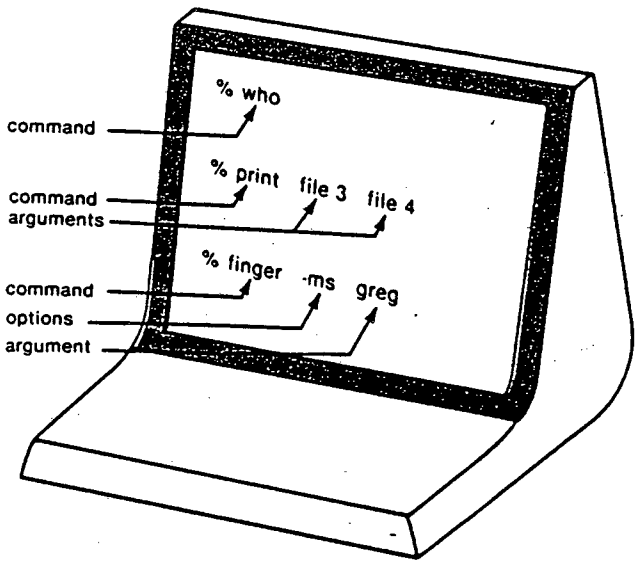
  - specifies file(s) to be used

- Use the semicolon (;) to separate multiple commands on the same line.
- Continue commands to the next line by placing a backslash (\) at the end of the line.

Command Structures

Sample command formats:

`lf`	Command only
`date`	Command only
`lf -la`	Command with two options
`lf -l .login`	Command with an option and an argument
`cal 1986`	Command with one argument
`cal 8 1986`	Command with two arguments
`cal 81986`	Unrecognized command; no spaces
`cal 8\` `1986`	Command continued on another line
`date; cal 8 1986`	Multiple commands separated with semicolon (;)



BASIC COMMANDS

`date`	displays date and time
`whoami`	displays who you are
`cal` *[month] year*	displays calendar for year specified
`cat` *filename*	displays contents of file
`lf`	list contents of current directory

Basic Commands

Tryout the *date* command by entering:

date

The screen displays:

```
% date
Tue Aug  5 15:09:09 CDT 1986
%
```

Display the date and a calendar for August by typing:

date; cal 8 1986

Both the date and calendar are displayed:

```
% date; cal 8 1986
Tue Aug  5 15:09:09 CDT 1986
   August 1986
 S  M Tu  W Th  F  S
                1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
31
%
```

CORRECTING TYPING MISTAKES

Correction keys (before pressing the RETURN key):

- Backspace key or CTRL-h – erases the last character or several characters typed
- CTRL-w – erases the last word you typed
- CTRL-u – erases the entire input line

  Can be changed with the *stty* command in your *~/.login* file

Correcting Typing Errors on the Command Line

Enter the following command incorrectly as shown; do not press the RETURN key.

daet

Use the BACKSPACE key or CTRL-h to erase the *et*; then type the **te**. When you strike the BACKSPACE key or CTRL-h two times, your entry looks like:

```
% daet (incorrect entry)
% da (entry after deletions)
% date (corrected entry)
Tue Aug  5 15:09:09 CDT 1986
%
```

Enter the following command incorrectly as shown; do not press the RETURN key.

cal 8 29866

Use CTRL-w to erase the last word typed; change the year to **1986**. Your entry looks like this:

```
% cal 8 29866  (incorrect entry)
% cal 8  (entry after deleting 29866)
% cal 8 1986 (entry after making correction)
   August 1986
 S M Tu  W Th  F  S
                1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
31
%
```

Enter the following command but do not press the RETURN key.

daet; col 2986

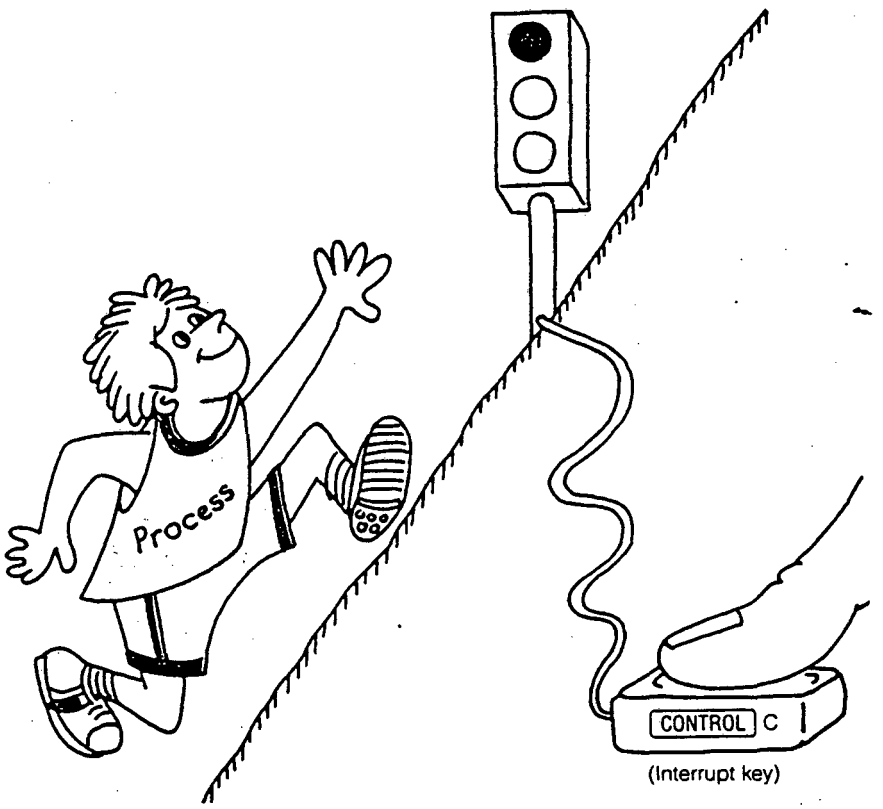
Erase the entire entry with the CTRL-u key. Your entry will look like this:

```
% daet; col 2986 (incorrect entry)
% (deletion leaves a blank line following the prompt)
```

FUNCTION OF CONTROL CHARACTERS

Control characters:

- CTRL-s – suspends output to the terminal
- CTRL-q – resumes output to the terminal
- CTRL-o – discards output to the terminal
- CTRL-c – aborts an executing command or a command as you are typing it



Running Processes Can Receive Signals

Using Control Characters to Control Terminal Output

Suspend and resume the output of the command *lf /etc*. Suspend the output by typing **CTRL-s**; resume the output by typing **CTRL-q**. Your entry looks similar to:

```
% lf /etc
ac            fastboot   mtab         rc          stripecap
accton        fasthalt   mtbfcrunch   rc.local    swapon
activities    fsck       ncheck       rc.uucp     sysgen
CTRL-s SUSPENDS OUTPUT--DOES NOT DISPLAY ON YOUR SCREEN
CTRL-q RESUMES OUTPUT--DOES NOT DISPLAY ON YOUR SCREEN
actwho        fstab   networks   rdump    syslog
actwhocheck   ftpd    newfs      reboot   syslog.conf
%
```

Drop the output of the command *lf /etc* to the terminal using CTRL-o. Your entry:

```
% lf /etc
ac            fastboot mtab        rc       stripecap
accton        fasthalt mtbfcrunch  rc.local swapon
activities    fsck     ncheck      rc.uucp  sysgen
actwho        fstab    networks    rdump    syslog
actwhocheck   ftpd     newfs       reboot   ^O
%
```

Abort the *lf /etc* command before it begins to execute. Your entry looks like this:

```
% lf /etc ^C
%
```

Abort the command *lf /etc* while it is executing using CTRL-c. Your screen looks similar to:

```
% lf /etc
ac            fastboot mtab        rc       stripecap
accton        fasthalt mtbfcrunch  rc.local swapon
activities    fsck     ncheck      rc.uucp  sysgen
actwho        fstab    networks    rdump    syslog
actwhocheck   ^C
%
```

LOGGING OUT

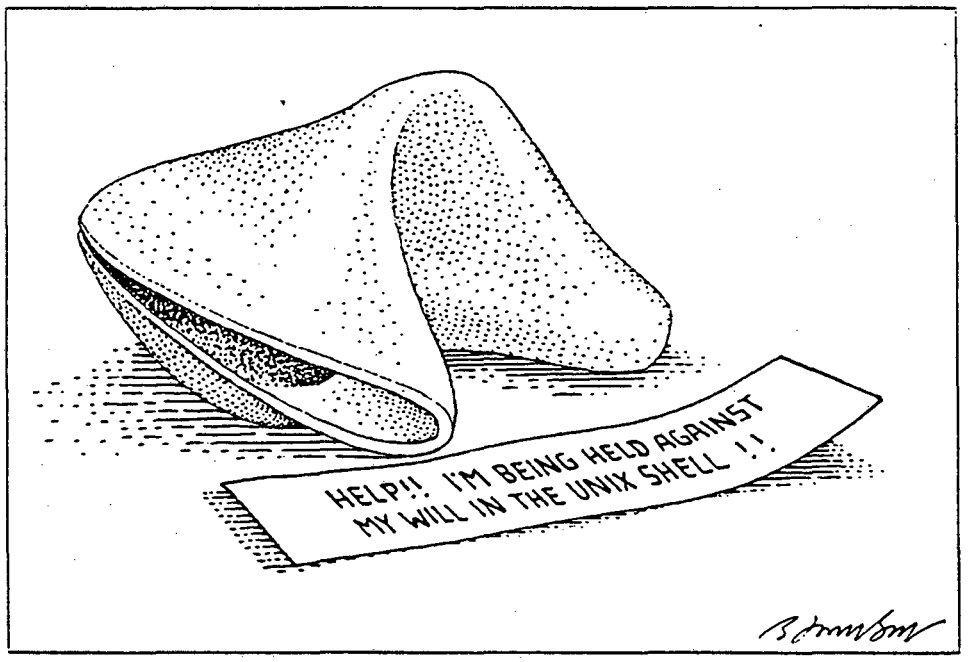
- End a UNIX session by typing *logout* at the system prompt (%).

```
% logout
```

- Your terminal may display one of the following when you attempt to logout:

```
There are stopped jobs.
```

```
Not login shell.
```



Logging Out

Use the *logout* command to logout. The % (*shell* prompt) must be displayed. Your screen will look similar to this after logging out:

```
% logout
Your screen may go blank, display a fortune, or
login:
```

If your screen displays `There are stopped jobs`, type **logout** a second time. (Stopped jobs will be killed.)

```
% logout
There are stopped jobs.
% logout
```

If your screen displays `Not login shell`, type **exit** followed by **logout**.

```
% logout
Not login shell
% exit
% logout
```

PROBLEM/SOLUTIONS

After login, everything displays in capital letters with slashes:

- Logout and login again.

Output to the terminal seems to hang:

- Type CTRL-q.
- Check the NOSCROLL key to be sure it is not toggled on.
- Press the RETURN key.
- Type CTRL-c two or three times.
- Type CTRL-d.
- Check the physical connection between your terminal and its data cable.
- Contact your System Manager if all these steps fail.

Exercises for Chapter 1

1. Who was primarily responsible for the creation of UNIX?

2. List three benefits of ConvexOS.

3. If you are currently logged onto the system, logout. Log on to the system by typing your password in capital letters.

   After you are logged on, type the command **date**. What happens? Can you enter the command in lowercase characters?

   Take corrective action to resume normal keyboard entry.

4. Type **lf /** and abort the command.

5. Type **lf /etc** and suspend its output; then resume the output.

6. Someone has discovered your current password. Change it to one that is not easily discovered.

7. Type **whoiam** but do not press the RETURN key. Correct the entry to read **whoami**; then press the RETURN key.

8. You saw one of your colleagues using the *cat* command. You decide to use it by typing:

   cat /etc

   When the system prompt displays (if it does), type **date**. Can you? Your terminal may seem to be hung. Take corrective action to resume normal keyboard entry.

9. At the system prompt, type **CTRL-d**. What happened? Why?

10. Identify the arguments for the following commands:

    date
    cal 1987
    whoami
    lf /etc
    lf

11. What is the purpose of the *passwd* command?

12. What is the purpose of the *%*?

13. What is the current time?

14. On what day were you born?

15. If your password contain 12 characters, do you need to type all the characters to be able to log on to the system?

16. Was is the minimum number of characters a password must contain?

17. How can you enter two commands on the same line?
18. What is the maximum length for a login ID?
19. Is the login *--patty* valid? Why?

2

On-line Help Features

Objectives for Chapter 2

After completing this chapter, you will be able to:

1. Use appropriate online help function to find information on ConvexOS commands.
2. Access the online tutorial for files and the *vi* editor.
3. Display a description for specified ConvexOS commands.
4. Use the *whatis* command.
5. Determine the pathname of specified files.
6. Use the *which* command.

ONLINE DOCUMENTATION

The *info* command:

- Provides information by topic on ConvexOS
- Provides pointers to written documentation
- Provides online examples of ConvexOS commands
- Has the form:

  % `info`

  - Main menu displays
  - Make selection by number or command name
  - Exit with q

Using *info*

Invoke the *info* system. Select number **6** from the main menu. Then select number **1** on the submenu "CHECK USER, JOB, OR SYSTEM STATUS." Explore the various topics listed on the "CHECK USER STATUS" menu. After viewing several topics, exit *info*.

The first time you execute *info*, you will see four screens of introductory information; then the main menu is displayed. The following example assumes that you have previously viewed the introductory information.

```
%info
                        CONVEX INFO SYSTEM MAIN MENU

 1. Contact other users or machines
 2. Use ConvexOS 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 in your selection and press <RETURN>: 6

CHECK USER, JOB, OR SYSTEM STATUS Submenu of Main Menu

 1. Check user status
 2. Check job. status
 3. Check system status

Please type in your selection and press <RETURN>: 1

CHECK USER STATUS - Check User, Job, or System Status Submenu

 1. Get personal user information
 2. List group memberships for a user
 3. Show the most recent user logins
 4. Show the most recently executed commands
 5. Remind yourself when you need to leave
 6. Display the current directory name
 7. Display your login shell (Bourne or C shell)
 8. Display disk usage and limits
 9. Summarize disk usage
10. List users on the system
11. Print the pathname of the user's terminal
12. Display your environment settings

Please type in your selection and press <RETURN>: q
%
```

ONLINE DOCUMENTATION

The *man* command:

- Displays online documentation about ConvexOS commands
- Has the form:

  man *command*

  % man cat

```
CAT(1)              UNIX Programmer's Manual              CAT(1)

NAME
     cat - catenate and print

SYNOPSIS
     cat [ -u ] [ -n ] [ -s ] [ -v ] file ...

DESCRIPTION
     Cat reads each file in sequence and displays it on the
     standard output.  Thus

          cat file

     displays the file on the standard output, and

          cat file1 file2 >file3

     concatenates the first two files and places the result on
     the third.
```

(more output appears in the actual display)

Displaying Manual Pages

View the manual page for *man*. Your entry:

```
% man man
MAN(1)          UNIX Programmer's Manual              MAN(1)



NAME
   man - find manual information by keywords; print out the
   manual

SYNOPSIS
   man -k
   man -f
   man [ - ] [ -t ] [ section ] title ...

DESCRIPTION
   Man is a program which gives information from the program-
   mers manual.  It can list one-line descriptions of commands
   specified by name, or all commands whose descriptions con-
   tain the specified keyword.  It can also provide online
   access to sections of the printed manual.
...
...
%
```

ONLINE DOCUMENTATION (cont.)

The *man -k* command:

- Displays the page that includes the specified keyword
- Has the form:

man -k *keyword* or apropos *keyword*

% man -k print or apropos print

```
banner (see banner(6))     - print large banner on printer
bigcal (see bigcal(1))     - print out calendar of month
cal (see cal(1))           - print calendar
cat (see cat(1))           - catenate and print
cpr  (see cpr(1))          -  print 'C' files
date (see.date(1))         - print and set the date
```

(more output appears in the actual display)

Accessing Manual Pages

Display a listing from the man pages for the topic *editors*. Your entry:

```
% man -k editor
a.out (see a.out(5))    - CONVEX assembler and link editor output
ed (see ed(1))          - text editor
emacs (see emacs(1))    - a screen editor
ex, edit (see ex(1))    - text editor
ld (see ld(1))          - link editor
sed (see sed(1))        - stream editor
vi, view (see vi(1))    - screen oriented (visual) display editors based on ex
%
```

COMMAND DESCRIPTIONS

The *man -f keyword* command:

- Lists all commands that have *keyword* as an index in the man pages
- May also be invoked as *whatis*
- Prints a one-line description (from table of contents of "man page") for the specified *keyword*
- Prints the "man page" name for the keyword, including its section number
- Has the form:

```
man -f keyword
% man -f print
print (see print(1)) - pr to the line printer
```

An alternative:

```
whatis keyword
% whatis whatis
whatis (see whatis(1)) - describes what a
command is
```

- *keyword* can be:
  - a command
  - a system file
  - a device name
  - a system call

Online Manual Page Command Descriptions

Use the *man -f* command to determine where you can find information on the command *tty* in the "man pages," as well as display a one-line description of the command. Your entry:

```
% man -f tty
tty (see tty(1))        - get terminal name
tty (see tty(4))        - general terminal interface
%
```

Use the *whatis command* to describe *emacs*. Your entry:

```
% whatis emacs
emacs (see emacs(1))  - a screen editor
%
```

LOCATING COMMANDS

The *which* command:

- Locates a program file, aliases, and paths (*csh* only)
- Looks for the file which would have been executed had that name been given as a command
- Searches along the path from the user's current path
- Displays the pathname of a given command
- Has the form:

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

Using the *which*(1) Command

Locate the program file *ex*. Your entry:

```
% which ex
/usr/ucb/ex
%
```

ONLINE DOCUMENTATION

The *learn* command:

- Accesses a tutorial on several topics:
  - files
  - editor
  - vi
  - more files
  - C
- Has the form:

  `% learn`

  - Introductory material displays
  - Specify the topic
  - Work through the sessions
  - Exit with `bye`

Accessing *learn*

Access the online tutorial and specify the topic *vi*. If time permits, you can explore this tutorial; otherwise, exit with **bye**. Your entry will be similar to this:

```
% learn

These are the available courses -
 files
 editor
 vi
 morefiles
 macros
 eqn
 C
If you want more information about the courses,
or if you have never used 'learn' before,
press RETURN; otherwise type the name of
the course you want, followed by RETURN.
vi
If you were in the middle of this subject
and want to start where you left off, type
the last lesson number the computer printed.
If you don't know the number, type in a word
you think might appear in the lesson you want,
and I looks for the first lesson containing it.
To start at the beginning, just hit RETURN.

Press the RETURN key

  This is a sequence of lessons in the use of the text-editing
program called 'vi'.  'Vi' stands for 'Visual' and is really a
screen-oriented mode of the editor 'ex'.  It allows you to create and
change files containing text, programs, or pretty much anything else.
These lessons are due for the most part to Pavel Curtis at Cornell.

  You can run the vi program by using the command
            vi <filename>
where '<filename>' stands for the name of the file you want to create
or change.  In most of these lessons, vi will be called automatically
by 'learn'; you'll usually not need to use the above command until
after you've completed this course.

  If at any time you want to leave the learn program and you have a
percent sign as a prompt, just type 'bye' and learn will quit.  There is
a logical stopping place about 20 minutes into the tutorial which is well
marked.

  Type 'ready' when you're ready to go on to the first lesson.

(Leave the tutorial)
bye
To take up where you left off type "learn vi 1.1a"
Bye.
%
```

GETTING USER INFORMATION

The *finger* command:

- Provides general information about users
- Has the form:

```
finger [ -m -l -s ] [name]
        or
f [ -m -l -s ] [name]
```

```
% f person
Login name: person                  In real life: First Person
Office: 86/485, x489                Home phone: 952-0200
Directory: /doc/person              Shell: /bin/csh
On since Sep  9 08:57:34 on tty27
Project: Never ending....
No Plan.
```

- f -m searches only for login names
- f -l displays long form
- f -s displays a short form:

```
Login    Name           TTY Idle   When      Office     Phone
person   First Person     27       Tue 08:57  86/485     x489
```

- Personal information can be changed with:

```
chfn
```

Getting User Information - the *finger*(1) Command

Use the *finger* command to display information about yourself. Your entry looks similar to this:

```
% f username
Login name: username                    In real life: First Last
Office:                                 Home phone:
Directory: /mnt/username                Shell: /bin/csh
On since Sep 23 10:45:34 on tty11
No Plan.
%
```

Since you have no home number listed, add that to your file. Your entry looks similar to:

```
% chfn
Default values are printed inside of of '[]'.
To accept the default, type <return>.
To have a blank entry, type the word 'none'.

Name [First Last]:
Room number (Exs: 18A or 17B) [86/485]:
Office Phone (Ex: 223) [489]:
Home Phone (Ex: 6610379) [9313159]: 952-0200
%
```

Display your user information to verify that your telephone number has been added. Your entry looks similar to this:

```
% f username
Login name: username                    In real life: First Last
Office:                                 Home phone: 952-0200
Directory: /mnt/username                Shell: /bin/csh
On since Sep 23 10:45:34 on tty11
No Plan.
%
```

GETTING USER INFORMATION (cont.)

The *who* command:

- Lists the people currently logged into the system, terminal being used, log on date, and log on time
- Has the form:

```
% who
        user1   tty03   Sep  8 15:44
        user2   tty05   Sep  8 16:33
        user3   tty06   Sep  8 15:47
```

The *w* command:

- Displays the same information as *who* plus idle time, job (terminal session), current process cpu time expended, what the user is doing
- Has the form:

```
% w

4:43pm up 11 days, 5:23, 44 users, load average: 4.30, 4.29, 4.10
User     tty      login@  idle   JCPU   PCPU  what
user1    tty03    3:44pm    48      4      3  -csh[user1] (csh)
user2    tty05    4:33pm     5      3      2  mail
user3    tty06    3:47pm     1     16      3  mail
```

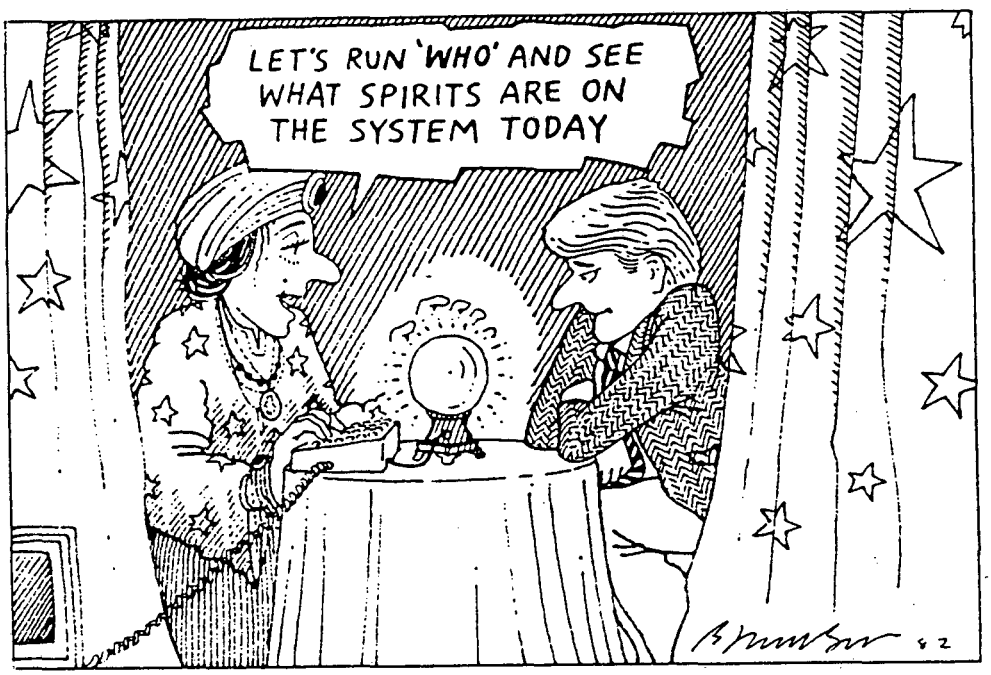
Finding Out who is on the System

Display a list of the users currently logged on to the system. Your entry looks similar to this:

```
% who
user1   tty01   Sep 23 10:16
user2   tty02   Sep 23 10:13
user3   tty04   Sep 23 10:10
user4   tty05   Sep 23 08:09
...
...
%
```

Now determine the time, system uptime and number of users currently logged on to the system. Your entry looks similar to:

```
% w
 11:20am up 7 days, 15:26,  46 users,  load average: 2.72, 2.31, 2.53
User     tty         login@   idle   JCPU   PCPU   what
user5    console      8:26am     7    8:31   1:29   dump 8Guf /dev/rmt20 /usr/spool
user6    tty01       10:16am     1    1:35      7   -csh[user6] (csh)
...
...
%
```



Exercises for Chapter 2

1. List the commands (manual page entries) that give information or are associated with *tty*.

2. What is the difference between the *w* and *who* commands?

3. You want to learn how to use the text editor *vi*. What online documentation is available to provide you information on *vi*?

4. Add a phone extension number to your user information. What command did you use to accomplish this task?

5. You need to know the extension number of user *sloppy*. What command would get you this information?

6. What subjects are covered by learn?

7. How many options are there for the *ls* command?

8. What is the command to find all the commands that are associated with *date*?

9. Using the *info* system, determine how to use *bc*, an online calculator. Write the commands that determine the answer for 5 times the product of 5 divided by 2.

10. On the manual page for *ls*, there is an entry about determining your group. Read the information; then write the command that will display the name of the group(s) you belong to.

11. How many users are currently logged on to your local system? How did you determine the number of users?

12. Determine what each user is currently doing on the system. What command did you use to obtain this information?

3

File Management

Objectives for Chapter 3

After completing this chapter, you will be able to:

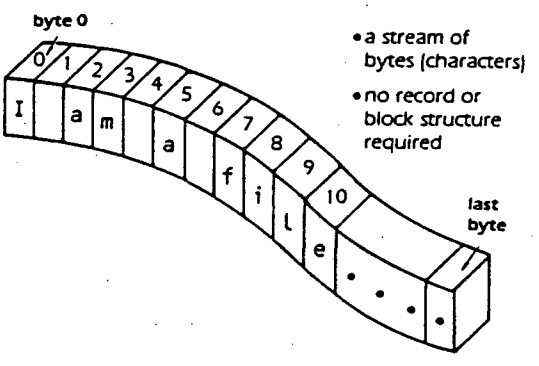
1. Describe the ConvexOS file system.
2. Draw the ConvexOS hierarchical system indicating your directory in the hierarchy.
3. Differentiate between relative and absolute pathnames.
4. Create files with the *copy* command.
5. List files of any given directory.
6. Use the *ls* command with appropriate options (*a*, *F*, *l*) depending upon the requested operation.
7. Copy, move, and remove files within your directories as directed.
8. Copy, move, and remove directories as specified.
9. Move to different directories as specified.
10. Use relative pathnames for movement within directories.
11. Change permission on personal files and directories as specified.

CONVEXOS FILE SYSTEM

CONVEXOS organizes information on its mass storage devices into files.

Files:

- Considered sequence of bytes of raw data
- Different types:
  - Executable programs
  - Data files
  - Physical devices
  - Text files
  - Communication channels
  - Directory files



ConvexOS File System

Each of the files is just a sequence of bytes as far as the system is concerned. For example, to see a visual representation of all the bytes of the file test1, type:

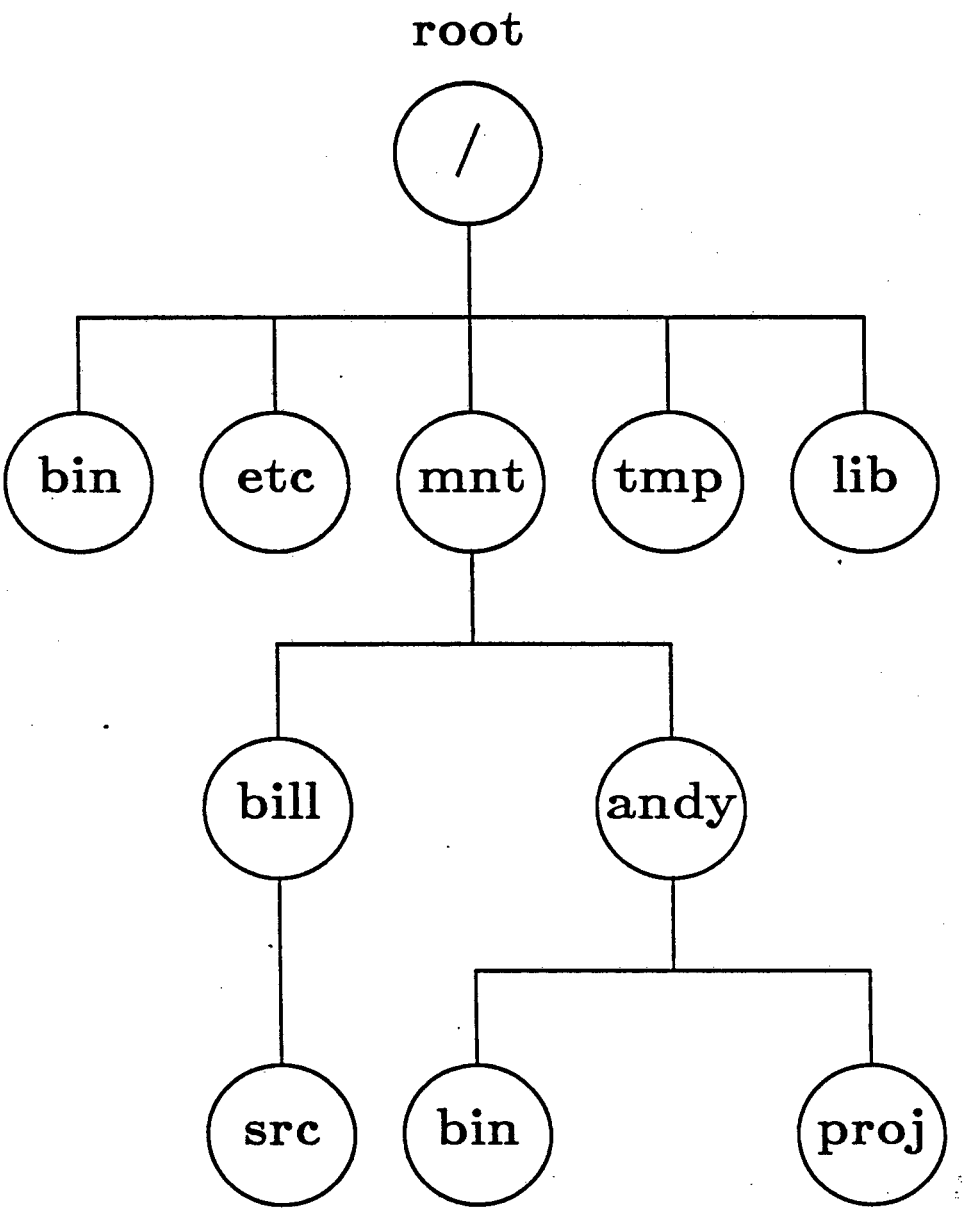
od test1

Your entry and the results:

```
% od test1

0000000  054557 072440 073551 066154 020156 067564 064543 062440
0000020  072150 060564 020164 064145 020164 062570 072040 074557
0000040  072412 060562 062440 062556 072145 071151 067147 020141
0000060  070160 062541 071163 020141 072040 072150 062440 061157
0000100  072164 067555 005157 063040 074557 072562 020163 061562
0000120  062545 067056 020040 041565 071162 062556 072154 074454
0000140  020171 067565 020143 060556 005143 067562 071145 061564
0000160  020145 071162 067562 071440 067556 066171 020157 067040
0000200  072150 062440 061565 071162 062556 072012 072145 074164
0000220  020154 064556 062440 073551 072150 020164 064145 020102
0000240  040503 045523 050101 041505 020153 062571 027012 052150
0000260  064563 020151 071440 072150 062440 060544 062151 072151
0000300  067556 060554 020154 064556 062440 067546 020164 062570
0000320  072056 005000
0000323
%
```

HIERARCHICAL STRUCTURE

Files are organized into directories. The directory tree:

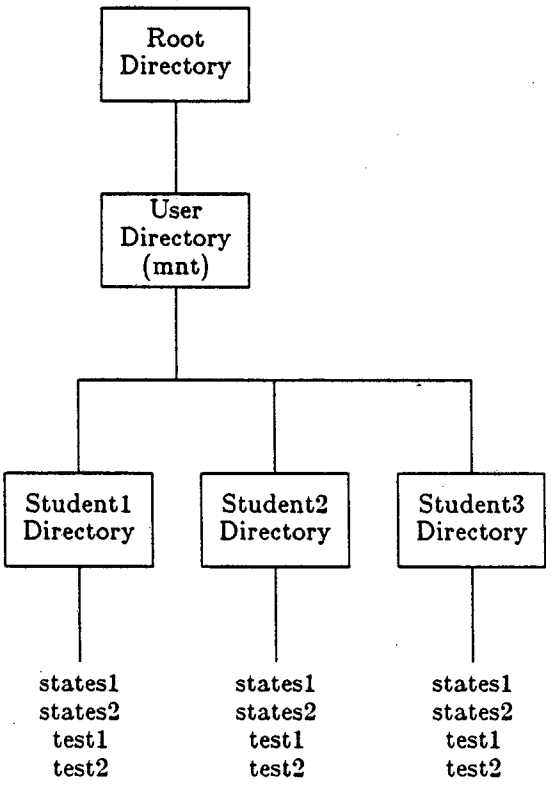


Hierarchical Structure

The hierarchical structure is a network of directories. The structure:

- root - top level in the ConvexOS system file structure
- bin - the library of ConvexOS utility programs
- etc - the system files; essential data and maintenance utilities
- usr - general-purpose directory, may contain users' directories
- tmp - temporary file storage
- lib - linkable subroutine libraries for FORTRAN and C programs

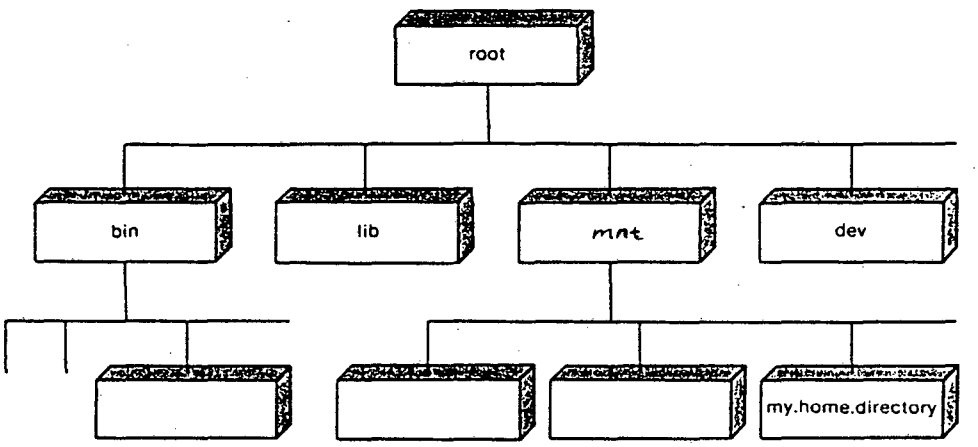
Your directories have the parent *mnt*. Your placement in the hierarchy at this time is:



DIRECTORIES

- Home Directory:
  - This is your initial working directory each time you log in.
  - The home directory is a subdirectory of the root and usually has the same name as your CONVEXOS user name.
  - You can display its pathname by using the *pwd* command (print working directory).

The directory system.

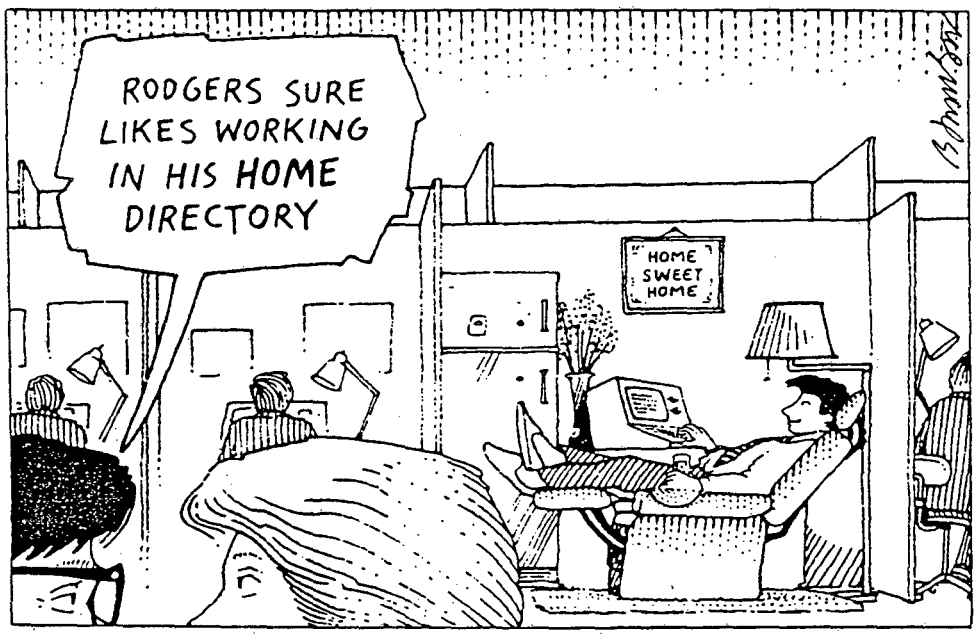


Home Directory

Determine the pathname of your home directory (current working directory) using the *pwd* (print working directory) command.

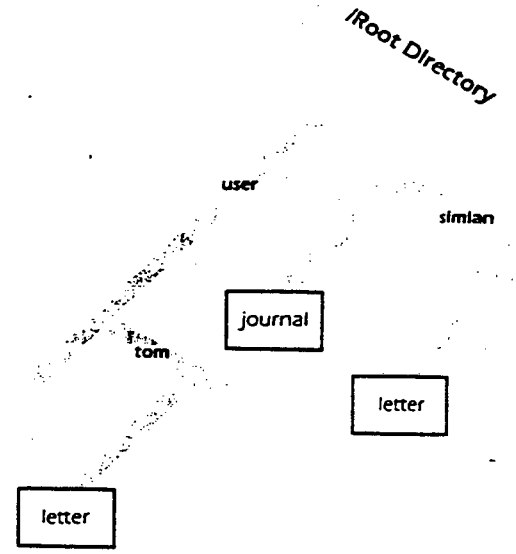
Your entry:

```
% pwd
/mnt/usrname
%
```



PATHNAMES

- Pathnames are the roadmaps through the CONVEXOS hierarchical structure.
- Pathnames are unique filenames (file may be a directory).
- An absolute pathname:
  - Leads from the root directory (/)
  - Includes names of parent directories
  - Contain slashes separating the directory names

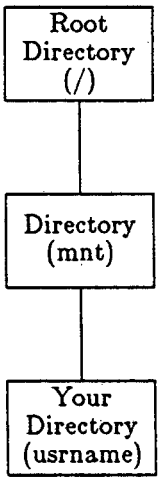


Pathnames

/user/simian	for the directory "simian"
/user/simian/journal	for the file "journal" in simian's directory
/user/simian/letter	for the file "letter" in simian's directory
/user/tom/letter	for the file "letter" in tom's directory

Absolute Pathnames

When you log on, your location is your home directory. The absolute path to your directory looks like this:



PATHNAMES

Relative Pathnames:

- Begin at current working directory
- Relative to your current location in the hierarchical structure
- Begin without a slash
- Begin with subdirectory name or filename in current working directory

  For example, if your working directory is:

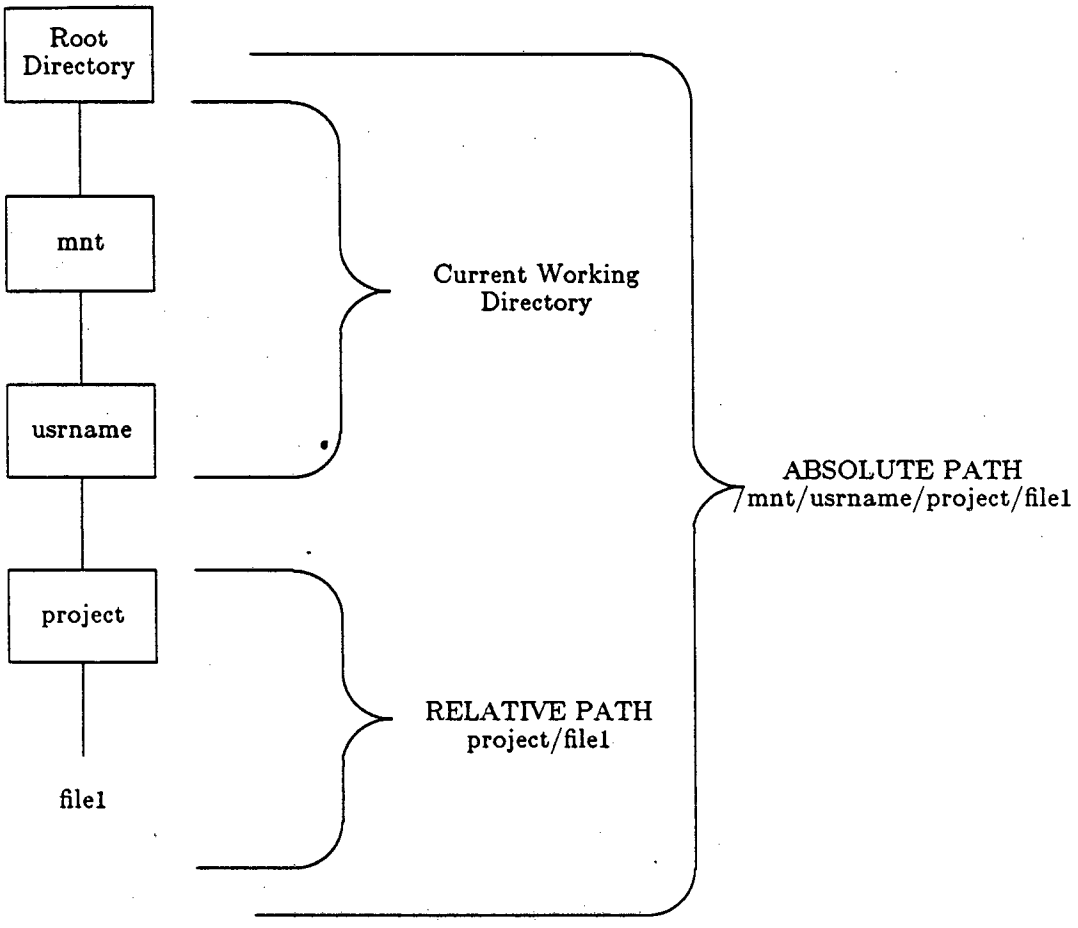
  ```
  /mnt/andy
  ```

  then you can refer to the file *data* in the *proj* subdirectory as:

  ```
  proj/data
  ```

Relative Pathnames

Assume that your current working directory is */mnt/usrname*, and that you want to work with a file named *file1* in the directory *project*. What is the relative pathname to the file named *file1*?



LISTING DIRECTORY CONTENTS

The *ls* command lists the contents of a directory.

- The format:

  `ls` *[-a, -l, -F, others] [directory_name ...]*

- `ls` without command options or a directory name lists the contents of your current working directory

- `ls` with the option `-a`:

  - Lists all the files of your current directory (like *ls*)

  and

  - Lists files beginning with a period (.)

Listing Directory Contents

Display the contents of your current working directory. (If you have previously changed directories, type **cd**.) Your entry and the response looks similar to:

```
% ls
states1   test1        test1.echo    test3
states2   test1.date   test2
%
```

In addition to the contents displayed in the previous exercises, you also have some files beginning with a period. Include those files when you list the contents of your current working directory. Your entry:

```
% ls -a
.         .exrc     .msgsrc   test1       test2
..        .login    states1   test1.date  test3
.cshrc    .logout   states2   test1.echo
%
```

LISTING DIRECTORY CONTENTS (cont.)

- Using the *-l* option causes a listing of the directory contents in "long" form:

```
% ls -l or ll
total 2
drwxrwxr-x 2 andy   24 Aug 15 13:40 bin
drwxrwxr-x 2 andy   24 Aug 12 12:15 proj
```

total 2 is the number of disk blocks used by the files in the directory

- The fields:

  - File type (*d*) and permissions (*rwxrwxr-x*)
  - Number of names linked to the file (*2*)
  - Name of owner of the file (*andy*)
  - Size of file in bytes (*24*)
  - Date the file was last modified (*Aug 15*)
  - Time file was last modified (*13:40*)
  - Filename or directory name (*bin*)

Listing Directories in Long Form

List the contents of the *root* directory in the "long" format. Your entry looks similar to this:

```
% ls -l /
total 686
drwxrwxr-x  2 root     1024    Jun 14   15:18   bin
drwxrwxr-x  2 root     3072     Aug  4   11:15   dev
lrwxrwxr-x  1 root       14     May  1   17:13   disco -> /scratch/disco
drwxr-xr-x  2 root     2560    Aug 12   09:31   etc
drwxr-xr-x 10 root      512    Jul 31   21:48   fonts
-rwxrwxr-x  1 root    77824    Jun 26   17:29   hunt.driver
drwxrwxr-x  2 root      512    Jun 25   15:55   lib
drwxr-xr-x  2 root     8192     Nov  5    1985   lost+found
drwxr-xr-x 53 root     1536     Aug  8   17:31   tac
drwxrwxrwx 15 root     1024    Aug 11   23:32   scratch
lrwxr-xr-x  1 root        8    Jul 15   11:20   ssc -> /mnt/ssc
lrwxrwxr-x  1 root        3     Jun  3   06:23   sw -> mnt
lrwxrwxrwx  1 root        7     Jun  6   18:43   sys -> usr/sys
drwxr-xr-x 56 root     1024    Aug 12   13:03   mnt
drwxrwxrwx  6 root     2560    Aug 12   15:33   tmp
drwxrwxrwx 10 root      512     Aug  8   14:57   tmph
drwxr-xr-x 26 root      512     Aug  6   16:34   usr
drwxr-xr-x 28 root     1024    Jul 14   10:23   utd
-rw-rw-r--  1 root   561152    Jun 17   10:25   vmunix
%
```

LISTING DIRECTORY CONTENTS (cont.)

Type/permission fields (drwxrwxr-x):

- The first character is the file type:

  d = directory

  c = character device

  b = block device

  l = symbolic link to another file

  - = regular file

  s = socket

- The next nine characters form 3 sets of 3 permission flags – user (rwx), group (rwx), other (r-x)

  r = read permission
  w = write permission
  x or s = execute permission

Exploring Directory Contents

Review the "long" listing of the *root* directory and answer the following questions:

1. How many disk blocks are used by the files in the *root* directory?

2. How many subdirectories appear in the *root* directory?

3. Some of the listings begin with a dash (-); what does the dash mean?

4. What is the name of the largest file listed in the *root* directory?

5. Which files or directories allow read, write, and execute permission for user, group, and other?

LISTING DIRECTORY CONTENTS (cont.)

- The *-F* (BSD) option marks:
  - Directories with a slash (/)
  - Executable programs with an asterisk (*)
  - Symbolically linked files with an at sign (@)
- In the following example, the file *prog* is an executable program, and the file *project* is a directory:

```
% ls -F /usr/andy
prog*   project/   test1   data1
```

Using the *lf* command instead of *ls -F* obtains the same results.

Using the *ls -F* Command

Use the *-F* option to list the *root* directory. Your entry looks similar to:

```
% ls -F /
bin/        etc/          lost+found/  sw/     tmph/
core        fonts/        mnt/         sys/    usr/
dev/        hunt driver*  scratch/    tac/    utd/
disco@      lib/          ssc/         tmp/    vmunix
%
```

Identify the executable files, directories, and symbolically linked files.

FILENAMES AND METACHARACTERS

- Asterisk (*) - matches any number of characters

  `dat*` matches *dat*, *date*, *data1*, and *date.old*

- Question Mark (?) - matches any single character

  `file?` matches *file1*, *fileA*, and *filed*

- Square Brackets [] - List of ASCII characters inside the brackets matches any character in the list; separate a range of characters with a dash (-).

  `*[135]` matches any filenames that end with a "1", a "3", or a "5"

  `[A-Z0-9]*` matches any filenames that begin with an uppercase letter or a digit

Using Metacharacters

Use the metacharacters with the *ls* command. First list the file in your home directory that begin with *t*. Your entry:

```
% ls t*
test1   test1.date   test1.echo test2   test3
%
```

List the files *test1, test2, test3*. Your entry:

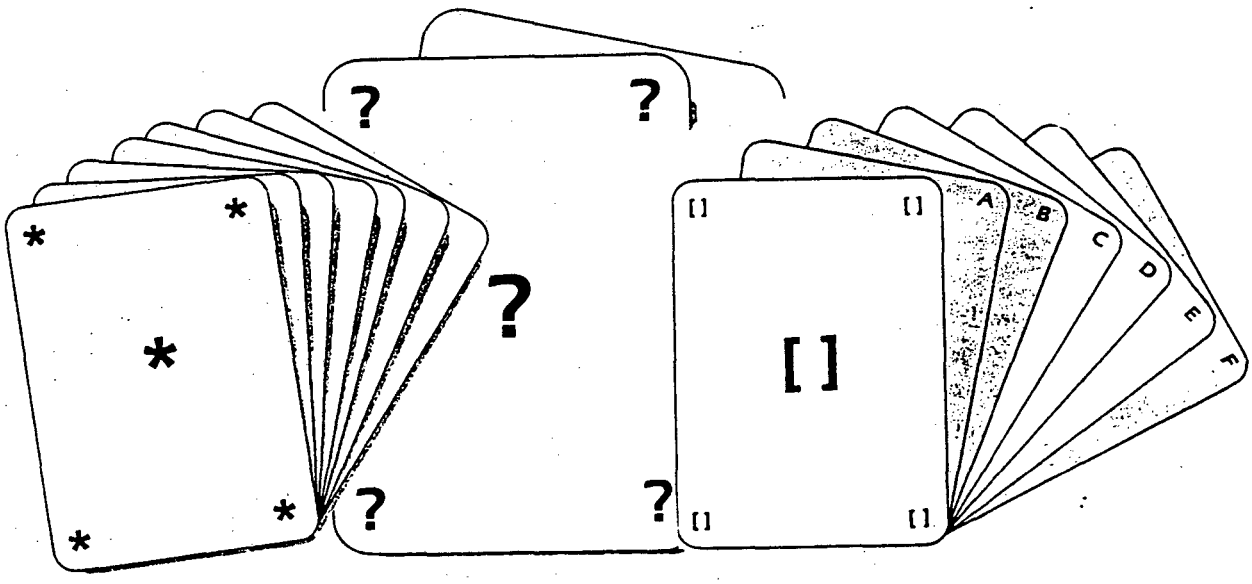
```
% ls test?
test1   test2   test3
%
```

List all the files in your home directory that end with *1* or *3*. Your entry:

```
% ls *[13]
states1   test1   test3
%
```

List all the files in your home directory that begin with uppercase characters A-E. Your entry:

```
% ls [A-E]*
no match.
%
```

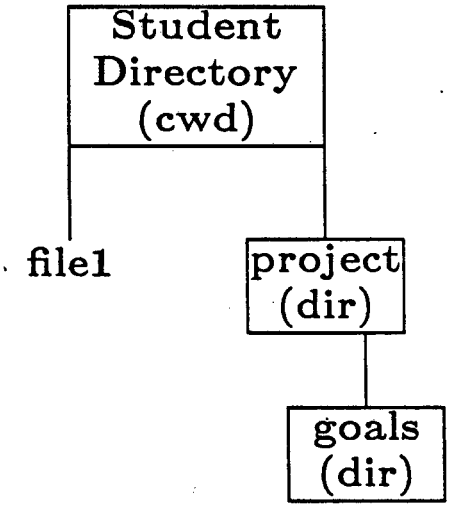


CREATING DIRECTORIES

The *mkdir* command:

- Allows you to build your own directory subsystem
- Allows you to develop a system that fits your needs
- Creates an empty "new directory"
- Has the form:

```
mkdir directory_name ...
% mkdir project
% mkdir project/goals
```



Creating Directories

Make a directory named *project*. Your entry:

```
% mkdir project
%
```

List your home directory contents using the option to identify directories in the listing. Your entry:

```
% ls -F
project/  states2  test1.date test2
states1   test1    test1.echo test3
%
```

Remaining in your home directory, make a directory called *goals* in the *project* directory. Your entry:

```
% mkdir project/goals
%
```

Make another directory entitled *bin* as a subdirectory of your home directory. Your entry:

```
% mkdir bin
%
```

Determine the subdirectories by listing the contents of your home directory. Your entry:

```
% ls -F
bin/      states2    test1.echo
project/  test1      test2
states1   test1.date test3
%
```

You made a directory named *goals*. Why isn't it listed in your home directory?

CHANGING DIRECTORIES

- Working with directories:

  cd *pathname* transfers you to the named directory

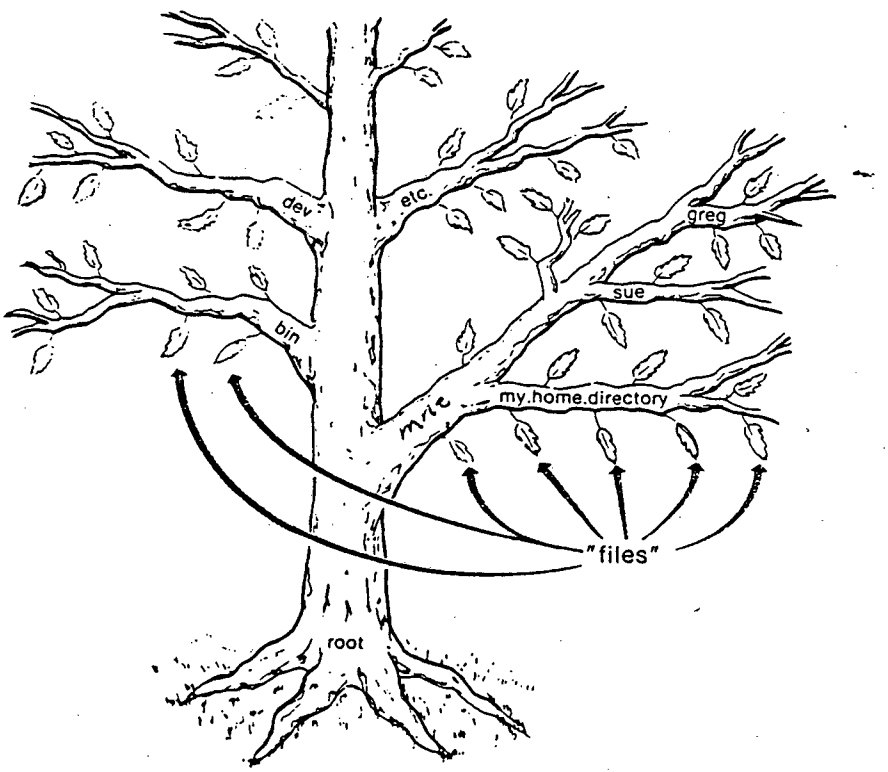
  . refers to your current directory

  .. refers to parent directory

  ~ serves as shorthand name for your home directory

  ~username serves as shorthand for the specified user's home directory

  cd returns you to your home directory



Changing Directories

Change your directory to *goals* and display the pathname. Your entry:

```
% cd project/goals
% pwd
/mnt/usrname/project/goals
%
```

Use the *cd* .. command and display your location. Your entry:

```
% cd ..
% pwd
/mnt/usrname/project
%
```

The tilde can be used to specify directories other than your own. Change to another user's directory (another user in your class), display the pathname, and list the files.

```
% cd ~floppsy
% pwd
/usr/floppsy
% ls
. unreadable
%
```

Use the tilde to specify your home directory. Change the directory to *project.* Then display the pathname with the *pwd* command.

```
% cd ~/project
% pwd
/mnt/username/project
%
```

Use **cd** to return to your home directory. Verify that your home directory pathname is displayed.

```
% cd
% pwd
/mnt/username
%
```

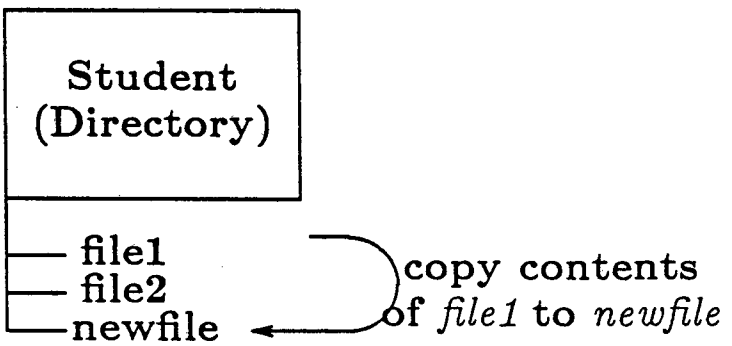
COPYING FILES

The *cp* command:

- Duplicates existing file as a new file
- Makes an exact copy of the existing file
- Creates the "new file" if it does not exist
- Overwrites "file" if it already exists
- Has the form:

  cp *filename_to_be_copied new_filename*

  % cp file1 newfile



- To keep an "existing" file from being overwritten, use the form:

  cp -i *filename_to_be_copied new_filename*

Copying Files

Make a copy of the file *test1.date*; name the file *yourcopy*. Your entry:

```
% cp test1.date yourcopy
%
```

To see that *yourcopy* was created, list the contents of your home directory. Your entry looks similar to:

```
% ls
states1  test1      test1.echo test3
states2  test1.date test2      yourcopy
%
```

As you can see, a "new file" was created.

Now copy *test1.echo* to *yourcopy*. What happened? Your entry:

```
% cp test1.echo yourcopy
%
```

Use the *-i* option when you copy *states1* to *yourcopy*. Since *yourcopy* exists, you are prompted with `overwrite yourcopy?`. Don't overwrite the file; respond with **n**. Your entry:

```
% cp -i states1 yourcopy
overwrite yourcopy?n
%
```

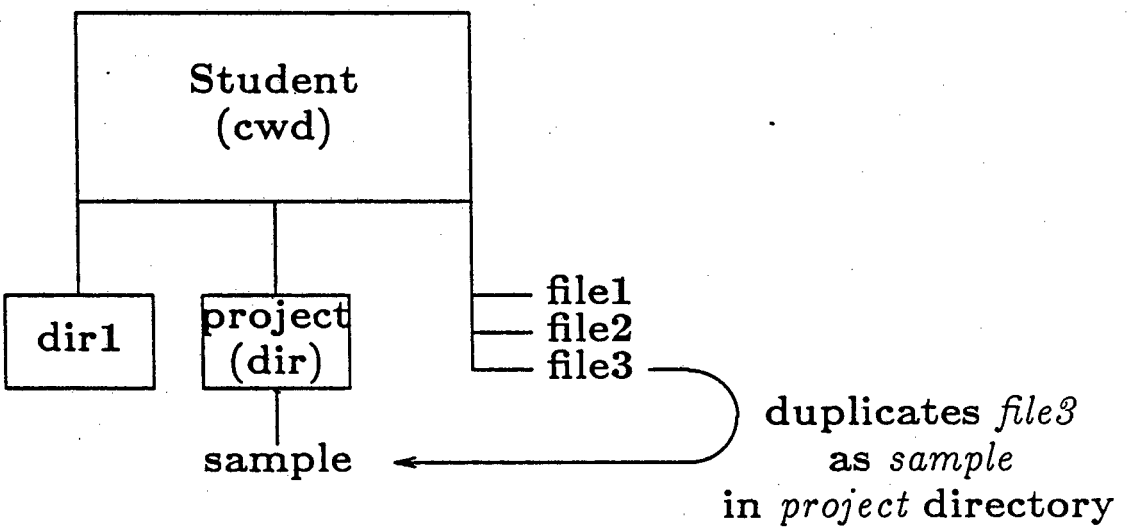
COPYING FILES BETWEEN DIRECTORIES

The *cp* command:

- Copies a file(s) between directories
- Does not create a "new directory"; the directory must exist
- Has the form:

cp *filename(s) dirname*

% cp file3 project/sample



Copying Files Between Directories

Make sure you are in your home directory. (If you aren't, type **cd**). Copy *test1* to the file named *sample1* in the *goals* directory. Then display the directory contents of *goals*. Your entry:

```
% cp test1 project/goals/sample1
% ls project/goals
sample1
%
```

Copy both *state1* and *states2* to the *bin* directory. Do the copying in one step; use the asterisk (*) as part of the filename. Then list the contents of *bin*. Your entry:

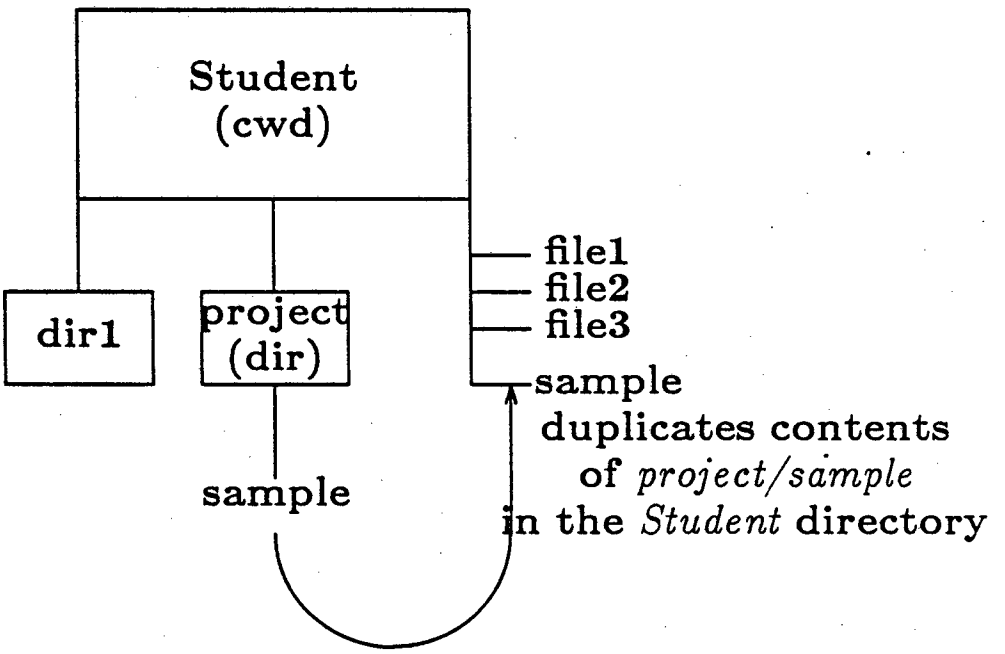
```
% cp states* bin
% ls bin
states1  states2
%
```

COPYING FILES BETWEEN DIRECTORIES (cont.)

- To copy a file from another directory into your current working directory (cwd), use a dot in place of the directory name:

cp *dir/filename* .

```
% cp project/sample .
```



Copying Files to the Home Directory

Remaining in your home directory copy *sample1* to your home directory and list the contents of your home directory. Your entry:

```
% cp project/goals/sample1 .
% ls
bin       states1   test1.date   test3
project   states2   test1.echo
sample1   test1     test2
%
```

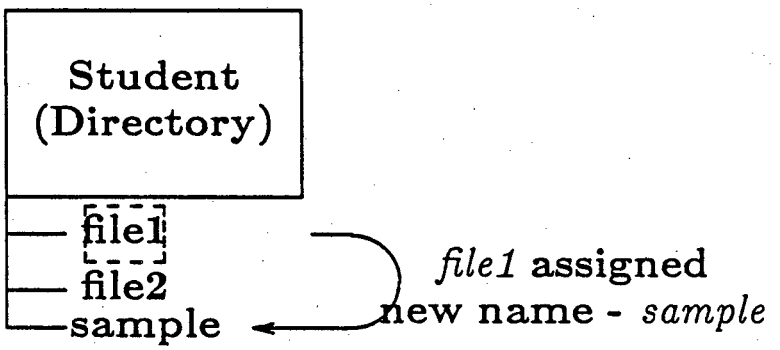
MOVING OR RENAMING FILES

The *mv* command:

- Gives a "new name" to the "existing file"
- Creates the "new file" if does not exist
- Overwrites the "file" if it exists
- Removes the existing filename when the *mv* command is invoked
- Has the form:

  `mv` *file_to_be_renamed new_filename*

  `% mv file1 sample`



- To keep an "existing" file from being overwritten, use the form:

  `mv -i` *filename_to_be_renamed new_filename*

Renaming Files

Rename (move) the file *yourcopy* to *sample*. Your entry:

```
% mv yourcopy sample
%
```

To see that *yourcopy* has been renamed *sample*, do a listing of your home directory contents. Your entry looks similar to:

```
% ls
sample   states2  test1.date test2
states1  test1    test1.echo test3
%
```

MOVING FILES TO ANOTHER DIRECTORY

The *mv* command:

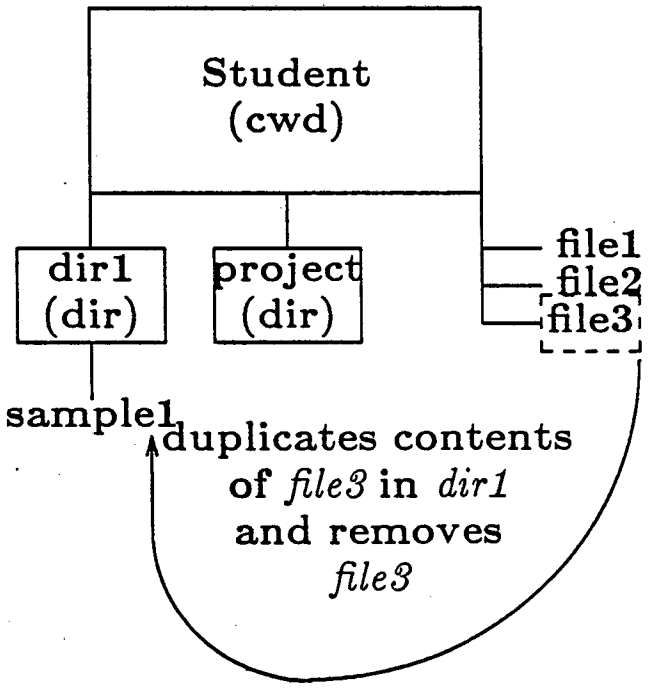
- Moves an existing file to another directory
- Creates the "new file" if it does not exist
- Overwrites the "file" if it exists
- Removes the existing file
- Has the form:

mv *dir_filename new_dir_filename*

or

mv *dir_filename* .

% mv file3 dir1/sample1



Moving Files to Directories

Move *test1.date* from your home directory to the *project* directory; name it *sample3*. List the contents of the *project* directory. Your entry:

```
% mv test1.date project/sample3
% ls project
goals sample2 sample3
%
```

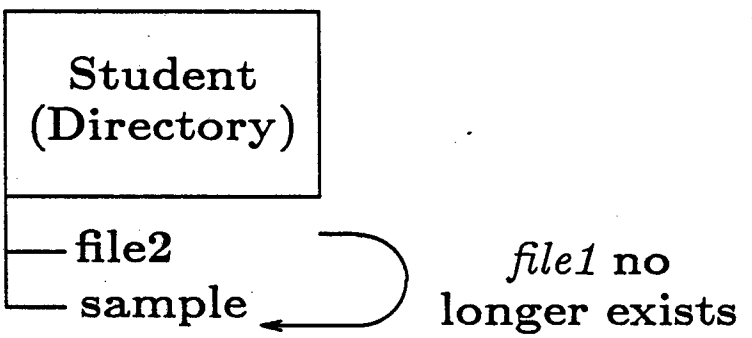
REMOVING FILES

The *rm* command:

- Deletes the named file(s)
- Has the form:

  `rm` *filename ...*

  `% rm file1`



- To be prompted with `remove` *filename* each time you attempt to remove a file, use the form:

  `rm -i` *filename*

Removing Files

Remove the file *sample*. Your entry:

```
% rm sample
%
```

Now list your directory contents:

```
% ls
states1  test1      test1.echo test3
states2  test1.date test2
%
```

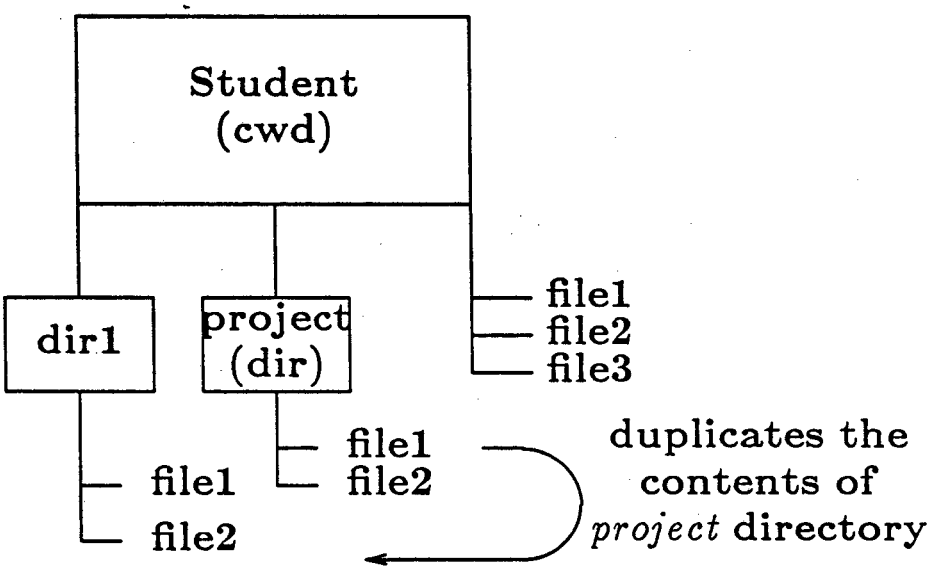
COPYING DIRECTORY CONTENTS

The *cpall* command:

- Copies the directory contents (with owner's permissions)
- Both directories must exist before executing *cpall.*
- Has the form:

`cpall` *dirname1 dirname2*

```
% cpall project dir1
```



Copying Directory Contents

From your home directory copy the *project* directory contents to the *bin* directory using the *cpall* command. Your entry:

```
% cpall project bin
%
```

Display the contents of *bin*. Your entry:

```
% ls bin
goals sample2 sample3 states1 states2
%
```

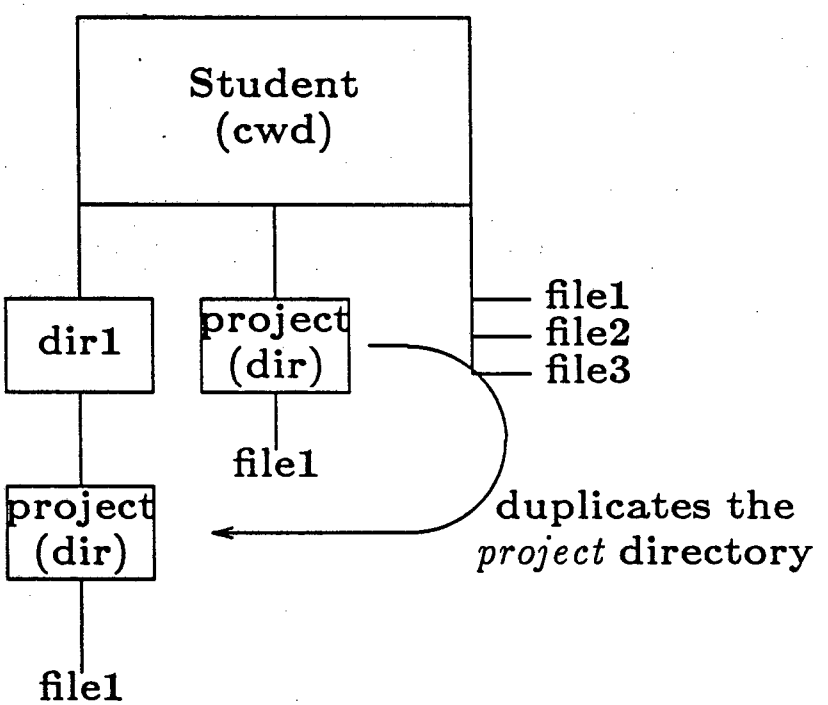
COPYING DIRECTORY STRUCTURES

The *cp* command with the *-r* option:

- Copies directory structures (with user's permissions)
- Both directories must exist before executing *cp -r*.
- Has the form:

```
cp -r dirname1 dirname2
```

```
% cp -r project dir1
```



Copying Directory Structures

In your home directory, create a directory named *dir1*. Then copy the directory structure *project* to this directory. List the directory contents. Your entry:

```
% mkdir dir1
% cp -r project dir1
% ls -F dir1
% project/
%
```

List the contents of *dir1/project*. Your entry:

```
% ls dir1/project
goals sample2 sample3
%
```

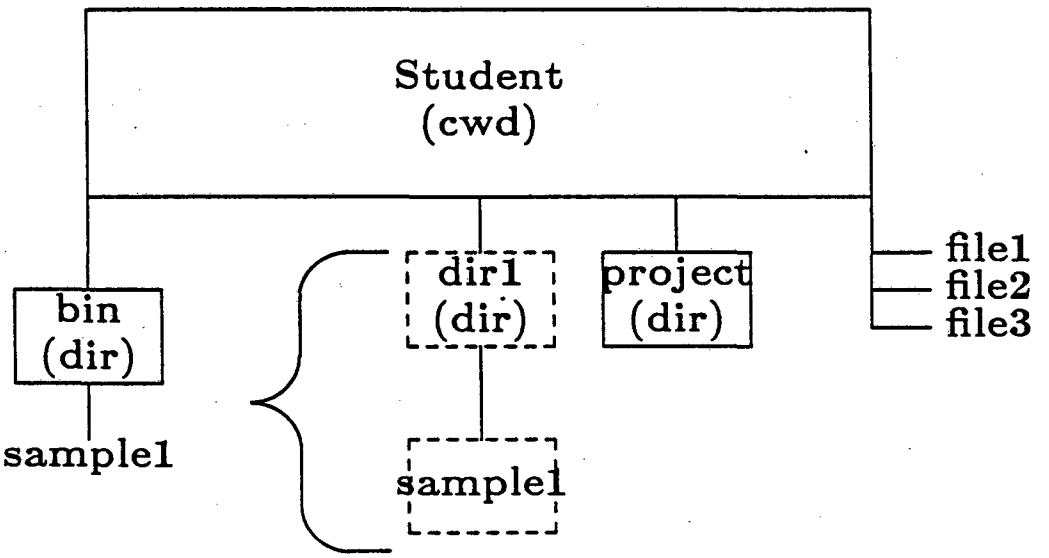
RENAMING DIRECTORIES

The *mv* command:

- Allows renaming of a directory
- Moves the existing directory to "new" directory
- Has the form:

  `mv` *dir_name newdir_name*

  `% mv dir1 bin`



Renaming Directories

Give the the *goals* directory a different name; call it *docs*. Do a listing of your home directory contents, using the option to indicate directories. Your entry:

```
% mv project/goals docs
% ls -F
bin/                    project/  states2    test2
docs/                   sample1   test1      test3
mytest                  states1   test1.echo
%
```

List the contents of *project*; notice the *goals* directory no longer exists in the *project* directory. Your entry:

```
% ls project
sample2 sample3
%
```

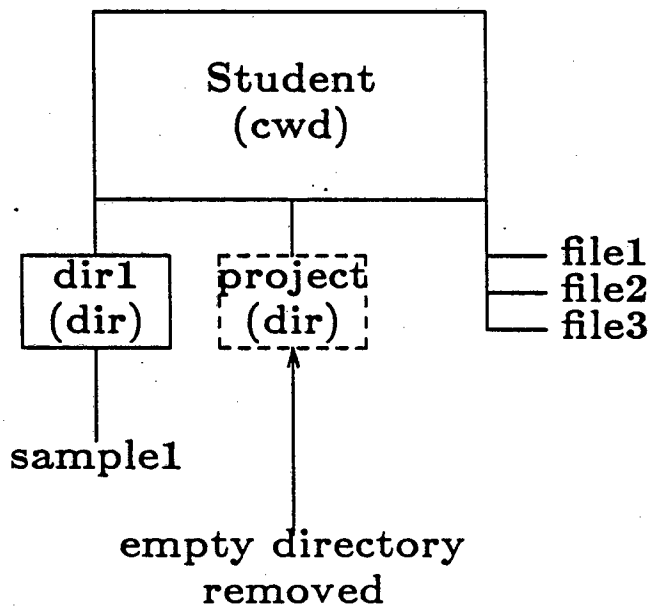
REMOVING DIRECTORIES

The *rmdir* command:

- Removes "empty" directories
- Has the form:

`rmdir` *dir_name*

```
% rmdir project
```



Removing Empty Directories

Attempt to remove the *dir1/project* directory. Your entry:

```
% rmdir dir1/project
rmdir: dir1/project Directory not empty
%
```

Why can't you remove this specific directory?

Make a directory entitled *test*. Make sure that it exists, then delete it. Your entry looks something like this:

```
% mkdir test
% ls -F
bin/            project/  states2  test1.echo
dir1/           sample1   test/    test2
mytest          states1   test1    test3
% rmdir test
%
```

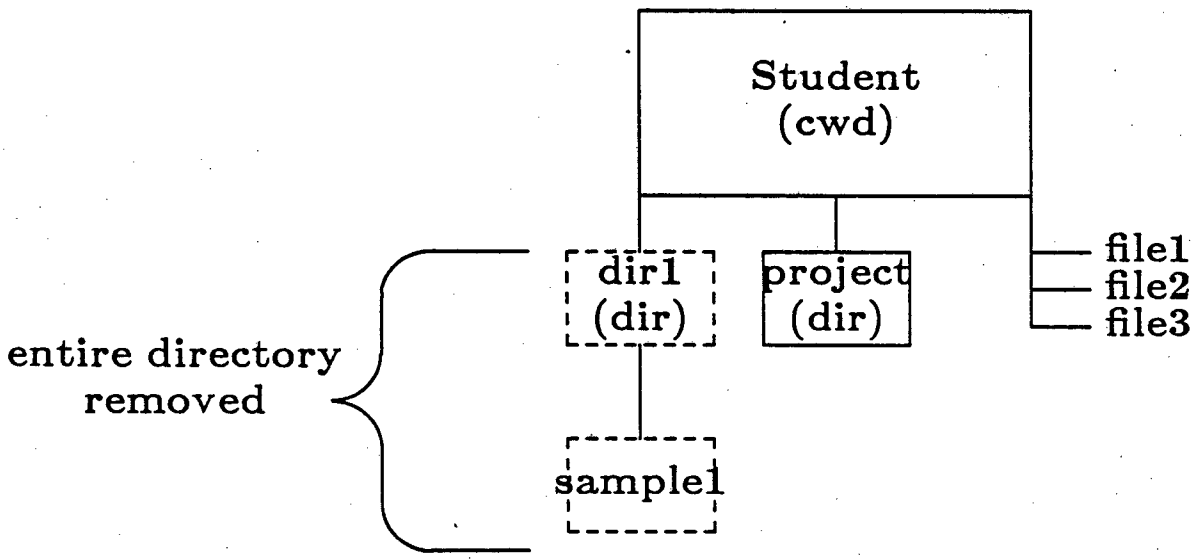
REMOVING DIRECTORIES

The *rm -r* command:

- Removes "full" directories
- Deletes the contents of everything in the directory, recursively
- Has the form:

`rm -r` *dir_name*

```
% rm -r dir1
```



Removing Directories and Their Contents

Remain in your home directory. Even though the *dir1* has files and directories in it, you no longer need it; remove it.

```
% rm -r dir1
%
```

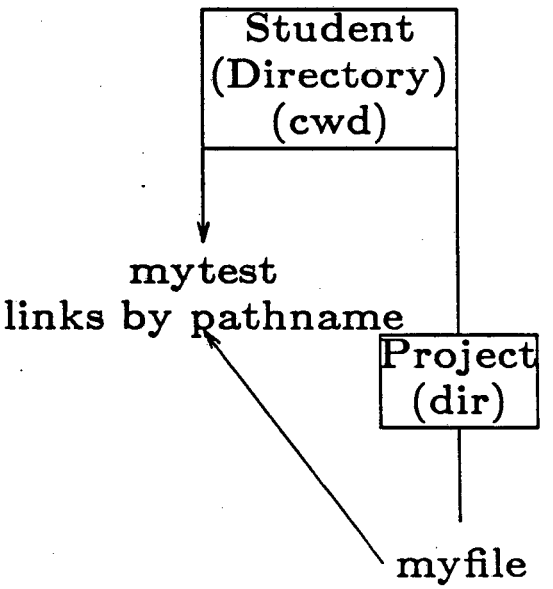
LINKING FILES

A hard link:

- Allows you to assign mutliple names to a single file
- Establishes another pathname to the same file
- Does not remove the data when a link is deleted
- Has the form:

  `ln` *existing_file file_to_be_linked*

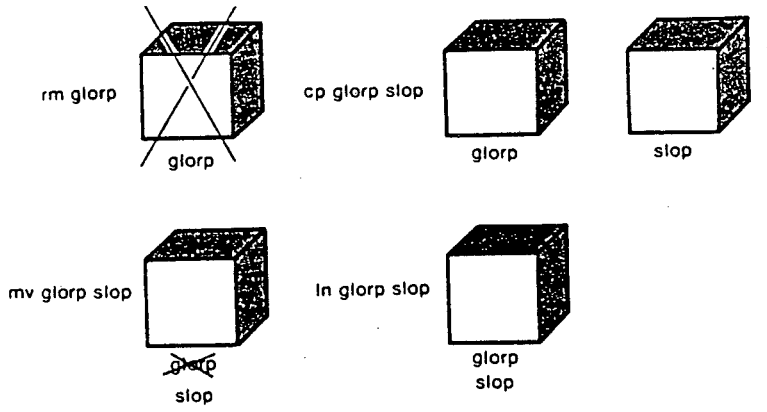
```
% ln project/myfile mytest
```



Linking Files

Remain in your home directory. Make a hard link between *sample1* in *goals* with the new file *mytest* in your home directory. Then do a "long" listing of both *sample1* and *mytest*. Your entry:

```
% ln project/goals/sample1 mytest
% ls -l mytest
-rw-r--r-- 2 usrname     211 Aug 20 10:22 mytest
% ls -l project/goals/sample1
-rw-r--r-- 2 usrname     211 Aug 20 10:22 project/goals/sample1
%
```



A comparison of *rm*, *mv*, *cp*, and *ln*.

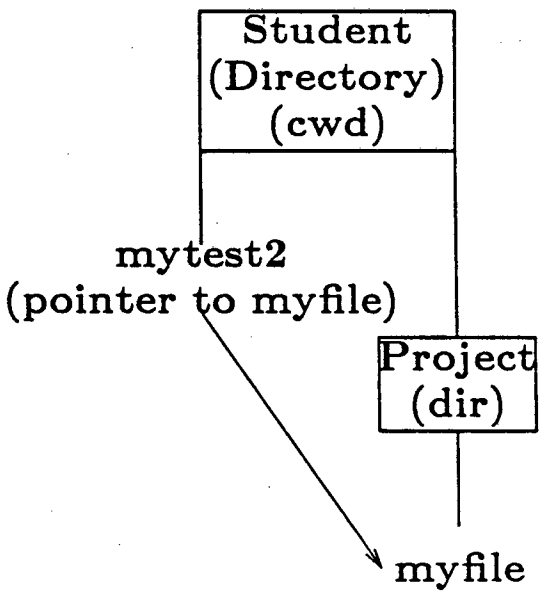
LINKING FILES (cont.)

A symbolic link:

- Links a new filename to a block on the disk that already contains a file
- Is really only a text string that is substituted in a pathname whenever the symbolic link appears in a pathname
- Has the form:

  `ln -s` *existing_file file_to_be_linked*

  `% ln project/myfile mytest2`



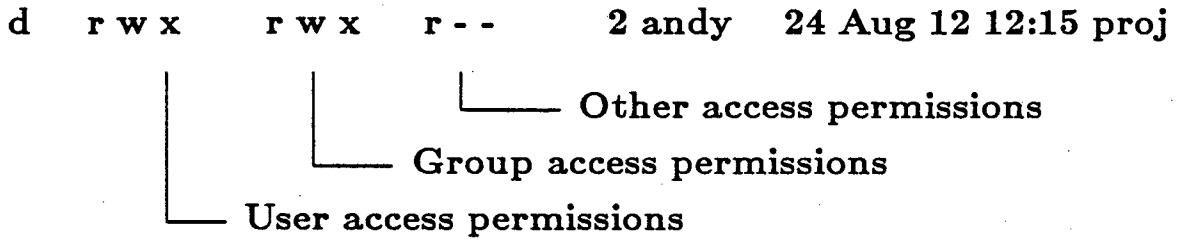
Symbolic Linking

Remain in your current directory. Make a symbolic link between */tac/bayne/lnsample* and *sample2* in the *project* directory. Then do a "long" form listing for each file. Your entry:

```
% ln -s ~bayne/lnsample project/sample2
% ls -l ~bayne/lnsample
-rw-r--r--  1 bayne   37 Aug  7 16:43 lnsample
% ls -l project/sample2
lrwxrwxr-x  1 usrname   5 Aug 20 10:57 project/sample2 -> /tac/bayne/lnsample
%
```

FILE SECURITY

- Three classes of permissions:



  - User - the owner of the file
  - Group - selected individuals
  - Other (world) - anyone with access to the system

- Types:

  - Read (r) - permission to read the contents of the file
  - Write (w) - permission to change the contents of a file
  - Execute (x) - permission to run the computer program contained in a file
  - Dash (-) - permission denied

Identifying Permissions

If you are not in your home directory, type **cd**. Determine the permissions on your directories and files. Your entry:

```
% ls -l
total 104
drwxwrxr-x 2 usrname   512    Aug 20   13:44    bin
drwxrwxr-x 2 usrname   512    Aug 20   13:44    docs
-rw-rw-r-- 2 usrname   211    Aug 20   10:22    mytest
drwxrwxr-x 2 usrname   512    Aug 20   13:44    project
-rw-r--r-- 1 usrname   211    Aug 20   10:25    sample1
-rw-r--r-- 1 usrname    37     Aug 7   16:43    states1
-rw-r--r-- 1 usrname    38     Aug 7   16:45    states2
-rw-r--r-- 1 usrname   211     Aug 7   16:19    test1
-rw-r--r-- 1 usrname    20    Aug 12   13:38    test1.echo
-rw-r--r-- 1 usrname   138     Aug 7   16:45    test2
-rw-r--r-- 1 usrname    88    Aug 12   11:58    test3
%
```

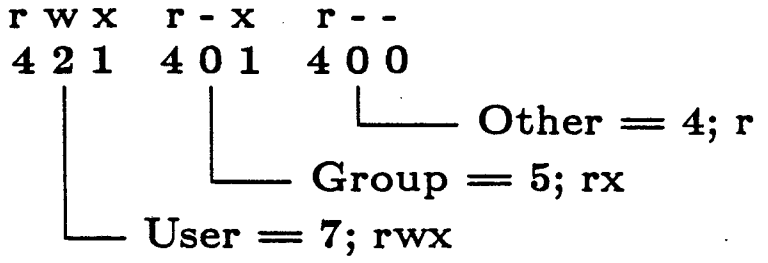
What are the user, group, and other permissions, respectively?

CHANGING FILE PROTECTION

- The *chmod* command allows you to change the protection on your files.
- The *chmod* command with numeral protection indicators:

  chmod *permission_bits filename*

- Octal numbers specify permission access:



- Permission for each class (owner, group, other) can total 7:
  - Read permission = 4
  - Write permission = 2
  - Execute permission = 1
  - Permission denied = 0
- Examples:

  % chmod 777 *filename* (rwxrwxrwx)

  % chmod 444 *filename* (r--r--r--)

  % chmod 755 *filename* (rwxr-xr-x)

Changing Permissions with Octal Indicators

Using the octal numbers, change the permission on *test1* to:

user - read, write, execute
group - read, write, execute
other - read, write, execute

Display the permissions for this file after you have changed them.

Your entry:

```
% chmod 777 test1
% ls -l test1
-rwxrwxrwx 1 usrname       211       Aug 7   16:19       test1
%
```

For the same file, change the permissions to read:

usr - read, write
group - read
other - read

Display the permissions after you have completed the change.

Your entry:

```
% chmod 644 test1
% ls -l test1
-rw-r--r-- 1 usrname       211       Aug 7   16:19       test1
%
```

CHANGING FILE PROTECTION (cont.)

An alternate *chmod* command format:

chmod [*who operator permission*] *filename*

- who:
  - user = u
  - group = g
  - other = o
  - all three classes = a
- operator:
  - add = +
  - remove = -
  - assign = =
- permission:
  - read = r
  - write = w
  - execute = x
- Example: rw-rwxrwx . . . states1

```
% chmod g=r states1 (rw-r--rwx)
% chmod 647 states1 (rw-r--rwx)

% chmod o-rwx states1 (rw-r-----)
% chmod 640 states1 (rw-r-----)
```

Changing Permission Indicators

Change the permissions on the file *test3* using *who*, *operator*, and *permission* indicators. The permissions should be:

user - read, write
group - r
other - all permissions denied

Display the results.

Your entry (assuming the permissions are -rw-r--r--):

```
% chmod o-r test3
% ls -l test3
-rw-r-----  1 usrname       88        Aug 12   11:58       test3
%
```

For the *test3* file:

usr - read, write, execute
group - read, write, execute
other - read, write, execute

Then display your results. Your entry:

```
% chmod a=rwx test3   (chmod u+x,g+x,o+rwx test3)
% ls -l test3
-rwxrwxrwx  1 usrname       88        Aug 12   11:58       test3
%
```

Exercises for Chapter 3

1. What is the name of the directory into which you log in?

2. Draw the hierarchical structure to show a subdirectory (of your home directory) named *personal* with a file named *money*.

3. Write a command that always takes you back to your home directory when you are in another directory?

4. Who owns the file */etc/termcap*? Write the command(s) you used to find the username. What are the permission flags for this file? What do these permission flags mean?

5. Someone from your group has been making changes in your *test9* file. Change the permissions on this file so that no one in your group can write to this file. What command did you use?

6. Change the permissions on your home directory so that your group and others cannot read a listing of your files. Verify that the permissions are corrected by asking one of your colleagues to attempt a listing of your home director. What command did you use?

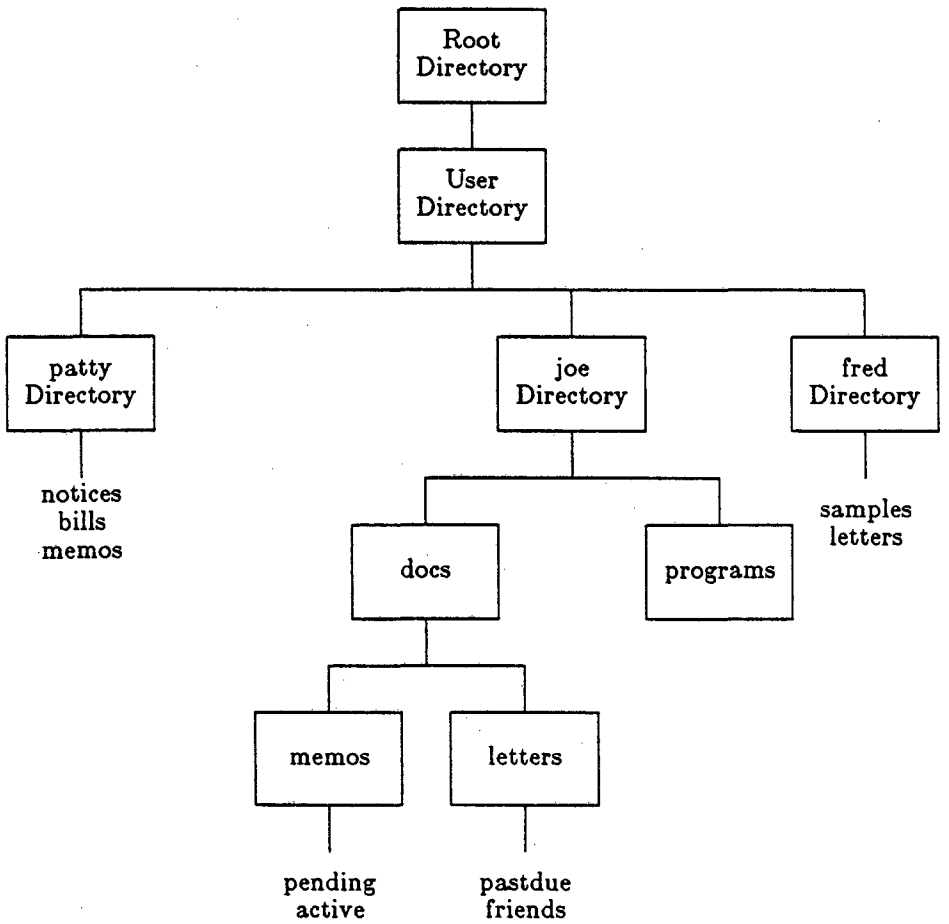
7. In your home directory, what is the largest file? How did you obtain this information?

8. How many subdirectories does the *root* directory contain? How many executable files are contained in this directory? What command did you use to get this information?

9. List two methods by which you can tell that a listing in a directory is a directory.

10. How can you tell if a file is linked to another file?

11. What does the symbol "." (period) refer to in the UNIX file system?



12. The following questions pertain to the preceding directory structure:

    a. Write a command that allows *fred* to change his directory to *joe*'s home directory.

    b. Write the command that allows Fred to change the directory back to his home directory.

    c. Write a command that changes the directory from *joe* to *letters*.

    d. Write a command that changes the directory from *letters* to *programs*.

    e. What happens if the current directory is *joe* and the command *cd memos* is entered?

    f. Write a command that allows Joe to copy Fred's file *letters*, naming it *sam1* to his *letters* directory.

    g. Write a command that changes the directory from *memos* to Patty's home directory.

    h. Patty wants to make a directory named *examples* in Fred's home directory. Can she do this? If yes, what command should she type? If no, why?

13. Link *test1* to */tmp/sample*. What message displayed? Why did you get this message?

14. What is the major difference between using a hard link and a symbolic link?

15. What happens when you create a symbolic link to a file owned by another user and that user removes the file? Can you still access the file?

16. Make a directory named *MINE*. NOw do a listing of this directory. What was displayed? Why?

17. Make a directory named *my.t*. Change to that directory; create some files using the following command:
    `touch file1 file2 file3 file4`
    Using the command that will prompt you about removal of each file, remove all the files in the *mt.t* directory. What commands did you use to complete this task?

18. Lists all the files in your directory beginning with the letters *d-h*. What command did you use?

19. If you completed all the exercises in the preceding chapters, you have a directory named *bin*. Remove it with the command **rmdir bin**? Were you successful? Why?

20. How many files begin with the letter *s* in your home directory? What command(s) did you use to obtain this information?

21. From your home directory create a new directory named *practice* as a subdirectory of your *project* directory. (If you don't have a project directory, also create it.) What command(s) did you use?

22. Change your directory to *project/practice*. Now return to the parent directory. What is the name of the parent directory? What command(s) did you use?

23. Remove the *practice* directory. What(s) command did you use to remove this directory?

24. Change the directory to the parent directory of your home directory. What command did you use? What is the name of the parent of your home directory?

25. Move *test3* from your home directory to your *bin* directory. What command(s) did you use?

26. (This is a challenge question. Can you answer it?) What is the absolute pathname of the home directory of the user named *notes*? How did you determine this information?

27. Change your directory to */usr/bin*. Remain in this directory and list the contents of your home directory. What command did you use?

28. Change your directory to */usr/spool/notes* then move to */usr/spool/uucp*. What commands did you use? List the pathname of your current working directory. Return to your home directory.

29. What does the *d* mean in this listing:
drw-rw-xr-- 2 user 1024 Jun 14 15:18 letters

    Write a command that changes the permissions on this directory to read, write, and execute for the user; read and execute for group; and no permissions for others.

30. What permissions does *file1* have when **chmod 644 file1** is entered?

31. If one of your files has the permission set at *-rw-r--r*, can members of your group copy the file? If no, why can't it be copied? If yes, what permissions will the copied file have?

32. What is the difference between absolute and relative pathnames?

33. Write a command that changes the permissions on all files that end with *.f* so that you have read, write, and execute permission; the group has read and execute; and others have read only. The files to be modified are located in your *project* directory in the subdirectory *goals*.

34. Change the access privileges of all the files in your home directory so that no one else can access them. What command did you use?

4

vi Tutorial

This chapter provides tutorial instruction for using *vi*, an interactive text editor. The chapter is designed to lead you through the fundamental steps of creating and revising text. Although this guide is intended for use primarily by novice users, an experienced user may find the illustrations of the more advanced *vi* commands helpful.

Getting Started with *vi*

vi is a full-view line editor used primarily for creating and modifying programs, documents, or any textual material using directions provided by the user at a terminal. As you create and revise documents with *vi*, you will see that the movement commands are the foundation for *vi*. The movement commands are used in conjunction with the delete (*d*) and change (*c*) commands. In addition, you will also notice that you can prefix *vi* commands with numbers providing a means for executing commands on multiple lines. *vi* commands are used only for manipulating text; there are no formatting or printing commands.

The exercises in this section lead you through the fundamental steps of creating and revising text. Scan each example and then experiment with the commands to get a feeling for the actual text-editing process.

Creating a File

After you have logged in and the % (your *shell* prompt) is displayed, you are ready to create (or edit) files with *vi*. (If you wish to make a special directory for your *vi* files, do so now. Making the directory is not included in the tutorial sessions that follow.)

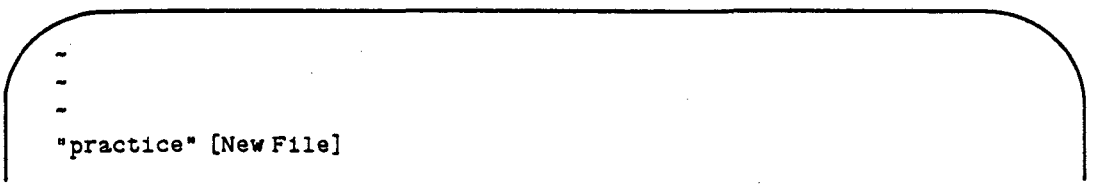
To create a new file, type **vi**, followed by a space and the *filename*; then press the **RETURN** key. You can also use a pathname instead of only a filename when creating (or editing) a file; for example, entering **vi /mnt/fred/tutorial/practice** would create (load) the file *practice* in the directory *tutorial*. If you wish to edit several files in succession, enter the string of filenames, i.e., **vi file1 file2 *.c**. In this case, the command would edit *file1*, then *file2*, followed by all *.c* files. (To change files, enter **:w** followed by **:n**.) Filenames can contain any character except a slash (/); however, using metacharacters in filename may cause the inexperienced UNIX user some problems when trying to remove the file or even invoking *vi* on it.

Example 1:

Create a file named *practice* by typing:

vi practice

then press the **RETURN** or the **ENTER** key.



A blank screen appears with one line displayed at the bottom, indicating you have opened a new file. When *vi* loads your file or you create a new file, it is held in a buffer; you can consider this buffer a temporary storage place for your file until you save it. The buffer's (file's) contents display on the screen in a window. (The file is not modified until you write the buffer to the file.)

Modes of Operation

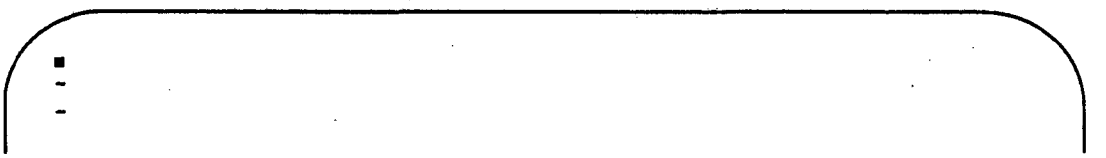
There are two modes in *vi*. When you invoke *vi*, the editor automatically starts up in command mode. This means you can enter commands that *vi* understands, but you cannot enter text. To enter text, you must put *vi* in text input mode. The two major methods of entering text are: appending and inserting. The following exercises illustrate how to use both command and text input modes.

Entering Text

Since this is a new file, you will be appending text. You must invoke the text input mode of *vi* to enter (append) text in a new file. Typing the character **a** (or **i**, it really doesn't matter when creating a file) invokes the text input mode and allows you to append (enter) text at the cursor position.

Example 2:

Type **a**:



NOTE: When you type *a*, nothing is displayed; you are ready to enter text.

Enter the text exactly as shown in Example 3, ending each line, except the last, with a RETURN. When entering the text, if you make typing errors, do not correct them. You can correct them as you try out the different commands in later exercises.

Example 3:

Type the following paragraph exactly as it appears (including errors).

> **The Convex C 1 64-bit supercomputeer system set a new standard for price/performance in scientific computer applications. Buy using a unique combination of widelyaccepted software and hardware standards, implemented with advanced technology, the Convex C 1 system offers the scientific user access to super computer performance at minicomputer prices. A well-balanced computer system, the convex C 1 compuetr blends high-speed 46-bit integrated scalar and vector processing with productivity oriented system software.■**

Obviously, this paragraph needs to be revised and corrected. The exercises in this tutorial show you how to make extensive revisions to this paragraph.

Fundamentals

Before you experiment with the commands in the following exercises, locate the *ESC* key and the *CTRL* key. These two keys and (primarily) the alpha-character keys are used for *vi* commands. The *CTRL* key is represented by a caret (^) throughout the document.

The *CTRL* key is used like a *SHIFT* key. For instance, a command using the *CTRL* key, e.g., ^*b*, means press and hold the *CTRL* key and simultaneously strike the *b* key. You may see ^*b* written as ^*B*; both of these are the same character and are typed the same way, using the lowercase character.

The *ESC* key causes *vi* to exit text input mode and return to command mode. When you press the *ESC* key, the cursor moves back to the last character you entered; this indicates you have returned to command mode. (If you are in command mode when you press the *ESC* key, you will hear a beep; don't worry, nothing happens to your document. The beep is *vi*'s way of telling you that you are already in command mode. When you aren't sure what mode you are in, press *ESC* until you hear the beep.) After you have completed adding text and wish to move through your document, you must return to command mode by pressing the *ESC* key. If you attempt to type text and no text is displayed, but you hear a beep, then you know you are in command mode. You may also hear a beep when trying to move your cursor to a position where there are no spaces or characters located, such as beyond the end of a line or the top or bottom of the buffer (file).

Aborting Commands

If you begin entering a command and wish to abort it, press the **ESC** key. If you make a typing error when entering text, you can backspace and typeover the text containing the error. In addition, typing **u** (when in command mode) reverses the effect of the last command that changed the buffer; typing **U** reverses the effect of all commands on the current line, if you have not moved from that line since the last change.

If you wish to exit an edit session without saving the revisions, type **:q!** when in command mode. *vi* returns you to the *shell* without saving any revisions to the file.

Clearing the Screen

To redraw your screen if it becomes garbled, type ^l (the letter 'el') while in command mode.

Saving a File

When you are in text input mode, you cannot save the contents of the buffer; you must be in command mode to invoke any of these commands.

Example 4:

Exit insert mode by pressing the:

ESC

key. Nothing is displayed on your screen when your press the *ESC* key; however, you can no longer enter text, but you can enter commands.

There are several commands that allow you to save the buffer contents to a file. *vi* recognizes the following commands for saving the buffer contents; you must press the RETURN key after all commands except *ZZ*:

Table 4-1: Saving Buffers

Command	Explanation
ZZ	Writes buffer contents, if modified; returns you to the *shell*
:x	Same as ZZ
:wq	Writes buffer contents to permanent storage; returns you to the *shell*
:w	Writes buffer contents to permanent storage (same file as entered) without exiting to *shell*; use this command as a precautionary measure against losing part of a file, as *vi* does not automatically make a backup file
:q	Allows you to exit the editor; used in conjunction with *:w*; typing *:w* followed by *:q* saves the buffer and exits to the *shell*; if a buffer has been modified, using *:q* is not permitted
:w *filename*	Writes buffer contents to permanent storage to new *filename* without exiting to *shell*; will not write to an existing file
:w! *filename*	Writes buffer contents to an existing *filename*; overwrites contents of *filename*
:w>> *filename*	Appends buffer contents to an existing *filename*
:*x,yth* **w** *filename*	Writes specified lines (*x,yth*) to *filename* without writing the edit buffer to the file
:q!	Exits the editor and aborts all modifications to the buffer (file)

Example 5:

Since you are now in command mode, you can save the contents of the buffer by typing:

:wq

and then pressing the **RETURN** key. (You can also save the file and exit by typing **ZZ**.) Your screen looks similar to this:

```
The Convex C 1 64-bit supercomputeer system set a new standard
for price/performance in scientific computer applications. Buy
using a unique combination of widelyaccepted software and hardware
standards, implemented with advanced technology,
the Convex C 1 system offers the scientific user access to
super computer performance at minicomputer prices. A
well-balanced computer system, the convex C 1 compuetr blends
high-speed 46-bit integrated scalar and vector processing
with productivity oriented system software.■
~
~
~
:wq
"practice" 9 lines, 520 characters
%■
```

Invoking *vi* on an Existing File

When you want to edit an existing file, you have several options for invoking *vi* and positioning the cursor. Normally, when you invoke *vi* (**vi** *filename*), the text displays on the screen with the cursor located on the first character of the first line. If you wish to begin editing at a specific location, you can use one of the following options:

vi +*n filename*	When the text appears on the screen, the cursor is positioned on the *n* line.
vi +*/string filename*	When the text appears on the screen, the cursor is positioned on the line containing the named *string*.

If the system crashes and you are editing with *vi* (or *ex*), you can find out if the file was saved by typing:

vi -r

A list of the saved files will be displayed on your screen. Then to restore (or edit) this file, type:

vi -r *filename*

Using the Arrow Keys for Cursor Movement

Your keyboard has four arrow keys that generally allow you to manipulate the cursor vertically and horizontally through the text; however, you must be in command mode when striking these keys. If you are in text input mode, each time you strike an arrow key, a capital letter appears on your screen. To exit input mode, press the *ESC* key.

Example 6:

Before continuing with the exercises, invoke *vi* on your *practice* file by typing **vi practice** and pressing the **RETURN** key.

```
% vi practice
The Convex C 1 64-bit supercomputeer system set a new standard
for price/performance in scientific computer applications. Buy
using a unique combination of widelyaccepted software and hardware
standards, implemented with advanced technology,
the Convex C 1 system offers the scientific user access to
super computer performance at minicomputer prices. A
well-balanced computer system, the convex C 1 compuetr blends
high-speed 46-bit integrated scalar and vector processing
with productivity oriented system software.
~
~
~
"practice" 9 lines, 520 characters
```

Try the arrow keys to see if you can move the cursor; the cursor should now move through the text. If capital letters appear on the screen, you are in text input mode; to exit, press the **ESC** key which returns you to command mode. Then press the arrow keys. If you hear a beep, it means that you cannot move any further in that direction.

Basic Commands

The following sections contain general information and hints for using the fundamental *vi* commands. The commands described in this section provide the foundation for using *vi* efficiently. In later sections, additional commands for the more experienced user are illustrated.

Using Cursor Movement Commands

In addition to the arrow keys, several commands allow you to move the cursor. The following table describes the basic commands for *vi* cursor movement, as well as screen scrolling commands. When you use the scrolling commands, a new line moves up or down into view. Scrolling makes the top line leave the screen as a new line is displayed at the bottom, or the bottom line leaves the screen as a new top line is displayed. If you give the command to scroll down, the screen (window) moves downward from your current position; it exposes one or more lines at the bottom of the screen. Conversely, scrolling up exposes one or more lines at the top of the screen.

Some of the scroll commands require a screenful of text to execute properly; if you have less than a screenful displayed, you will probably hear a beep when executing the command(s).

The movement commands can be prefixed with a number. For example, **4b** would move the cursor back four words; using **5j** would move the cursor down five lines, as would typing **5** followed by a RETURN.

You can use many of the movement commands in conjunction with the change and delete commands, which are illustrated in later sections.

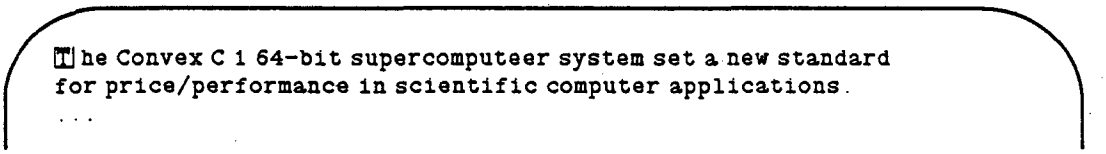
Table 4-2: Movement Commands

Command	Explanation
h	Moves left one character within current line
-	Moves left one character within current line
l	Moves right one character within current line
+	Moves right one character within current line
SPACE bar	Moves right one character within current line
BACKSPACE key	Moves left one character within current line
b	Moves left (back) to beginning of word or to punctuation mark
f*c*	Moves to specified characters *c*
t*c*	Moves to space preceding specified character *c*
B	Moves left (back) to beginning of word (ignores punctuation)
w	Moves right to beginning of next word or to punctuation mark
W	Moves right to beginning of next word (ignores punctuation)
e	Moves right to end of current word or next word, depending on cursor position when you invoke the command
0 (zero key)	Moves to the first character of the line (left)
^	Moves to first non-white-space character (^ is usually shift-6)
$	Moves to the last character on the line
j	Moves to following line (down) in the same column
↓	Moves to following line (down) in the same column
+	Moves to the beginning of the following line (down)
RETURN key	Moves to beginning of next line (down)
↑	Moves to previous line (up)
-	Moves to beginning of previous line (up)
k	Moves to previous line (up) in the same column
)	Moves to beginning of next sentence
(	Moves to beginning of previous sentence
1G	Moves to beginning of current buffer
G	Moves to beginning of last line of the current buffer
*n***G**	Moves to line number (*n*) specified
n|	Moves to column *n* on current line
{	Moves to beginning of previous paragraph
}	Moves to end of current or next paragraph
^y	Jumps down one line
^e	Jumps up one line
^f	Scrolls forward one screenful
^b	Scrolls backward one screenful
^d	Scrolls forward one-half page
^u	Scrolls backward one-half page
H	Moves cursor to top left of screen (home)
M	Moves cursor to middle of screen
L	Moves cursor to bottom of screen

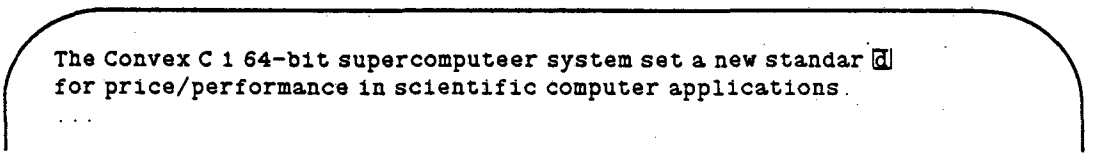
Example 7:

Be sure you are in command mode before attempting the following exercises. If you are in text input mode, pressing **ESC** returns you to command mode. Once you are in command mode, you can type the commands in succession. (The rectangle represents the placement of your cursor **after** you have typed the command.)

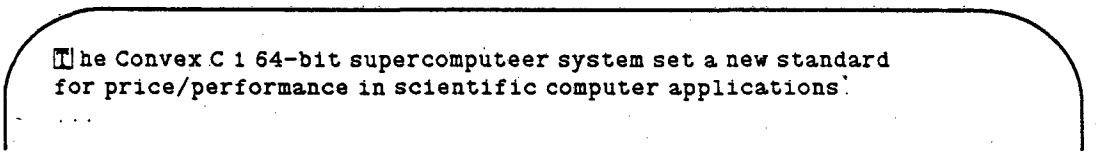
First move your cursor to the top of the screen by typing **H**.



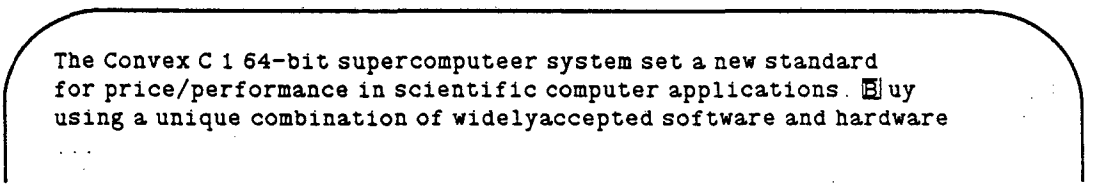
Move to the last character of the first line (*d*) by typing $.



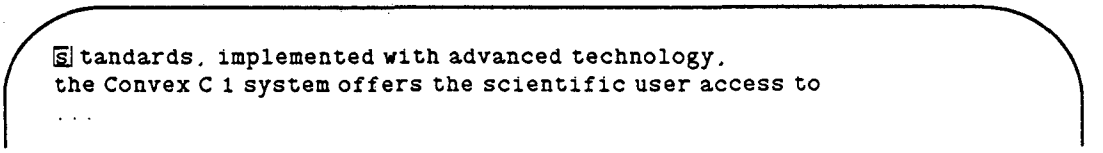
Type **0** to return to the beginning of the line (*T*).



Now move to the beginning of the next sentence (*B*) by typing).



Next, move the cursor to the first *s* on *standards* by pressing the **RETURN** key two times:



Now move forward one word by typing **w**.

```
standards □ implemented with advanced technology.
the Convex C 1 system offers the scientific user access to
...
```

Notice how the cursor moves to the , instead of the next word; the punctuation is considered a new word. Using *W* instead of *w* causes the cursor to move to the next word i.e., *i* on "implement", skipping the punctuation mark.

Finally, move the cursor to the beginning of the buffer (*T*) by typing **1G**.

```
□he Convex C 1 64-bit supercomputeer system set a new standard
for price/performance in scientific computer applications.
...
```

Continue experimenting with the commands shown on the chart. When you feel comfortable with the movement commands, proceed to the section "Using Delete Commands."

Using Deletion Commands

Corrections can be made in different ways depending upon an individual's preference; there really is no "right" or "wrong" way to make a correction.

A majority of the deletion commands are a combination of the movement commands, which you studied in the previous section, and the *d* command. For example, to delete a word use **dw** or use **d0** to delete from the current position to the beginning of line. The following table and exercises illustrate how to combine the movement commands and deletion commands to make corrections. You can experiment with the commands by revising the paragraph entered for Example 3.

There are several commands that delete characters, words, and phrases that can be combined with movement commands. These commands include:

Table 4-3: Delete Commands

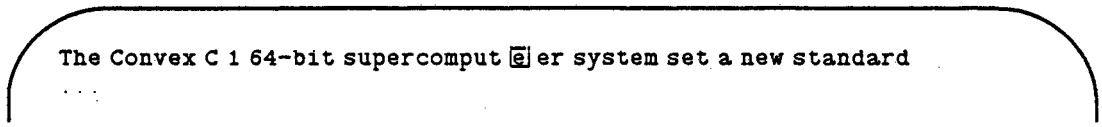
Command	Explanation
x	Deletes character at current cursor position
X	Deletes one character left of the cursor position
***n*x**	Deletes character at current cursor position and next *n* characters
D	Deletes beginning at cursor position to end of line
dd	Deletes the current line of text or blank line
***n*dd**	Deletes *n* lines, including line of current cursor position
:*x*,*y*thd [RETURN]	Deletes the range of lines specified with *x*,*y*
dw	Deletes one word right (does not delete punctuation following the word)
dW	Deletes one word right, including punctuation following the word
d0	Deletes from beginning of line to current cursor positon
dH	Deletes from cursor position to top of screen
dM	Deletes from cursor position to the middle of the screen (full screen—deletes up to 12 lines)
dL	Deletes from cursor position to bottom of screen
dG	Deletes from cursor position to end of file
d*n*G	Deletes the specified line *n*
d)	Deletes from cursor position to end of sentence
d(	Deletes from cursor position to beginning (left) of sentence
db	Deletes one word or one punctuation mark left of cursor position
dB	Deletes one word left from cursor position, including accompanying punctuation mark;
df*c*	Deletes from cursor position to specified character (*c*)
dt*c*	Deletes from cursor position up to specified character (*c*)
d/*string* [RETURN]	Delete characters beginning at cursor position through specified *string*

Note: When the movement commands are used with the delete command, they can be prefixed with numbers. For example, typing **2dw** will delete two words; likewise, **2d)** deletes two sentences (beginning at cursor position.)

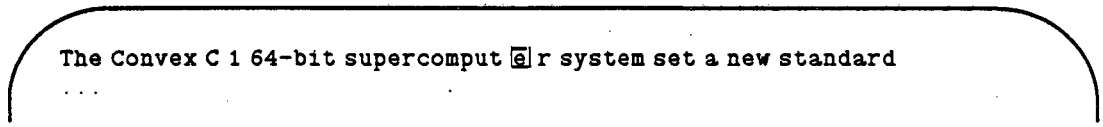
Example 8:

You must be in command mode to execute these commands. If text appears instead of a command being executed, press the **ESC** key. Experiment with the delete commands by revising the paragraph entered for Example 3, using the following illustrations as a guide for revisions. (The rectangle represents the cursor position).

Delete the extra *e* in *supercomputeer* using the *x* command. First move the cursor to the *e* that needs to be deleted; one way to do this (assuming your cursor is at the top of the screen, if it is not, type **H**) is to type **W** four times (or **4W**) and strike the **SPACE** bar until your cursor is on *e*:

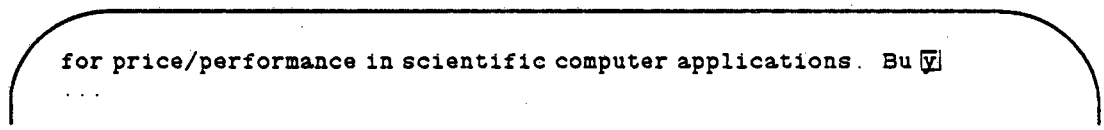


Now type **x**; the *e* is deleted.

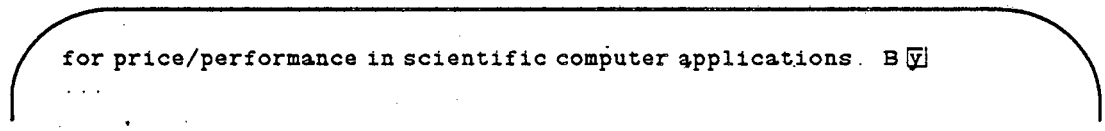


Delete the *u* in "Buy" using one of the following methods for deletion: the *x* command, or the *X* command. To use the *x* command, move the cursor to the *u* and type the command **x**.

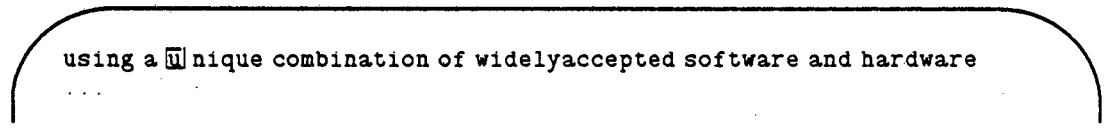
To use the *X* command, position the cursor on *y* by typing **j** and **$**:



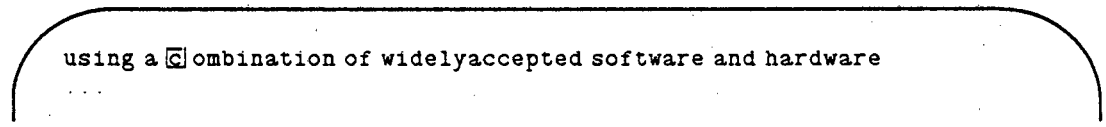
and type **X**:



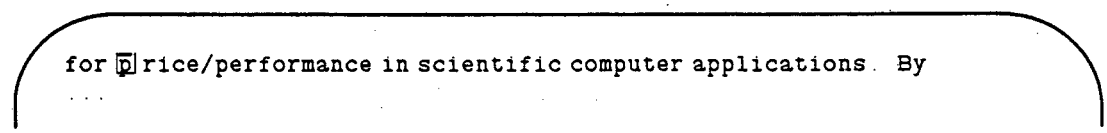
Delete the word *unique* using the *dw* command. Move the cursor to the *u* in "unique" (type **j** followed by **7B**):



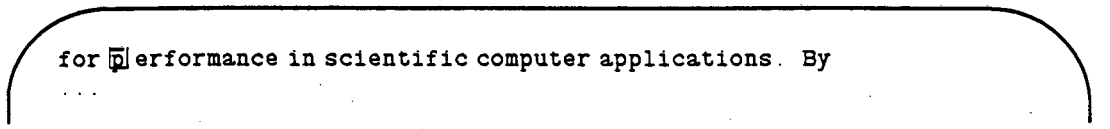
then type **dw**:



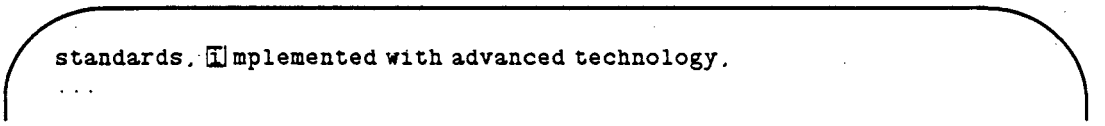
Delete *price/*; there are several ways you can accomplish this task. However, for this exercise, use the *nx* command. First move the cursor to *p* (type **k** and press **BACKSPACE** (or type **b**) until your cursor is on *p*):



Now type **6x** (or **xxxxxx**):



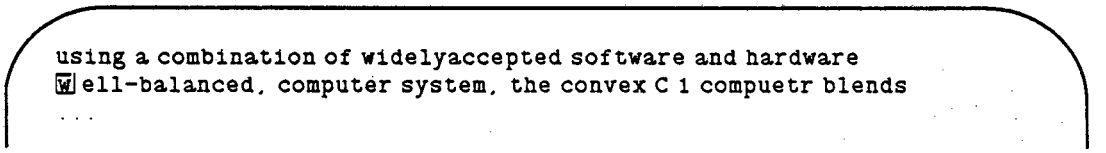
This time delete the phrase *implemented with advanced technology* by using the *D* command. Position the cursor on the *i* (type **2j** followed by **W**):



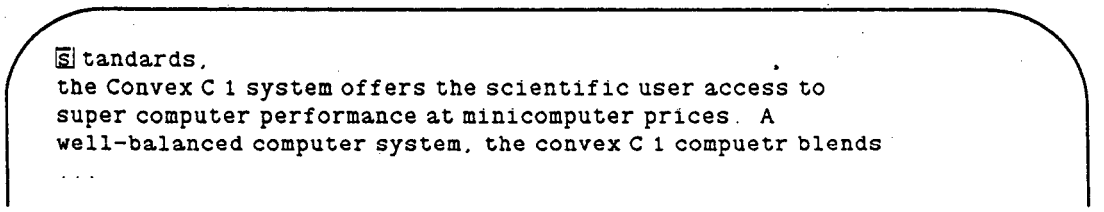
Enter the command **D**:

Don't worry about the blank space remaining on the line following the comma. If you had deleted the entire line (including the word *standards*) using the *D* command, a blank would remain where the line was deleted. In this instance, you could delete the entire line by using the *dd* command. To delete an entire line of text and leave no blank line, *dd* is more efficient.

This time delete the next 3 lines and then use the *undo* command to restore these lines. Enter the command: **3dd**



Now restore these lines by typing **u**:



If you wish to experiment with additional deletion commands or practice using the commands covered thus far, first save the current contents by typing **ZZ**. Since this file is needed for later exercises, copy it to a new file named *junk*—**cp practice junk**. Use this *junk* file for additional revision practice. When you feel comfortable with the deletion commands, proceed to the next section, "Inserting Text."

Inserting Text

vi provides commands that allow you to insert text only or blank lines and text. When you use any of the insertion commands, you must type **ESC** after entering your text to return to command mode.

The commands that allow you to insert characters in *vi* are:

Table 4-4: Insert Commands

Command	Explanation
i	Places you in text input mode; insertion occurs before current cursor position; press ESC to return to command mode
I	Moves cursor to beginning of current line and enters text input mode at that position; press ESC to return to command mode
o	Opens (inserts) a blank line following the current line, moves cursor to beginning of that new line, and enters text input mode at that position; press ESC to return to command mode
O	Opens a blank line before the current line, moves cursor to beginning of that new line, and enters text input mode at that location; press ESC to return to command mode

The *si* Option

vi has several options available that allow you to customize the editor to fit your needs. These options can be placed in the *.exrc* file or supplied while you are running *vi*. Those options that you give within *vi* are available only during the current edit session and must be reset each time you enter *vi*. However, setting options in the *.exrc* makes them available during all *vi* sessions. In a later session, you will set *vi* options in your *.exrc* file.

Example 9:

If you are not currently in the *practice* file, invoke *vi* on it.

One of the *vi* options is helpful when you are in text input mode. The *si* option notifies you when the editor is in text input mode by displaying *I* at the bottom of the screen in reverse video. Set this option by typing:

:set si

and pressing the **RETURN** key.

When you make this entry, your screen looks like this:

```
The Convex C 1 64-bit supercomputer system set a new standard
for performance in scientific computer applications. By
...
...
~
~
:set si
```

Example 10:

You are now ready to begin the following exercises. Each time you begin inserting text, *I* is displayed at the bottom of your screen. When you end text input mode by pressing the *ESC* key, the *I* disappears from the screen, indicating that *vi* is now in command mode.

Insert a comma after *high-speed*. Assuming the cursor location is at the beginning of the file (if it isn't, type **1G**), type **7** and press the **RETURN** key, which places the cursor on the *h*. Now move the cursor to the space immediately following the *d* by pressing the **SPACE** bar until the cursor is placed on the space:

```
high-speed 46-bit integrated scalar and vector processing
...
```

Type **i** to allow insertion, followed by the character to be inserted **,** (comma):

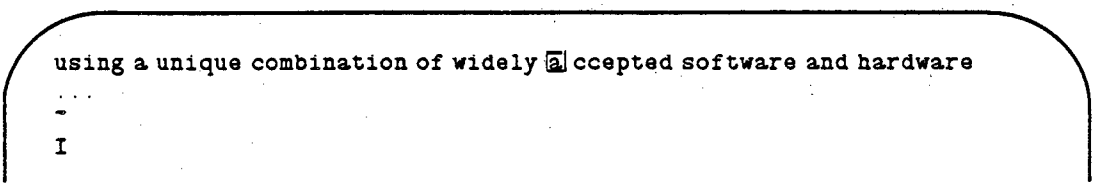
```
high-speed, 46-bit integrated scalar and vector processing
...
~
I
```

Now press the **ESC** key to return to command mode. (If you attempt to move the cursor without being in command mode, characters will appear on the screen.)

Add a space between *widely* and *accepted*. Position the cursor on the *a* in *accepted*; you can move up by typing **5** and pressing the **up arrow key**, typing **5k**, or by typing **5-**. Next press the **SPACE** bar until you move to *a*:

```
using a unique combination of widely accepted software and hardware
...
```

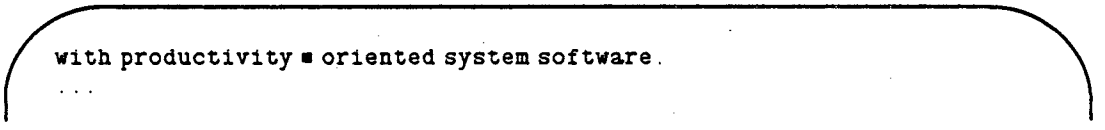
Type **i** to allow insertion; press the **SPACE** bar one time.



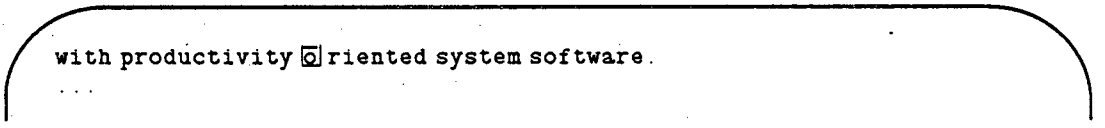
Be sure to return to command mode by pressing the **ESC** key.

Now insert a hyphen after *productivity*. To complete this revision, you must eliminate the space and insert the hyphen (no spaces are used in hyphenated words). Use one of the commands previously discussed to delete the extra space and then insert the hyphen. (This revision can also be made using the *r* command, which is discussed in the section entitled "Using Change Commands.")

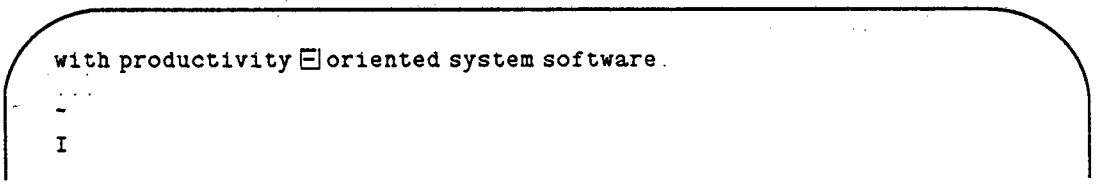
First move the cursor to the space following *y*. Use any of the previously discussed commands to move the cursor to the space. (You can move to the end of the paragraph by typing } and then move back to the space following *y* by depressing the **BACKSPACE** key.)



Type *x*:



Then type **i** to allow insertion, followed by the hyphen (-).



Again, end by pressing the **ESC** key.

Begin a new paragraph at *A well-balanced.* Position the cursor at *A*. One way to move the cursor: type **3k** followed by **$**.

Now type **i**:

```
super computer performance at minicomputer prices. A
well-balanced computer system, the convex C 1 compuetr blends
...
-
I
```

Then press the **RETURN** key two (2) times followed by **ESC**:

```
super computer performance at minicomputer prices.

A
well-balanced computer system, the convex C 1 compuetr blends
...
```

Suppose you need to insert a line or several lines of text between other lines of your document. The *o* commands enable you to insert a blank line with the cursor remaining at the beginning of the blank line so you are able to begin typing on that line.

Put a blank line at the beginning of the file; first move the cursor to the line that begins with *The Convex C 1 64-bit supercomputer* by typing **1G**.

```
The Convex C 1 64-bit supercomputer system sets a new standard
...
```

Now type **O**:

```
■
The Convex C 1 64-bit supercomputer system sets a new standard
...
-
I
```

A blank line appears above *The*, and the cursor is at the beginning of the blank line. You can now begin entering text in the normal manner. To return to command mode, press the **ESC** key. (Delete this extra line using the *dd* command.)

Appending Text

The append commands are similar to the insert commands, except append commands allow you to append text following the cursor position; whereas, insert commands allow insertion preceding the cursor position. Again, as with insertion, you must press the **ESC** key to end text input mode and return to command mode. Also, you will notice the reverse video *I* displays at the bottom of the screen each time you enter one of the append commands, if you have set *si*. If you wish to append text, you can use one of the following commands:

Table 4-5: Append Commands

Command	Explanation
a	Places you in text input mode; appends text after the current cursor position; press ESC to return to command mode
A	Moves cursor to end of current line and enters text input mode at that location; press ESC to return to command mode

Example 11:

Be sure to press the *ESC* key at the end of each of the following exercises. You cannot enter any commands when you are in text input mode.

Append an *s* to the word *set*; move the cursor to the *t* on *set* by pressing the **SPACE** bar or by pressing the right **arrow** key until the cursor is on the *t*; or type either **W** six times or type **6W** and press the **SPACE** bar until the cursor is on the *t*)

```
The Convex C 1 64-bit supercomputer system se t a new standard
...
```

Now type **a** (to append) and **s** followed by **ESC**:

```
The Convex C 1 64-bit supercomputer system set a new standard
...
```

Now append the phrase *emplemented with advanced technology,* (include the typing error) following *standards,*. Move the cursor to the line beginning with *standards* (type **3** and press the **RETURN** key, press **RETURN** 3 times, or type **3+**):

```
s tandards,
...
```

Now type **A** followed by **emplemented with advanced technology,**; then press **ESC**:

```
standards, emplemented with advanced technology□
...
```

Using Change Commands

vi provides several commands that allow you to change characters, words, and lines. These commands, which generally place you in the text input mode, allow you to typeover existing text or, in some cases, delete the existing text and insert new text. You will notice that a $ appears on the screen following the text when you type some of these commands. The $ indicates the final character that you can replace, or if not replaced, what will be deleted when *ESC* is pressed. Return to command mode by pressing the *ESC* key.

vi recognizes the following change commands. Any text included in the command but not changed is deleted when you leave text input mode.

Table 4-6: Change Commands

Command	Explanation
r	Deletes only the character at current cursor position and allows you to insert one character; you are automatically returned to command mode; no ESC is necessary
cw	Replaces word at cursor position and enters text input mode; press ESC to return to command mode; any characters not changed are deleted when ESC is pressed
ct*c*	Replaces text beginning at cursor position up to specified character and enters text input mode; press ESC to return to command mode; any characters not changed are deleted when ESC is pressed
cf*c*	Replaces text beginning at cursor position through specified character and enters text input mode; press ESC to return to command mode; any characters not changed are deleted when ESC is pressed
C	Allows overtyping of line beginning at cursor position; press ESC to return to command mode; any characters not changed are deleted when ESC is pressed
s	Replaces the character at the current cursor position and enters text input mode; press ESC to return to command mode
*n*s	Replaces next *n* characters (includes character at the current cursor position) and enters text input mode; press ESC to return to command mode
~	Changes case of character at current cursor location
R	Allows you to replace all characters beginning at cursor position until you press ESC; at end of lines, inserts additional text

Change Commands (cont.)

Command	Explanation
xp	Transposes two letters; swaps letter at cursor position with following character
ddp	Reverses order of two lines; reverses current line with following line
c/*string* [RETURN]	Allows you to replace text beginning at current cursor position to specified *string*; press ESC to return to command mode; any characters not changed are deleted when ESC is pressed

You can prefix *r* and *cw* with numbers to make multiple changes. Using **3r** allows you to change three characters; using **2cw** allows you to change two words.

Example 12:

Using the *r* command, change the *e* in *emplemented* to *i*. First move the cursor to the *e*; if the cursor is located on the comma following *technology*, move the cursor by typing **4b**; otherwise, use the appropriate command to move the cursor (be sure you are in command mode):

```
standards, emplemented with advanced technology,
...
```

Then type **r** followed by **i**:

```
standards, implemented with advanced technology,
...
```

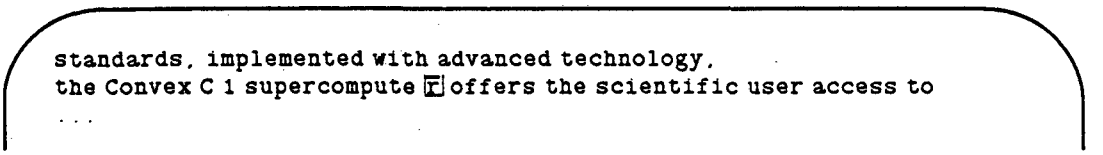
Change the word *system* to *supercomputer*. Move the cursor to *s*; one way to move the cursor is to type **j** followed by **W**.

```
standards, implemented with advanced technology,
the Convex C 1 system offers the scientific user access to
...
```

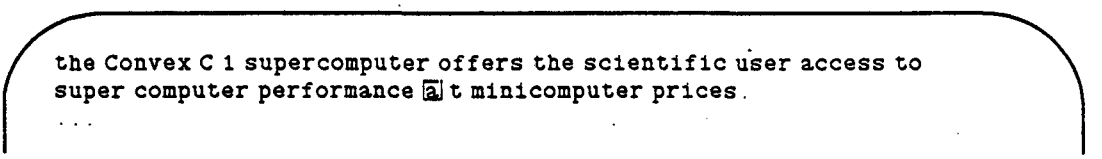
Type **cw**:

```
standards, implemented with advanced technology,
the Convex C 1 syste$ offers the scientific user access to
...
-
I
```

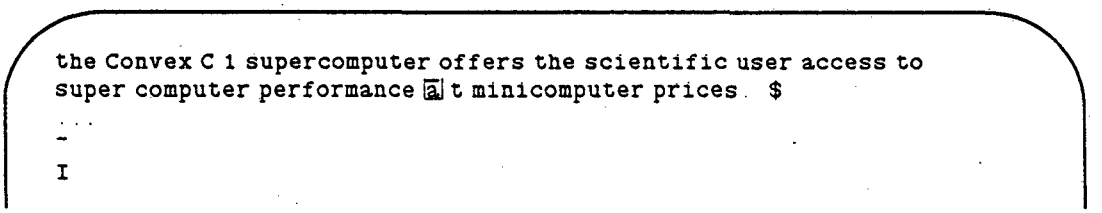
Notice how the $ is displayed in place of *m*; it marks the last character to be changed or deleted. Now type the characters **supercomputer** and press the **ESC** key.



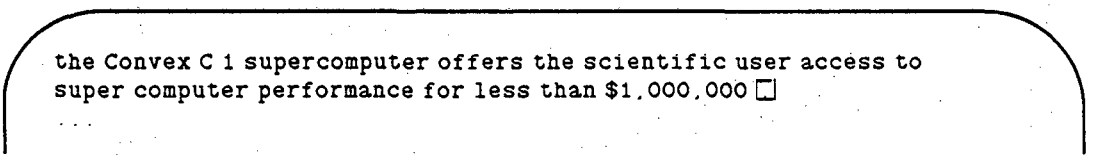
Change the phrase *at minicomputer prices* to read **for less than $1,000,000.** First move the cursor to the *a* in *at*; you can move the cursor by typing **j**:



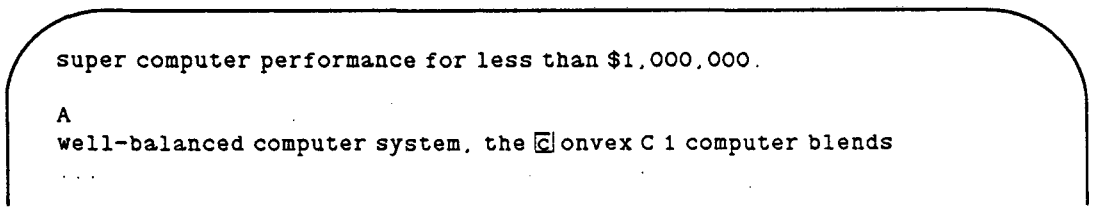
Next specify "change to end of line" by typing **C**.



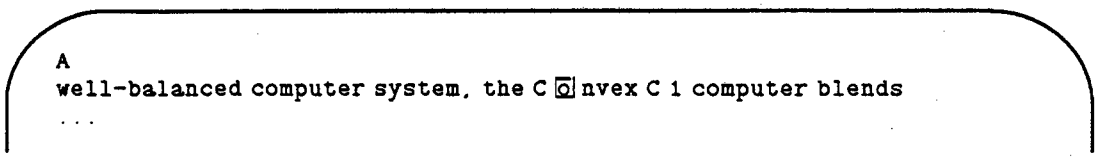
Again, the $ marks the last character to be changed or deleted. (Your screen may have the $ placed differently. If you did not supply spaces after the period, it will be at that position.) You can now type the change **for less than $1,000,000.**; press the **ESC** key to return to command mode.



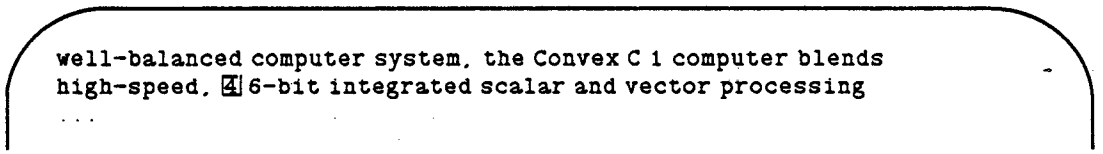
Capitalize the *c* in *convex* which appears in the line *the convex C 1 computer blends.* To move to the *c*, press the **RETURN** key three times and type **4W**:



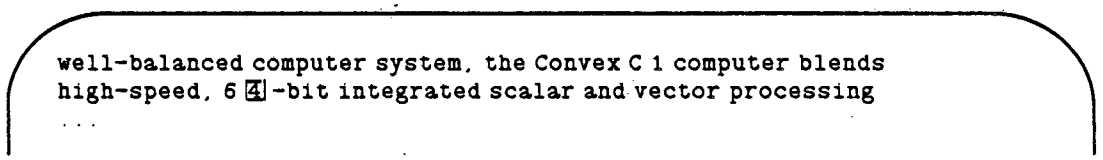
Now capitalize the letter by typing ~:



Now change *46-bit* to *64-bit*. Move the cursor to the *4* by pressing the **RETURN** key and typing **W**:



Next transpose by typing **xp**:



The change command can be used in combination with other movement commands in the same manner that delete commands are used with them. The following commands delete the text and also allow for insertion, i.e., a combination of the delete and insert commands. When you review the following table, you will see that the commands are built from the change and movement commands.

Table 4-7: Combining Commands

Command	Explanation
c0	Deletes from current cursor position to beginning of line and enters text input mode; press ESC to return to command mode
cH	Deletes from cursor position to top of screen and enters text input mode; press ESC to return to command mode
cM	Deletes from cursor position one-half of the lines displayed on the screen and enters text input mode; press ESC to return to command mode
cc or **S**	Deletes the current line (cursor can be at any position in the line) and enters text input mode; press ESC to return to command mode
cL	Deletes from cursor position to bottom of screen and enters text input mode; press ESC to return to command mode
cG	Delete from cursor position to end of file and enters text input mode; press ESC to return to command mode
c)	Deletes from cursor position to end of sentence and enters text input mode; press ESC to return to command mode
c(	Deletes from cursor position to beginning of sentence and enters text input mode; press ESC to return to command mode
c/*x* [RETURN]	Deletes from cursor position to specified *x* and enters text input mode; press ESC to return to command mode
c}	Deletes from cursor position to end of current paragraph and enters text input mode; press ESC to return to command mode
c{	Deletes from cursor position to beginning of current paragraph and enters text input mode; press ESC to return to command mode

Example 13:

Change the first line of the first paragraph to read:

> **The Convex supercomputer offers the design engineer analysis power at price and performance levels new to the industry for increased and improved produtcivity.**

First move to the first line of the paragraph by typing **1G**:

```
The Convex C 1 64-bit supercomputer system sets a new standard
for performance in scientific computer applications. By
~
~
```

Now use the command that allows you to delete the first line and insert text; type **c)**:

```
By
~
~
I
```

Now enter the text and then press **ESC** to return to command mode:

```
The Convex supercomputer offers the design engineer analysis
power at price and performance levels new to the industry for
increased and improved productivity. ■By
...
```

Now replace the last paragraph with the following:

This high-performance, low-cost system is now being used by engineers for structural, thermal, fluid, electrical, and static electromagnetic analysis.

First move to the second paragraph by typing **6** and pressing the **RETURN** key.

```
increased and improved productivity. By
....
A
```

Now change the last paragraph by typing **c}**:

```
super computer performance for less than $1,000,000.

■
-
I
```

Insert the text and conclude the entry by pressing the **ESC** key:

```
super computer performance at minicomputer prices.

This high-performance, low-cost system is now being
used by engineers for structural, thermal, fluid, electrical,
and static electromagnetic analysis
```

If you wish to spend more time experimenting with the change commands, save this file by typing **ZZ**; then edit the *junk* file that you created previously. If you did not create this file previously, do so by typing **cp practice junk**. When you feel comfortable using the previous commands, proceed to the next section, "Editing Exercises."

Editing Exercises

If you have not saved the *practice* file, save it by typing **ZZ** (in command mode).

The following exercises will allow you to practice *vi* editing techniques. For this exercise, either copy the file specified by your instructor to your home directory or create a file named *edit1* containing the following text, including the errors:

SEar Student,

By now you should be able to create a file,
edit the file using the delete, change, insert,
append commands.

Now to shoow everyone your new skills, edit
this file according to the insturctions in
your tutorial.

Sincerely,

Your Instructor

If you created *edit1* (didn't copy it), write the file before making the following modifications. Revise the *edit1* file; make the following changes:

1. Change *SEar* to **Dear**.
2. Change *Student* to your first name.
3. Change the comma that follows your name to a colon (:). If you have deleted the comma, add a colon after your name.
4. Delete one of the blank lines preceding the first paragraph.
5. Delete the comma in the first sentence following *file*.
6. Append a **space** and **and** after *file* in the first sentence.
7. Insert **and** at the beginning of the third line of the first paragraph.
8. Delete the extra **o** in show in the first line of the second paragraph.
9. Transpose the *ur* in *insturctions*.
10. Insert two blank lines before the line *Your Instructor*.
11. Write the modified contents to a file named *edit1.rev*.
12. Exit the current buffer (file) without saving any of the modifications; abort the changes.
13. Print the *edit1* file and the *edit1.rev* file and give the printed copies to your instructor.

If you feel comfortable using the commands discussed to this point, procede to the next section. Otherwise, review the preceding examples for commands you feel uncomfortable using.

Using the Search Commands

Search commands are a time-saving tool for making revisions to an existing document or to locate a particular item in a document/program. *vi* has several search commands that aid you in locating a particular word, phrase, or string in a document or program. However, all of the commands are based on entering a search string.

To use these commands from command mode, type / followed by the string of characters to search for and press the **RETURN** key. The search begins at the cursor position—the cursor moves to the first occurrence of the string. At this point you can make changes, move text, add text, delete text, etc. If you wish to move to another occurrence of the same string, type **n** to continue the search forward; **N**, to search backward. As long as you enter **n** or **N**, *vi* keeps searching for the string in a wrap-around fashion—searches continue to the end of the file (either direction) and begin again. (You may optionally disable this feature by setting *nowrapscan*; see the section entitled "Customizing the *vi* Environment" for more information.)

Since matches are made according to the string entered, you may find a short string matching longer strings. For example, when you enter a search string of "in", it matches all strings

containing "in". If you want a match for just the characters entered as the string, be sure to include a space preceding and following the string. Thus, to match only "in" and nothing else, enter **in** (strike the space bar before and after typing the *n*). In addition, if you are searching for characters that were entered in capitals, be sure you enter the search string in capitals.

The following search commands are available:

Table 4-8: Search Commands

Command	Explanation
/string	Searches forward from cursor position for *string*
?*string*	Searches backward from cursor position for *string*
?^*string*	Searches backward from cursor position for *string* at beginning of line only
/string$	Searches forward for *string* at ending line of only
/^string	Searches forward for *string* at beginning line of only
n or **N**	Repeats search in same direction (*n*) or opposite direction (*N*); does not quit the search at end of file—wraps around and continues the search
/[RETURN]	Repeats search forward; does not quit the search at end of file—wraps around and continues the search
?[RETURN]	Repeats backward search; does not quit the search at beginning of files—wraps around and continues the search
f*x*	Finds character *x* on current line; to repeat search for same character use the semicolon (;); use a command (,) to reverse the search
F*x*	Finds previous character *x* on current line; to repeat search for same character use a semicolon (;); use a comma (,) to reverse the search
T*x*	Moves backward to character before specified *x* on current line; to repeat search for same character use a semicolon (;); use a comma (,) to reverse the search
t*x*	Moves forward to character before *x* on current line; to repeat search for same character use the semicolon (;); use a comma (,) to reverse the search
%	Moves to matching *()*, {}, or *[]*; when command is invoked, cursor must be positioned on one of these characters

Example 14:

The following exercises allow you to experiment with the search commands.

To continue the exercises, you must use the *practice* file; enter **vi practice**, if appropriate. Remember *si* is no longer set; if you wish to set it, from within *vi* type **:set si** followed by a **RETURN**.

Before beginning these exercises, move the cursor to the beginning of the file (if it is not there) by typing **1G**:

```
The Convex supercomputer offers the design engineer analysis
power price and performance levels new to the industry for
```

Now search for *super* by typing **/super** and then pressing the **RETURN** key:

```
The Convex supercomputer offers the design engineer analysis
...
~
~
/super
```

Continue the search until the cursor returns to the *s* in *supercomputer* in the first line of the first paragraph by typing **n** until the cursor moves to the *s*:

```
...
the Convex C 1 supercomputer offers the scientific user access to
...
~
/
```

Notice the *n* does not appear on the screen when you type it, but the next search is completed. This time when you type **n**, the cursor moves to the next location:

```
...
super computer performance for less than $1,000,000
...
~
/
```

Type **n** yet another time; notice even though there are no more *super*'s in the file, when you typed **n** again, *vi* continued searching, using the "wrap around" feature, and found the first occurrence of the string:

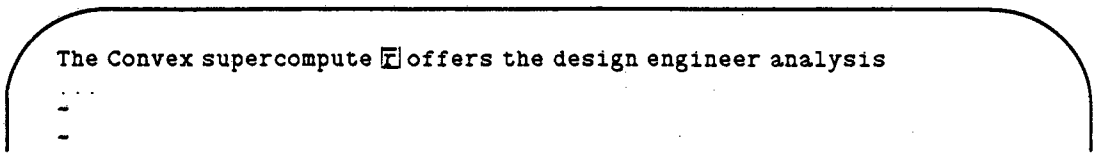
```
The Convex supercomputer offers the design engineer analysis
...
~
/
```

For this exercise, find all occurrences of *r* on the first line of the first paragraph. Type **fr**:

```
The Convex supe rcomputer offers the design engineer analysis
...
~
~
```

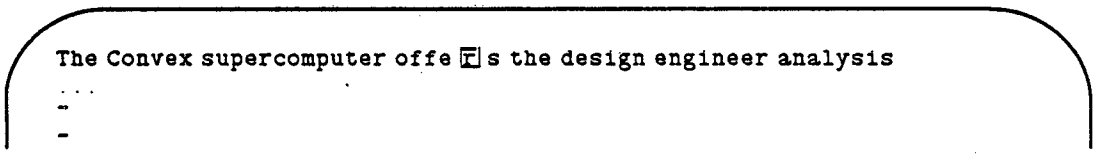
Notice the command does not display on the screen, but the cursor does move to the specified location.

Now type ; (semicolon), which repeats the search:

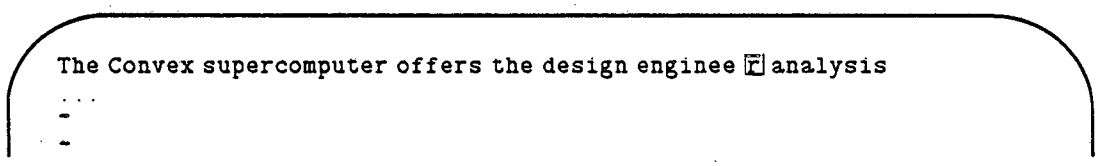


Again, notice that the repeat command does not display on your screen.

Repeat the command (;):

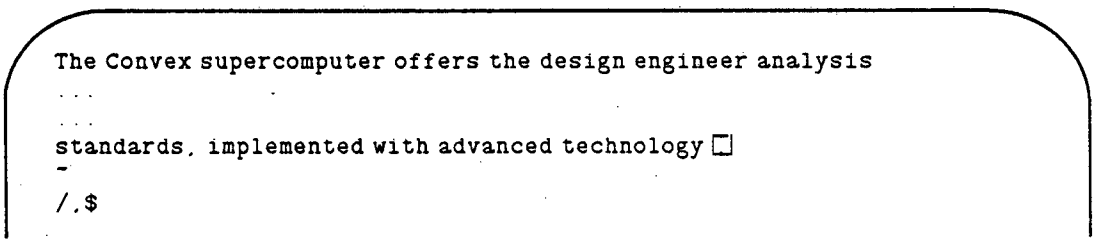


Repeat the command (;):



This time you should hear a beep when you type ;.

For this exercise, search for a comma at the end of a line. Type /,$ and then press the **RETURN** key:



Note: If you have inserted a space after the comma following *technology,* the search will not find the comma.

Now experiment with the other search commands. When you feel comfortable using these commands, proceed to the next section, "Searches Using Regular Expressions."

Searches Using Regular Expressions

vi normally interprets search strings as regular expressions. Regular expressions are a set of special characters which are interpreted according to special rules. The following table illustrates how to use regular expressions in searches.

Table 4-9: Using Expressions in Searches

Command	Explanation
/\c	Backslash (\) must precede a special character, i.e., \, $, &, ., ~, and |, in searches
/[ccc]*string*	Matches single character or range in the bracketed list; **[Tt]he** matches *The* or *the*; **[l-s]end** matches *lend, mend,* or *send*
/(*string*).(*string*)	Dot matches any single character preceded or followed by *string*; **.he** matches the, she, these, etc.; **s.nd** matches *send* or *sand*
/(*string*)*(*string*)	Matches zero or more occurrences of the preceding character; **Pr*** matches any string containing *P* followed by zero or more occurrences of *r*
/[^...]*string*	Matches any characters not in the bracketed list
/\>*string*	Matches *string* only at end of word
/\<*string*	Matches *string* only at beginning of word

Note: If the regular expressions do not work when you do searches, then you must set *magic*. (See the section entitled "Customizing the *vi* Environment" for additional information.) To set the *magic* option, when you are in *vi*, type:

:set magic

followed by a **RETURN**.

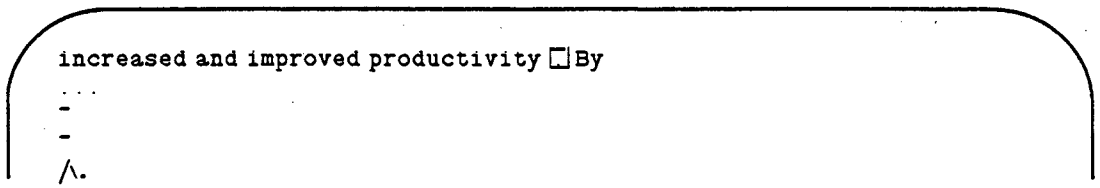
Example 15:

The following exercises illustrate how to use regular expressions in searches. You may find these regular expressions helpful in later exercises requiring searches and substitutions. You can experiment with these special expressions using the *practice* paragraph.

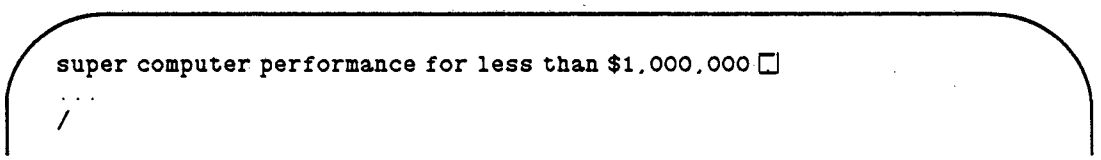
If you are not currently in the practice file, invoke *vi* on it. Move the cursor to the beginning of the file (if it is not there) by typing **1G**.

```
The Convex supercomputer offers the design engineer analysis
...
```

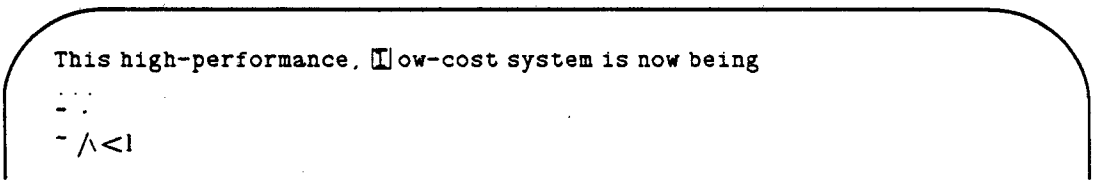
Search for the special character . (dot). Remember when searching for a special character, you must precede it with a \. To invoke this search, type /\. and press the **RETURN** key:



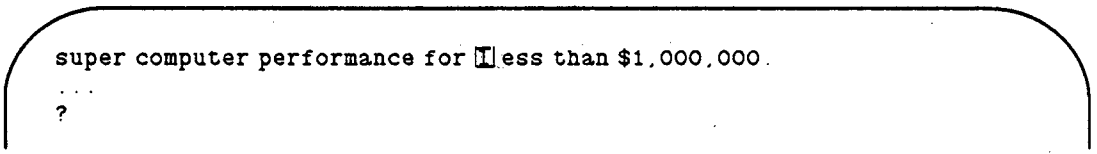
To repeat the search, type **n**:



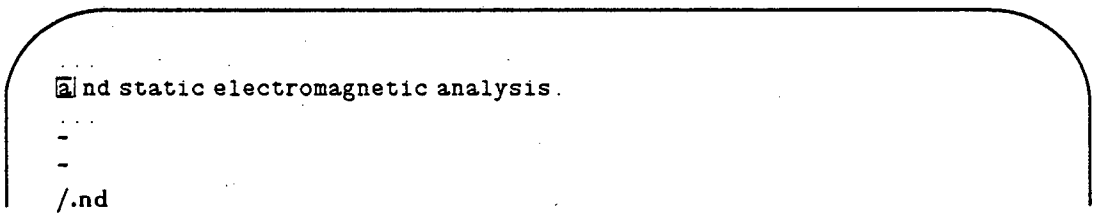
Assume you want to search for all the words that begin with *l*. Enter the command /\<l:



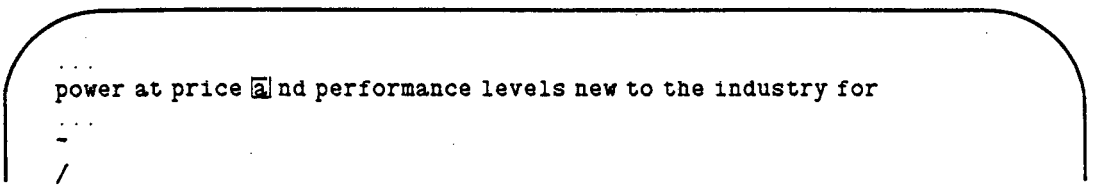
Repeat the search in the opposite direction by typing **N**:



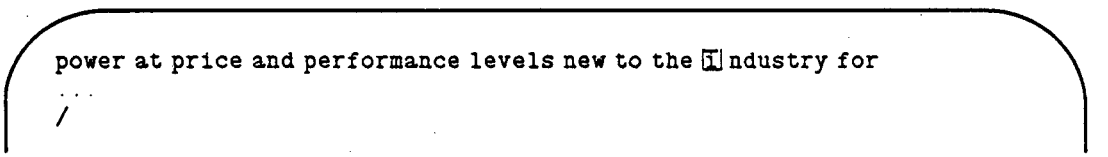
Assume for this exercise that you want to find all words that contain any character followed by *nd*. To do this search, type /.nd and press the **RETURN** key:



Repeat the search by typing **n**:



Repeat the search by typing **n**:



Continuing experimenting with special expressions in searches. When you feel comfortable using these commands, start on the next section of the tutorial, "Making Substitutions."

Making Substitutions

vi provides a substitution command to automatically find and replace text. There are several variations of this command. The following paragraphs and exercises will show you how to use this command to find and to replace one or more occurrences of a word or phrase.

For instance, if you wanted to substitute one specific occurrence of *Jane* with *Sally*, you would first move your cursor to that line and then enter the command:

:s/Jane/Sally/

and press the *RETURN* key. If the cursor was not on a line that contained *Jane* when you executed the command, no substitution would occur. You would get the message: `Substitute pattern match failed`.

Another command you could use if you wanted to find the first occurrence of *Jane* following the cursor position and replace it with *Sally* is:

:/Jane/s//Sally/

Only the first occurrence of *Jane* on that line would be changed to *Sally*. Notice this command is written slightly different from the previous command. You can think of it as search for *Jane* and substitute *Jane* for *Sally*.

If you wanted the substitution to take place at the first occurrence of *Jane* following the current line and also want all occurrences of *Jane* on that line to be replaced, you could use:

:/Jane/s//Sally/g

Note: The above form of the command will not change occurrences contained in the line where the cursor is located.

If you wanted to change only the **first** occurrence of *Jane* in **each** line in the document, you could use a *g* at the beginning of the search command:

:g/Jane/s//Sally/

Another command that you could use that accomplishes the same task is:

:1,$s/Jane/Sally/

Again, this changes only the first occurrence in each line. To change all occurrences, you must end the command with a *g*. Thus, the command to use is:

:g/Jane/s//Sally/g or **:1,$s/Jane/Sally/g**

Note: You can use *1,$s* in place of the first *g* or *%s*, i.e., *%s/Jane/Sally/*, when doing global searches.

Obviously, when you use the global substitute/replace command, you want to be sure you want every occurrence changed, or you may get some substitutions you are not expecting. For example, if you want to replace "mark" with "indicator" and there are words like "remark" in your text, if you enter **indicator**, words like 'remark' will also be changed, i.e., remark becomes "reindicator". (You can include a space preceding "mark" so matches of this type do not occur; however, if "mark" appears at the beginning of a line, it would not be changed. To insure that all occurrences would be changed, you would use the command *:%s/^<mark/indicator/g.*)

If you want a global replacement to be interactive; that is, ask you about each change, you can use the *c* option. Thus, to change all occurrences of *Jane* to *Sally* interactively, you would use the command:

:g/Jane/s//Sally/gc or **:1,$s/Jane/Sally/gc**

Then, for each occurrence, *vi* displays the line containing *Jane* (*Jane* will be underscored with ^'s) and waits for your input. To allow the change, type **y** and press the **RETURN** key; to give a negative response, type **n** and press the **RETURN** key. *vi* then searches for the next occurrence of *Jane* and again the sequence is repeated. When *vi* has found all occurrences, a message is displayed requesting that you press the RETURN key: `Hit Return to Continue`, and the cursor is positioned on the line containing the last occurrence of *Jane*.

Once you begin an interactive global substitution, you can't change your mind. However, you can complete the substitutions and then use the *undo* command (**u**); this negates the substitutions.

You can also specify a specific line number or range of lines when making substitutions. For instance, to change all occurrences of *Jane* to *Sally* in lines 5-10, you can use the command:

5,10s/Jane/Sally/g

Additionally, you can use an ampersand (&) in searches. The ampersand allows you to search for a string, insert a string before the "searched string" and keep the "searched string." For example, assume you have the phrase *mph*, and you want it to read *miles per hour (mph)*. To accomplish this search and replace, you could type:

:1,$s/mph/miles per hour (&)/

In this instance, the ampersand means keep the searched string, *mph*, and place it immediately following *mile per hour*. The *mph* is enclosed in parenthesis because the ampersand was placed inside parenthesis.

Example 16:

The following exercises provide practice using the various search/replace commands. To continue the exercises, you must be in the *practice* file.

For this exercise, change the word *system* to *convex C 1 computer.* To save yourself some keystrokes, use the command that searches from your current cursor position:

:/system/s//convex C 1 computer/

and press the **RETURN** key.

(Since there is only one *system* in the document, the current cursor position is immaterial.) However, for demonstration purposes, assume the current cursor position is on *industry* (you can move the cursor to this position). Your entry looks similar to this:

```
...
...
power at price and performance levels new to the Industry for
.....
This high-performance, low-cost system is now being
...
~
~
:/system/s//convex C 1 computer/
```

Now your screen displays the substitution:

```
This high-performance, low-cost convex C 1 computer is now being
...
~
:/system/s//convex C 1 computer/
```

For this exercise, change all the */[Cc]onvex's* to read *CONVEX.* You will need to search for both a lowercase and an uppercase *C.* Leaving the cursor at its current position, make this substitution by typing **:1,$s/[Cc]onvex/CONVEX/g** (you could also use *%s/[Cc]onvex/CONVEX/g* or *g/[Cc]onvex/s//CONVEX/g*) and pressing the **RETURN** key:

```
The Convex supercomputer offers the design engineer analysis
...
the Convex C 1 supercomputer offers the scientific user access to
...
This high-performance, low-cost convex C 1 computer is now being
~
~
:1,$s/[Cc]onvex/CONVEX/g
```

Now your screen displays the substitution:

```
The CONVEX supercomputer offers the design engineer analysis
...
the CONVEX C 1 supercomputer offers the scientific user access to
...
This high-performance, low-cost CONVEX C 1 computer is now being
~
~
:1,$s/[Cc]onvex/CONVEX/g
```

There are other ways you could make this same substitution; however, this demonstrates searching with a regular expression and making a substitution.

Use the interactive global substitution command to replace *computer* with *system*. Do not change either *supercomputer* or *super computer*. If the cursor is not at the beginning of the second paragraph, move it to that position.

To make the substitution, type **:1,$s/computer/system/gc** and press the **RETURN** key:

```
This high-performance, low-cost CONVEX C 1 computer is now being
...
~
:1,$s/computer/system/gc
```

The bottom of the screen shows the first line to be changed; type **n** and press the **RETURN** key, as this occurrence of *computer* should not be changed:

```
:1,$s/computer/system/gc
The CONVEX super computer offers the design engineer analysis
                 ^^^^^^^^n
```

The bottom of the screen shows the second line to be changed; type **n** and press the **RETURN** key, as this occurrence of *computer* should not be changed:

```
:1,$s/computer/system/gc
The CONVEX super computer offers the design engineer analysis
                 ^^^^^^^^
the CONVEX C 1 supercomputer offers the scientific user access to
                   ^^^^^^^^n
```

The next occurrence to change appears at the bottom of the screen; again, type **n** and press the **RETURN** key, as this occurrence should not be changed:

```
:1,$s/computer/system/gc
The CONVEX super computer offers the design engineer analysis
             ^^^^^^^^
the CONVEX C 1 supercomputer offers the scientific user access to
               ^^^^^^^^
super computer performance for less than $1,000,000.
      ^^^^^^^^n
```

The last occurrenee to change appears at the bottom of the screen. Since this occurrence should be changed, type **y** and press the **RETURN** key:

```
:1,$s/computer/system/gc
The CONVEX super computer offers the design engineer analysis
             ^^^^^^^^
the CONVEX C 1 supercomputer offers the scientific user access to
               ^^^^^^^^
super computer performance for less than $1,000,000.
      ^^^^^^^^
This high-performance, low-cost CONVEX C 1 computer is now being
                                               ^^^^^^^^y
```

Since this is the last occurrence of *computer*, the bottom of the screen now displays the message: `Hit return to continue:`

```
:1,$s/computer/system/gc
the CONVEX C 1 supercomputer offers the scientific user access to
               ^^^^^^^^
super computer performance for less than $1,000,000.
      ^^^^^^^^
This high-performance, low-cost CONVEX C 1 computer is now being
                                               ^^^^^^^^

Hit return to continue
```

Press the **RETURN** key and the replacements are made:

```
the CONVEX C 1 supercomputer offers the scientific user access to
super computer performance for less than $1,000,000.

This high-performance, low-cost CONVEX C 1 computer is now being
...
```

Move the cursor to the last line of the first paragraph (back two lines) by typing **2k**:

```
S uper computer performance for less than $1,000,000.
...
```

Now substitute *supercomputer* for *super computer.* Although this change could be made by deleting the space, use the substitution command to make this replacement.

Be sure you are on the correct line. To make the substitution, type **:s/ //** (substitute "no space" for the first space you come to) and press the **RETURN** key:

```
S upercomputer performance for less than $1,000,000.
...
:s/ //
```

If you wish to spend more time experimenting with the substitution commands, save this file by typing **ZZ**; then edit the *junk* file that you created previously. If you did not create this file previously, do so by typing **cp practice junk** at the shell prompt. When you feel comfortable using the previous commands, proceed to the next section, "Search/Replace Exercises."

Search/Replace Exercises

Before beginning the following exercises, save your *practice* file (if you have not done so) by typing **ZZ**.

For these exercises, you need the file *edit1.rev,* which you created for the edit exercises. If you do not have the *edit1.rev* file, either copy the file as instructed by your instructor or create the file with the following text:

Dear Firstname:

By now you should be able to create a file and edit the file using the delete, change, insert, and append commands.

Now to show everyone your new skills, edit this file according to the instructions in your tutorial.

Sincerely,

Your Instructor

Although the *edit1.rev* file is short, use the search and substitution commands to make the following modifications:

1. Use the appropriate search/substitution command to find *Dear* and change it to **Hi.**
2. Find all occurrences of *file* and change them to **document.**
3. Find *show* and change it to **demonstrate to.**
4. Begin the words delete, change, insert, and append with initial caps—capitalize first letter only of each word.
5. Change *Your Instructor* to your instructor's name.
6. Save this modified text to a file named *edit1.rev2.*
7. Abort the current buffer.
8. Print the files, *edit1.rev* and *edit1.rev2* and give the printed copies to your instructor.

After you have completed the exercises, proceed to the next section.

Cutting and Pasting

vi provides commands that aid you in moving text from one position in the document to another. One method involves the use of the delete commands and the put commands. When you invoke the delete command, the text is held in a temporary buffer; this text remains in the buffer until you enter another delete command. This text can be recovered using the *p* or *P* commands if you do not enter another delete command before invoking the put command.

When using the put (*p*, *P*) commands, you should be aware that they have slightly different meanings (just like *o* and *O*):

p Places whatever was last deleted or yanked following the cursor position

P Places whatever was last deleted or yanked preceding the cursor position

Example 17:

The following exercises provide practice using the delete and put command(s) to move text. To continue the exercises, you must be in the *practice* file.

Use the the appropriate command(s) to move the first sentence to the blank line following the first paragraph. One command that you can use is **1G.**

```
The CONVEX C 1 offers the design engineer analysis
power price and performance levels new to the industry for
increased and improved productivity.  By
...
...
```

Next delete the first sentence by typing **d)**:

```
y
using a combination of widely accepted software and hardware
...
...
```

The next step is to move the cursor to where you want the text to be placed—the blank line following the first paragraph—by typing **5** and pressing the **RETURN** key:

```
super computer performance for less than $1,000,000.
■
This high-performance, low-cost CONVEX C 1 computer is now being
...
```

Now type **p**:

```
super computer performance for less than $1,000,000.
T he CONVEX C 1 offers the design engineer analysis
power price and performance levels new to the industry for
increased and improved productivity.
This high-performance, low-cost CONVEX C 1 computer is now being ...
```

Notice that the blank line has disappeared; the last line of the text being moved is not followed by a newline. Thus, the loss of the blank line.

This time move *thermal,* (the comma too) before *electrical.* First move the cursor to *thermal,* (one way to move is to type **4** and press the **RETURN** key), then type **5W**:

```
used by engineers for structural, t hermal, fluid, electrical,
...
```

Next delete the word *thermal,* (delete the word and comma) and move to *electrical* by typing **dWW**:

```
used by engineers for structural, fluid, e lectrical,
...
```

Now type **P** which places *thermal,* and a space before *electrical*:

```
used by engineers for structural, fluid, thermal, electrical,
...
```

If you wish to experiment further with the delete/put commands, save the current contents by typing **ZZ**. Then edit another file (*junk*). Your *practice* file will be used in other exercises.

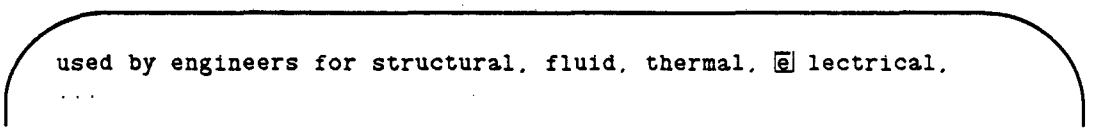
Marking Text

You can mark your current cursor position in the text and return to it using the mark command. Place the cursor over the letter that will serve as your "mark"; then type **m***x* where *x* represents any lowercase alpha character. To return to that position, type ` (grave accent—backward single quote) followed by the character you specified (*x*); to return to the beginning of the line that contains the mark, type ' (apostrophe—single forward quote) followed by the character that you specified (*x*). You can have more than one "mark" in the text; these "marks" are forgotten when you exit *vi*.

Example 18:

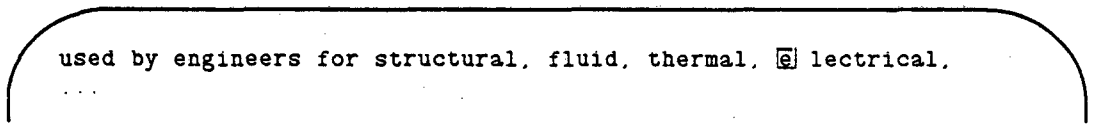
As in previous exercises, you will need to be in the *practice* file.

Mark the *e* on *electrical*; move the cursor to *e* (press SPACE bar) and type **mm**:

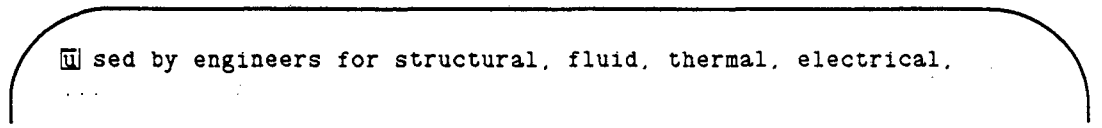


Now move to the beginning of the file by typing **1G**:

Now type **`m** to return to the "marked" position:



Move the cursor to the beginning of the line containing the "mark" by typing **'m**:



Marking and Moving Text

You can delete the copy from your "marked" character (*x*) to the current cursor position by typing **d`***x*. You can then move the cursor to where you want to add (paste) the text and type **p** or **P**. For additional information on blocking text, see the section entitled "Copying, Moving, and Writing Lines of Text."

Example 19:

Move the sentence beginning with *This high-performance* to the beginning of the file. Although this move can be completed other ways, for this exercise, mark the text that is to be moved, delete it, and move it to the beginning of the file.

First mark the *T* in the sentence beginning with *This high.* To do this, move the cursor to the *T* by typing); then mark the text by typing **mm**:

```
This high-performance, low-cost system is now being
used by engineers for structural, fluid, thermal, electrical,
...
```

Next move your cursor to the end of the paragraph by typing) (if the cursor isn't in the space following the period, add a space by typing *a* and pressing the SPACE bar, then ESC):

```
used by engineers for structural, fluid, thermal, electrical,
and static electromagnetic analysis.
...
```

The next step is to delete the text by typing **d'm**:

```
increased and improved productivity.
~
~
~
```

Now move to the beginning of the file by typing **1G**:

```
By
using a combination of widely accepted software and hardware
...
```

Place the text before the *B* by typing **P**:

```
This high-performance, low-cost CONVEX C 1 computer is now being
used by engineers for structural, thermal, fluid, electrical,
and static electromagnetic analysis.
By
```

Copying Text to Another Location

The yank command places a copy of the text (word(s), lines(s), sentence(s), paragraph(s), etc.) in a temporary buffer. This text remains in the buffer until you enter another command that places text into the buffer. You can use the yank (*y*) command with the

put commands to copy a block of text and move that text to another location in the current document; the original text is not deleted. (For additional information on copying blocks of text, see the section entitled "Copying, Moving, and Writing Lines of Text.")

The yank command can be used with the movement operators; the following table explains how to use this command with some of the movement operators:

Table 4-10: Combining Yank and Movement Commands

Command	Explanation
yy	Copies one line of text into the temporary buffer
Y	Copies one line of text into the temporary buffer; same as *yy*
***n*yy or *n*Y**	Copies *n* number of lines lines of text into the temporary buffer
yw	Copies one word of text into the temporary buffer
y)	Copies text beginning at cursor position to end of current sentence into the temporary buffer

Example 20:

For this exercise, make a copy of the first three lines of the file and move them to the end of the file. Assuming the cursor is at the beginning of the file, type **3yy** or **3Y**:

```
This high-performance, low-cost CONVEX C 1 computer is now being
used by engineers for structural, thermal, fluid, electrical,
and static electromagnetic analysis.
By
```

Notice the text is not deleted nor does the cursor move when you invoke the command. Now move the cursor to the end of the file by typing **G**:

```
...
...
Increased and improved productivity.
```

Type **p** to place the text at the end of the file:

```
increased and improved productivity.
This high-performance, low-cost CONVEX C 1 computer is now being
used by engineers for structural, thermal, fluid, electrical,
and static electromagnetic analysis.
```

Reading a File into the Buffer

You will find the read-file (*:r filename*) command particularly useful when you wish to use text saved in another file in your current text. This command reads (inserts) an entire file into your current document on the line immediately following your cursor location.

Example 21:

Insert a copy of your *practice* file into your current document. Position the cursor at the end of your current document by typing **G**:

```
and static electromagnetic analysis.
~
~
```

To insert the file, type **:r practice** and press the **RETURN** key:

```
and static electromagnetic analysis.
~
~
:r practice
```

A copy of the file is placed following the current line:

```
and static electromagnetic analysis.
This high-performance, low-cost CONVEX C 1 computer is now being
used by engineers for structural, thermal, fluid, electrical,
and static electromagnetic analysis.
By
using a combination of widely accepted software and hardware
standards, implemented with advanced technology,
the CONVEX C 1 supercomputer offers the scientific user access to
super computer performance for less than $1,000,000.
The CONVEX C 1 offers the >design engineer analysis
power price and performance levels new to the industry for
increased and improved productivity.
```

Your *practice* file may look different from that shown in this example.

Delete the text that you inserted by typing **d}**. Proceed to the following section for additional practice using the cut and paste commands.

Cut and Paste Exercises

Before beginning the following exercises, save your *practice* file (if you have not done so) by typing **ZZ**.

For these exercises, you need the file *edit1.rev2*, which you created for the edit exercises.

If you do not have the *edit1.rev2* file, either copy the file as instructed by your instructor or create the file with the following text:

Hi Firstname:

By now you should be able to create a document and edit the document using the Delete, Change, Insert, and Append commands.

Now to demonstrate to everyone your new skills, edit this document according to the instructions in your tutorial.

Sincerely,

Instructor_name

If you created the file *edit1.rev2*, be sure to save it before making any of the revisions. Make the following modifications to your *edit1.rev2* file:

1. At the end of the current text, insert your *practice* file.
2. Move the paragraph beginning with *By using a* directly preceding the second paragraph which begins with *Now to demonstrate.*
3. If you do not have a blank line preceding and following the paragraph you just moved, insert them.
4. Copy the first paragraph of your current document and put it at the end of the current file.
5. Save these modifications to a file named *edit1.rev3.*
6. Abort the modifications to the current file.
7. Print the files *edit1.rev2* and *edit1.rev3* and give the printed copies to your instructor.

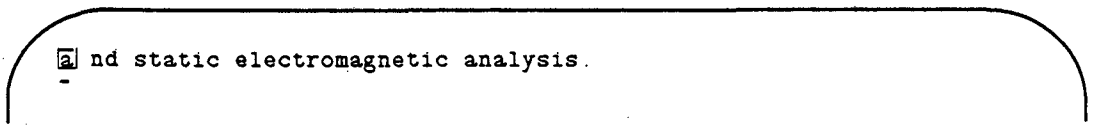
After completing these exercises, proceed to the next section.

Repeating Commands

Typing a period repeats the last command that modified the buffer (delete, insert, change, or append). For instance, if you have typed **2dw**, then typing a period will invoke the command again. If you have typed the command **cw** followed by the new word and type the repeat command (.), the new word is repeated.

Example 22:

To experiment with the repeat command, move your cursor to the end of the file by typing **G**:



Append the following text to the file by typing **A** followed by a **RETURN** or type **o**. Now enter the text:

This is the sample for the repeat command.

Your screen looks similar to:

```
and static electromagnetic analysis.
This is the sample for the repeat command.  ■
~
```

Return to command mode by press the **ESC** key.

Be sure you are in command mode (press **ESC**) before entering the repeat command. Now repeat the command by typing a period (.):

```
and static electromagnetic analysis.
This is the sample for the repeat command.
This is the sample for the repeat command.  ■
~
```

As you can see the sentence was repeated when you entered the repeat command.

Joining Lines

The *J* command allows you to join two lines. For instance, if you have the lines:

This is line one.
This is line two.
This is line three.

and you want all three lines to be on the same line, you would place the cursor on the first line and type **J** two times. Thus, the lines would then appear on one line:

This is line one. This is line two. This is line three.

Example 23:

For this exercise, you need the *practice* file. Move the cursor to the line that begins with *By*. Use the appropriate search command to move to this position. If you are at the end of the file, type *?By* followed by a **RETURN**:

```
and static electromagnetic analysis.
B y
using a combination of widely accepted software and hardware
...
?By
```

Now join the line beginning with *By* to the line beginning with *using* by typing J:

```
and static electromagnetic analysis.
By using a combination of widely accepted software and hardware
...
```

Fundamental *vi* Technique Exercises

Write the command(s) that you would enter to complete each task:

1. Replaces character under cursor with *b*.
2. Deletes three words.
3. Deletes two lines.
4. Joins two lines.
5. Moves cursor to line 10.
6. Marks the current cursor position.
7. Returns to the "marked" position.
8. Substitutes *T* for first occurrence of *t* at the beginning of a line only.
9. Substitutes *Fred* for every occurrence of *James*.
10. Changes character under cursor to a capital.
11. Finds the word *sample*.
12. Writes line 1-10 to a new file.
13. Aborts the modifications to the current buffer.
14. Writes the current buffer to a new file named *sample*.
15. Allows you to insert text preceding the current cursor position.
16. Allows you to append text to the end of the file.
17. Corrects the misspelling of *diferent*.

The following exercises allow you to practice *vi* editing techniques. For these exercises, you need a file named *edit1.rev2*. Either copy this file as directed by your instructor or create a file containing the following:

Hi Firstname:

By now you should be able to create a document and edit the document using the Delete, Change, Insert, and Append commands.

Now to demonstrate to everyone your new skills, edit this document according to the instructions in your tutorial.

Sincerely,

Instructor_name

If you created the file *edit1.rev2* and haven't saved it, do so before beginning the following exercises. Make the following modifications:

1. Add the following text as the second paragraph; be sure to precede the and follow the text with a blank line:

   You have learned a variety of ways to revise the document. It takes practice to use these commands effectively.

2. Capitalize your entire first name which appears as the first line of text.
3. Add this sentence as the last line of the first paragraph:

   In addition, you can move text easily.

4. Insert the current date at the beginning of the file and follow it with four blank lines.

5. Change the last paragraph to read:

   You are to be congratulated for your diligence. Now you can create masterpieces with ease.

6. Change *the* following *revise* to **a**.
7. Save the modifications to a file named *ap.ex*.
8. Abort the modifcations to the current file.
9. Print a copy of the files *edit1.rev2* and *ap.ex* and give the printed copies to your instructor.

You have now completed the tutorial on the fundamental *vi* commands. Additional commands for the more experienced user are described in the following pages. If you plan to use the tutorial portion in the following sections, you will need to retain your *practice* file.

vi Commands for the Experienced User

This section contains the more advanced features of *vi*. It illustrates variations of the basic commands, developing *vi* macros, as well as customizing your *vi* environment by editing or creating the *.exrc* file. In the following sections, it is assumed that you are an experienced *vi* user or that you have completed the preceding exercises contained in this tutorial. Since you are an experienced user, there will be no instructions on how to move the cursor. However, the examples do provide step-by-step instruction on using the commands.

Copying, Moving, and Writing Lines of Text

In addition to the mark commands discussed previously, *vi* provides commands that allow you to mark lines of text and then delete, copy, move, or write that text to a new file. For example, assume you have marked text that you want to write to another file; the text is marked with an *a* at the beginning of the text and a *b* at the end of the text. Now to write the text to a new file, type **:'a,'bw** *newfile* followed by a **RETURN**.

Another way you can accomplish this same task is to mark the beginning of the text you want to copy (delete) with an alpha character (**ma**) and move to the end of the text; then enter the command to write the text to a new file, **'a,w** filename. The text, beginning with the mark to the cursor position, is written to the file. (You can have more than one mark set in the text in the current buffer.)

The following table describes some of the methods for manipulating text:

Table 4-11: Text Manipulation Commands

Command	Explanation
:'*a*,'*b*t.	Copies marked lines *a*–*b* after the current line; original marked text remains unchanged
:'*a*,'*b*m.	Moves marked lines *a*–*b* after the current line; original marked text remains unchanged
:'*a*,'*b*d	Deletes marked lines *a*–*b*
:'*a*,'*b*w *filename*	Writes marked lines *a*–*b* to a new file; cannot be used to write to an existing file, as *vi* refuses to write the file
:'*a*,w *filename*	Writes marked lines through cursor position to a new file; cannot be used to write to an existing file (in *ex* writes entire line where cursor is located, not just to cursor position)
:'*a*,'*b*w! *filename*	Writes marked lines *a*–*b* to an existing file; overwrites contents
:'*a*,'*b*w>> *filename*	Appends marked lines *a*–*b* to an existing file; does not overwrite contents

Example 24:

For the following exercises, you will need the *practice* file created in previous exercises. If you do not have this file, create it with the following contents:

```
This high-performance, low-cost CONVEX C 1 computer is now being
used by engineers for structural, thermal, fluid, electrical,
and static electromagnetic analysis.
By using a combination of widely accepted software and hardware
standards, implemented with advanced technology,
the CONVEX C 1 supercomputer offers the scientific user access to
super computer performance for less than $1,000,000.
The CONVEX C 1 offers the design engineer analysis
power price and performance levels new to the industry for
increased and improved productivity.
This high-performance, low-cost CONVEX C 1 computer is now being
used by engineers for structural, thermal, fluid, electrical,
and static electromagnetic analysis.
This is the sample for the repeat command.
This is the sample for the repeat command.
```

While in the *practice* file, use the mark commands to mark the second occurrence of the text beginning with *This high-performance* and ending with *electromagnetic analysis.* Use *a* for the beginning marker and your cursor position for the ending marker.

To complete this task, move your cursor to the *T* on *This* and set the first marker. To set the marker, type **ma**:

```
...
increased and improved productivity.
This high-performance, low-cost CONVEX C 1 computer is now being
used by engineers for structural, thermal, fluid, electrical,
and static electromagnetic analysis.
This is the sample for the repeat command.
...
```

Move the cursor to the end of the block (if you didn't leave a space following the period, place the cursor on the period):

```
...
increased and improved productivity.
This high-performance, low-cost CONVEX C 1 computer is now being
used by engineers for structural, thermal, fluid, electrical,
and static electromagnetic analysis.
This is the sample for the repeat command.
...
```

Finally, write this text to a file named **example1** by typing **:'a,w example1** and pressing the **RETURN** key:

```
...
increased and improved productivity.
This high-performance, low-cost CONVEX C 1 computer is now being
used by engineers for structural, thermal, fluid, electrical,
and static electromagnetic analysis.
This is the sample for the repeat command.
...
~
~
~
:'a,w example1
```

After you execute the command, a message appears at the bottom of the screen: **"example1" [New file] 3 lines, 163 characters**

```
...
and static electromagnetic analysis.
This is the sample for the repeat command.
...
~
~
~
"example1" [New file] 3 lines, 169 characters
```

Now experiment with another command. This time append some text to the *example1* file (later you will edit this file.) Set marks *c* and *d* at the beginning and ending, respectively, of the last sentence in the file.

Of course, to do this, move to the beginning of the appropriate line and type **mc**:

```
This is the sample for the repeat command.
This is the sample for the repeat command.
~
```

Set the end mark by moving to the end of the line and typing **md**. It is not necessary to mark the position; you can leave your cursor at this position and enter the command to write the text to another file. (If you don't have a space after the period, place your cursor on the period.)

```
This is the sample for the repeat command.
This is the sample for the repeat command
~
```

Finally, append this text to *example1* by typing **:'c,'dw>> example1** and pressing the **RETURN** key; if you didn't mark the end of the text with *md*, type **:'c,w>> example 1**:

```
This is the sample for the repeat command.
This is the sample for the repeat command
~
:'c,'dw>> example1
```

As in the previous exercise, a message appears at the bottom of the screen: `"example1" 1 line, 43 characters`

```
This is the sample for the repeat command.
This is the sample for the repeat command
~
"example1" 1 line, 43 characters
```

(Again, you could have completed the previous exercise by marking the beginning of the text (**mc**), moving to the end of the text and entering the command **:'c,w >> example1**.)

Notice the message that is displayed does not say that the line was appended—it was. In a later exercise, you will view the file and see that the line actually was appended to the *example1* file.

If you wish to experiment further with the commands, save the current file as the *practice* file will be used in other exercises.

Using Named Buffers

In previous exercises, you used *vi*'s un-named buffer. However, the un-named buffer will hold only the last text that you deleted, changed, or yanked. Thus, you cannot do several commands in succession that required use of the buffer. The use of named buffers provides versatility and enhances productivity when creating and revising text. Using named buffers allows you to take text from other buffers and/or files and combine the text from those files, with the option of saving or deleting buffer text. Using the mark/block commands allowed you to do essentially the same thing; however, you could only create a new file or append to another file. With the named buffers, you will be able to place text exactly where you want it, not just at the end of existing text.

vi has 26 named buffers; they are accessed with lowercase single-character names of *a–z*. To use a named buffer, precede the command with quotes and the buffer name. For example, if you want to delete three lines into the buffer named *a*, type **"a3dd**.

Text can also be appended to the named buffers. When appending an existing buffer, use the uppercase buffer name instead of the lowercase name. For example, to append text to the existing buffer *a*, type **"A2dd**; the deleted text is appended to buffer *a*. If you forget to use the lowercase letter (a), all previous contents of the buffer are erased.

Example 25:

The following exercises shows you how to take text from one file and place it in another existing file. These exercises also show you how to go to another file without actually leaving your current file. As in previous exercises, you need to be in the *practice* file.

Copy the sentence beginning with *By using* to the buffer named *a*. Move your cursor to the the beginning of the appropriate sentence and type **"ay)**:

```
and static electromagnetic analysis.
By using a combination of widely accepted software and hardware
standards, implemented with advanced technology.
the CONVEX C 1 supercomputer offers the scientific user access to
super computer performance for less than $1,000,000.
...
...
```

Now place this text in the file *example1*, which you created in a previous exercise. To go to another file you can type **:e** *filename* followed by a **RETURN**.

To move to the *example1* file, type **:e example1** and press the **RETURN** key. The *example1* file is displayed on your screen:

```
This high-performance, low-cost CONVEX C 1 computer is now being
used by engineers for structural, thermal, fluid, electrical,
and static electromagnetic analysis.
This is the sample for the repeat command.
~
"example1" 4 lines, 212 characters
```

Place the text that you copied into buffer *a* preceding the last line of text. First move the cursor to the last line of text (if it isn't there); then type **"aP**. The text is inserted:

```
This high-performance, low-cost CONVEX C 1 computer is now being
used by engineers for structural, thermal, fluid, electrical,
and static electromagnetic analysis.
By using a combination of widely accepted software and hardware
standards, implemented with advanced technology.
the CONVEX C 1 supercomputer offers the scientific user access to
super computer performance for less than $1,000,000.
This is the sample for the repeat command.
~
4 lines
```

Now you can return to the *practice* file to make additional revisions. Since the contents of *example1* have been modified, you need to *write* the file; type **:w** and press the *RETURN* key. After you enter this command, your screen looks similar to:

```
This high-performance, low-cost CONVEX C 1 computer is now being
used by engineers for structural, thermal, fluid, electrical,
and static electromagnetic analysis.
B y using a combination of widely accepted software and hardware
standards, implemented with advanced technology,
the CONVEX C 1 supercomputer offers the scientific user access to
super computer performance for less than $1,000,000.
This is the sample for the repeat command.
~
"example1" 8 lines, 438 characters
```

Notice the message that is displayed at the bottom of the screen.

The command **:e#** lets you alternate between two files without entering a filename. Typing this command will return you to the *practice* file. Then if you want to go back to *:example1*, you can enter **:e#**; *vi* keeps track of the files for you.

Now move back to the *practice* file by typing **:e#** and pressing the **RETURN** key:

```
T his high-performance, low-cost CONVEX C 1 computer is now being
used by engineers for structural, thermal, fluid, electrical,
and static electromagnetic analysis.
...
...
"practice" 15 lines, 806 characters
```

If you had attempted to go directly to the *practice* file by typing **:w#** without saving the current file, you would have seen the message: `No write since last change (edit! overrides)`. Thus, if modifications have been made to the current buffer, you must either save (**:w**) the current buffer or enter **e!#** (does not save the modifications) to move to another file.

Accordingly, if you had deleted the text from *practice* and put it into a buffer (instead of copying it) and then tried to move to *example1*, a message would have displayed telling you no *write* had been made. To *write* the file, you would have needed to enter the command **:w**; then **:e example1.**

Now delete the last line of *practice* and append it to buffer *a*. First move to the appropriate line and type **"Add.** (If you type a lowercase *a* instead of a capital, the contents of *a* are overwritten.) Your screen looks similar to:

```
and static electromagnetic analysis.
This is the sample for the repeat command.
T his is the sample for the repeat command.
~
~
```

The text has been appended to buffer *a* (if you typed an uppercase *A*) although you cannot see the contents on your screen. Verify that you have "appended" the contents by placing the contents of buffer *a* at the end of the current file. Be sure your cursor is on the last line of the file; then type **"ap**. If you actually appended the text (instead of overwriting) to buffer *a*, your screen looks similar to:

```
...
and static electromagnetic analysis.
This is the sample for the repeat command.
By using a combination of widely accepted software and hardware
standards, implemented with advanced technology,
the CONVEX C 1 supercomputer offers the scientific user access to
super computer performance for less than $1,000,000.
This is the sample for the repeat command.
-
```

Creating Abbreviations to Use in *vi*

vi has a function that allows you to create abbreviations to be used during the current edit session or for all edit sessions. Once you have created the abbreviation, whenever you are in text input mode and you type the abbreviation with a space following it, the abbreviation is expanded. For example, if you want the string *cx* to expand to *CONVEX COMPUTER CORPORATION*, you can either place the appropriate command in your *.exrc* file to be available for all *vi* edit sessions, or type the appropriate command at the beginning of each edit session. In either case, for our example, the command is: **:abbr cx CONVEX COMPUTER CORPORATION**. However, when you put the abbreviation in the *.exrc* file do not include the colon (:).

Example 26:

You need to be in your *practice* file before beginning the following exercise.

Create the abbreviation *cx* for CONVEX COMPUTER CORPORATION; then verify that the abbreviation is expanded correctly.

The cursor can be located anywhere in the *practice* file. For this exercise, it is assumed your cursor is at the beginning of the file. Create the abbreviation by typing **:abbr cx CONVEX COMPUTER CORPORATION** and pressing the **RETURN** key:

```
This high-performance, low-cost CONVEX C 1 computer is now being
used by engineers for structural, thermal, fluid, electrical,
and static electromagnetic analysis.
...
...
:abbr cx CONVEX COMPUTER CORPORATION
```

Verify that the abbreviation is expanded correctly. Since the abbreviation is expanded only in text input mode, insert the abbreviation at the beginning of the file. Type **i** followed by the characters **cx** and then strike the **SPACE** bar. After entering the command, your screen looks similar to:

```
CONVEX COMPUTER CORPORATION T his high-performance, low-cost . . . . .
...
...
```

Press the **ESC** key to return to command mode. Then delete *CONVEX COMPUTER CORPORATION.*

Now add an abbreviation to your *.exrc* file. As an example, make an abbreviation for your name; create the abbreviation so that the characters *mn* expand to your first and last name. Don't exit *vi* but do write the file by typing **:w**.

Now move to the *.exrc* file by typing **:e .exrc**. Your screen looks similar to:

```
set ai nowarn
~
~
~
".exrc" 1 line, 27 characters
```

Now add this line to the file (it's assumed you know how to do this; by now you are an experienced *vi* user!):

abbr mn *First_name Last_name*

Your entry looks similar to:

```
et ai shell=/bin/csh nowarn
abbr mn First_name Last_name
~
~
".exrc" 1 line, 27 characters
```

Now write the file. Since *.exrc* is read only when *vi* is invoked, you must save the files and exit to the *shell*. Save the file by typing *ZZ*:

```
set ai nowarn
abbr mn First_name Last_name
~
~
".exrc" 2 lines, 40 characters
%■
```

Load the *practice* file by typing **vi practice**; your *practice* file is displayed:

```
This high-performance, low-cost CONVEX C 1 computer is now being
used by engineers for structural, thermal, fluid, electrical,
and static electromagnetic analysis.
...
...
```

Now test your abbreviation. Type **i**; then type **mn** and strike the **SPACE** bar. You should see the abbreviation expanded to your name. If it does not expand, you probably made an error in your *.exrc* file. Try re-editing the file; if the abbreviation still does not expand correctly, ask your instructor for help. When you have finished experimenting with this macro, delete all occurrences of *your name*.

Creating Macros in *vi*

vi allows you to create macros to be used during the current edit session or that are available for all *vi* edit sessions. The macros are similar to abbreviations, but macros execute a collection of *vi* commands. You can place the macros in your *.exrc* file or set them at each edit session. Limit your macro name to two characters.

For example, assume you want *q* to write the current buffer and return you to the *shell* prompt. You would add this line to your *.exrc* file:

map q :wq^M

As you already know, the portion *:wq* saves the file and exits to the *shell* prompt. You are probably wondering why the characters *^M* are part of the command. Since the *:wq* command normally ends with a RETURN, the RETURN must be included in the command. To include the RETURN as part of the command line, you must type **^V** and press the **RETURN** key. When you complete this sequence, the characters *^M* will be displayed on the screen. Then you press the **RETURN** key again—you want to go to the next line.

You must precede any command that normally ends/aborts your entry in *vi* with *^V*. For instance, an *ESC* would normally end a *vi* command; if *ESC* is actually part of the command, then precede it with *^V*

Macros can be attached to a function key. Assume you want to attach the previous command to a function key. You type the number of the key preceded by a **#**. Thus, instead of *q*, in our example, you would have typed **#1** to attach it to function key 1, **#2** to attach it to function key 2, etc. (The function keys are those keys labeled with *PF*.)

Macros can be set up to use in command mode, as are the previous examples, or in text input mode. To create a macro that is used during text input mode, put an exclamation mark following the word *map*. Thus, *map! command_name commands* would work only during text input mode.

Example 27:

For this exercise, you will need to be in the *.exrc* file. If you are currently in the *practice* file, move to *.exrc* by typing **:e .exrc** and pressing the **RETURN** key. Your screen looks similar to:

```
set ai nowarn
abbr mn First_name Last_name
~
~
".exrc" 2 lines, 42 characters
```

Attach the insert command (i) to key *PF1*. Append the appropriate command to the file:

map #1 i

Your entry looks similar to:

```
set ai nowarn
abbr mn First Last
map #1 i
■
~
```

Write the file by typing **ZZ**.

```
set ai nowarn
abbr mn First Last
map #1 i
~
~
".exrc" 3 lines, 50 characters
%■
```

Now load your *practice* file.

```
This high-performance, low-cost CONVEX C 1 computer is now being
used by engineers for structural, thermal, fluid, electrical,
...
...
...
"practice" 19 lines, 1040 characters
```

Now verify that the function key allows you to insert text. Press the **PF1** key and type the text:

Yes, it works!

If the command works properly, your screen looks like this after you press *PF1* and enter the text:

```
Yes, it works! T his high-performance . . . .
used by engineers for structural, thermal, fluid, electrical,
...
...
...
"practice" 19 lines, 1040 characters
```

Remember to return to command mode, you must press the **ESC** key. Then delete *Yes, it works!*.

For this exercise, create a macro that allows you to type **^F** to move forward one word, the equivalent of *w* in *vi*. Create this macro for this edit session only.

The cursor can be at any position in the file; however, it is assumed that the cursor is at the beginning of the file. To create the macro type **:map ^F w** and press the **RETURN** key. When you create the macro, actually hold the CTRL key and press the F key; don't type a caret (^) and a capital F.

When you enter the macro information, you screen looks similar to this:

```
T his high-performance, low-cost CONVEX C 1 computer is now being
used by engineers for structural, thermal, fluid, electrical,
...
...
-
-
:map ^F w
```

Now test your macro by typing **^f**; the cursor should move right to the beginning of the next word. If it doesn't, try retyping the command. If it still doesn't work correctly, ask your instructor for help. When you create the macro, the *w* is still operative.

You can map a series of commands. The following examples show how to map a series of commands. If you wish to try this exercise, save your current buffer.

Create a file called *macroprac*. Enter the following text with a carriage return following each line:

workspace that
display area
descriptive line
message line

Define a macro that places *It is a* at the beginning of each of the previous lines when executed. Assign the macro to *^ex*. (The *^ex* means hold the *CTRL* key while striking only the *e* key; then release the *CTRL* key and strike the *x* key.) In this example, the words *It is a* will display and the cursor will move to the beginning of the following line each time the macro is executed.

To create the macro, type what you would actually enter if you were adding the words *It is a* at the beginning of the line. Thus, you invoke insert mode (i), type the string (It is a), press *ESC* to leave insert mode, move down one line (j), and move to the beginning of that line (0). Remember when using *ESC* in creating a macro you must type ^*V* and then press the *ESC* key.

Now to create the macro, type **:map ^ex iIt is a ^[j0** and press the **RETURN**. (You won't actually type [. After you type the character *a*, press the SPACE bar and then type **^V** and press the **ESC** key—the [is displayed on the screen.) Finish entering the macro (j0 and press RETURN).

When you create the macro, your screen looks like this:

```
workspace that
display area
descriptive line
message line
■
~
~
:map ^Ex iIt is a ^[j0
```

Verify that you macro works correctly. Move your cursor to the beginning of the first line. Type the macro **^ex**. If you macro is correct, your screen looks like this when you execute the macro:

```
It is a workspace that
display area
descriptive line
message line
~
~
~
```

As you can see, the string was inserted; type **^ex** again and the string is inserted at the beginning of the second line:

```
It is a workspace that
It is a display area
descriptive line
message line
~
~
~
```

Again the string was inserted and the cursor is at the beginning of the following line.

Customizing the *vi* Environment

Whenever you run *vi*, the editor looks in the *.exrc* file for initial commands and options. You can create an *.exrc* file in your home directory that contains *vi* options, or you can set the options while in *vi* by using:

:set *option*
:set *option*=*value*

You can list more than one option on a line, i.e., **set** *option option* ....

To turn off an option, use the form: **set no***option*

In the *.exrc* file, you can set options by entering: **set** *option*

You can list more than one option on a line. Some of the options have abbreviated forms that can be used in place of the expanded form.

When you are using *vi* and you want to see a listing of the options that are set (different than the default), you can type **:set.** Typing **:set all** displays a listing of all options available; however, some of the options that are in the list are *ex* options. To determine a single option's value, type **set** *option***?**.

The following table describes options you may wish to set; abbreviations are included after the option name. For a complete listing of options, see *CONVEX Tutorial Papers, Ex Reference Manual.*

Table 4-12: Setting *vi* Options

Option	Description of Options When Set to On
autoindent (ai)	In text input mode causes the cursor to be placed at the same indentation level as the previous line; typing **^d** moves the cursor back to the left margin
autowrite (aw)	Causes current buffer to be automatically written when you enter **:!**, **:n** or **:tag**
ignorecase (ic)	Ignores cases in searching; MOVE matches move or MOVE
lisp	Modifies *vi* for easy entry of *lisp* programs
list	In text, indicates tabs with ^I and end of line with $
magic	Regular expressions (., *, ^, []) retain special meanings in searches
number (nu)	Lines displayed are prefixed with line numbers
redraw	Simulates a smart terminal on a dumb one
shiftwidth	Determines tabstops used by ^T and ^D and shift distance for <,>
showmatch	Shows matching (or } when) or } is typed
terse	Displays terse error warning messages
warn	`No write since last change` is displayed before a :! sequence if there has been no write
wrapmargin	When greater than 0, *vi* automatically inserts a carriage return at the first natural break point (word boundary)

Example 28:

To complete the following exercises, you can use a file of your choice.

To determine which options are currently set type **:set:**

```
You have several lines of text
on your screen.  It can be any
file.
~
~
~
:set
```

A list of option is displayed at the bottom of your screen:

```
You have several lines of text
on your screen.  It can be any
file.
~
~
~
autoindent redraw term=vt100 nowarn
```

For this exercise determine the value of *wrapmargin* by typing **:set wrapmargin?** and pressing the **RETURN** key. Your entry:

```
You have several lines of text
on your screen.  It can be any
file.
~
~
~
:set wrapmargin?
```

The value is then displayed for wrapmargin:

```
You have several lines of text
on your screen.  It can be any
file.
~
~
~
wrapmargin=0
```

When *wrapmargin* has a setting of zero, text wraps at about 80 characters, the width of your screen. This exercise demonstrates how to change the value of *wrapmargin* to 30.

To reset the wrapmargin, type **:set wrapmargin=30** and press the **RETURN** key.

```
You have several lines of text
on your screen.  It can be any
file.
■
~
~
:set wrapmargin=30
```

Invoke text input mode and start typing text of your choice; after you have entered about 50 characters (must include spaces to work correctly), a carriage return is inserted automatically, and the cursor moves to the beginning of the next line.

```
You have several lines of text
on your screen.  It can be any
file.
This text shows you how word wrap works when it is
set at 30.  This is only an approximation; you
will have to test it by entering some of your own
text.  You don't press RETURN; it is done automatically
for you.
■
```

Notice that word wrap breaks at word boundaries—words are not divided, so the right-hand margin is jagged.

This time turn off the autoindent option. If you don't know how autoindent works, do the following exercise before turning off the option.

Move to the last line of your current file or the *practice* file:

```
super computer performance for less than $1,000,000.
This is the sample for the repeat command.
~
~
~
```

Append the following text. When you enter the text, press the TAB key on the first line; do not press the TAB key again. Be sure to end each line with a RETURN:

(*tab*)**This shows how autoindent works.**
Every line that you type following the
first indentation will line up under that
indentation.

Your entry looks similar to this:

```
super computer performance for less than $1,000,000.
This is the sample for the repeat command.
        This shows how autoindent works.
        Every line that you type following the
        first indentation will line up under that
        indentation.
■
~
```

Each line is automatically aligned at the indentation. To move to the left margin, you can type ^d or press the **ESC** key, which ends text input mode and returns you to the command mode.

Now turn off the *autoindent* option by typing **:set noai** and pressing the **RETURN** key:

```
super computer performance for less than $1,000,000.
This is the sample for the repeat command.
        This shows how autoindent works.
        Every line that you type following the
        first indentation will line up under that
        indentation.
■
:set noai
```

Append three additional lines. Begin the first line with a TAB—press the **TAB** key—and type the first line of text ending with a RETURN. Now type the additional lines, pressing **RETURN** after each line. Notice when you type the additional lines that the text begins at the left margin, not at the indentation.

Use the following lines as your text:

*(tab)***This line is indented by pressing TAB.**
This lines starts at the left margin automatically.
So does this line.

Your screen looks similar to:

```
super computer performance for less than $1,000,000.
This is the sample for the repeat command.
        This shows how autoindent works.
        Every line that you type following the
        first indentation will line up under that
        indentation.
        This line is indented by pressing TAB.
This lines starts at the left margin automatically.
So does this line.
■
```

Other Useful Features

You may find some of the features described in the following paragraph helpful when using *vi*.

tags Files

As a programmer you may use *tags* files. *vi* provides some options for working with these files. For instance, if you have a *tags* file (contains the object name, usually a function or subroutine the file in which it is defined, and an address), you can edit at the specified tag with the command:

vi -t *tag*

If the current directory contains a file named *tags* which contains lines like *label file search-string*, you can use **:tag** *label* from within *vi*. This commands updates the current file (if necessary), reads in *file* and positions the cursor at the *search_string*. See *ctags*(1) and *ftags*(1) in the *CONVEX UNIX Programmer's Manual*.

Determining the Current Line Number

To determine the line number of the current line, type **^g**. A message is displayed at the bottom of the screen giving the line number and total number of lines in the buffer.

Returning to Previous Cursor Position

You can return to a previous cursor location after invoking commands such as */*, *G*, or *?* by typing ` ` while in command mode.

Escaping to the Shell

If you want to execute a shell command from within *vi*, type *:!command* and press the **RETURN** key. The command is run and when the command is finished, *vi* displays the message `RETURN to continue.` Either enter another command by typing *:!command* or press the RETURN key to continue editing.

You can also execute commands by getting a new shell. To get a new C shell, type **:!csh** and press the **RETURN** key. Enter the commands in the usual manner; when you are finished entering commands, return to the editor by typing **^d** and pressing the **RETURN** key (as directed).

Instead of requesting a new shell, you can suspend *vi* with **^z** which returns you to your shell. You can execute shell commands, then resume editing by bringing the job to the foreground (typing **fg**). If you don't know how to resume a stopped job, see the *csh*(1) manual page.

Using the Spell Function

UNIX provides a spell-check function. The *spell* command can be entered from *vi*. If you want to spell-check your current file, first write the buffer (**:w**). Then to do the spelling check from within *vi*, enter **:$r! spell %** and press the **RETURN** key. A message is displayed, `spell` *buffername* and the output is appended to your current buffer. Of course, after you have made the corrections, you need to delete the list of misspelled words.

Experienced *vi* User Exercises

Write the command(s) that you would enter to complete each task:

1. Explain how to mark a block of text. Then write the commands that would mark the beginning and ending of a block of text.

2. Write a command that appends a block of text marked with *f* and *g* to the file *sam.block*.

3. Write a command that copies a block of text marked with *f* and *g* immediately following the current line.

4. Write a command that appends three deleted lines to buffer *c*.

5. Explain how to move text from *file* to *file2*. The text must be placed as the second paragraph of *file1*.

6. Write the command that allows you to alternate between files when using the *vi* editor.

7. Write a command that expands the abbreviation *sy* to *Sincerely yours*.

8. Explain how to map function key 3 (for all edit sessions) to the command that deletes a line.

9. How do you enter an escape in a macro?

10. Write a command that causes *vi* to be case insensitive during searches.

11. What command displays your current line number?

12. Write a macro (%Bl) that can be executed in text input mode that: precedes a word or string with *\fB*.

13. Write a macro (%B) that can be executed in text input mode that precedes a word or string with *\fB* and follows it with *\fI*.

14. Write a command that displays a listing of the current *vi* options that are set.

This concludes the *vi* Tutorial. A list of *vi* commands is included in the Appendix of the training materials.

5

Information Handling Commands

Objectives for Chapter 5

After completing this chapter, you will be able to:

1. Display the contents of files using the *cat* and *more* commands.
2. Display specified number of beginning or ending lines of a file.
3. Determine the number of words, lines, or characters contained in a specified file.
4. Determine the first disagreement between two specified files.
5. Determine changes needed to make a specified file just like the second specified file.
6. Determine lines in common between two specified files.
7. Sort files alphabetically, in reverse order, and in dictionary order.
8. Remove a file from the print queue.
9. Determine the status of a specified file in the print queue.
10. Print files.

DISPLAYING FILE CONTENTS

The *cat* command:

- Means concatenate and print
- Displays file contents with continuous scrolling
- Has the form:

  `cat [-n -s]` *filename ...*

  `% cat /usr/lib/aliases`

  - Use *-n* to display a line number before each line
  - Use *-s* to eliminate multiple consecutive blank lines

- For information on other options, see *cat*(1) in the *ConvexOS Programmer's Manual.*

Using *cat*

Display the contents of */usr/lib/aliases* using the *cat* command. Your entry looks like this:

```
% cat /usr/lib/allases
#
# Don't change /usr/lib/aliases without also changing /usr/lib/aliases.common
# on the VAX or else /usr/lib/aliases.base here.
#
#
# Aliases in this file will NOT be expanded in the header from
# Mail, but WILL be visible over networks or from /bin/mail.
#
#       >>>>>>>>>>> The program "newaliases" must be run after
#       >> NOTE >> this file is updated for any changes to
#       >>>>>>>>>>> show through to sendmail.
#

# Basic system aliases -- these MUST be present
MAILER-DAEMON: postmaster
%
```

Your screen continues scrolling unless you stop or abort the output.

Display the same file with line numbers. Your entry looks like this:

```
% cat -n /usr/lib/allases
  1 #
  2 # Don't change /usr/lib/aliases without also changing /usr/lib/aliases.common
  3 # on the VAX or else /usr/lib/aliases.base here.
  4 #
  5 #
  6 # Aliases in this file will NOT be expanded in the header from
  7 # Mail, but WILL be visible over networks or from /bin/mail.
  8 #
  9 #    >>>>>>>>>>>  The program "newaliases" must be run after
 10 #    >> NOTE >>   this file is updated for any changes to
 11 #    >>>>>>>>>>>  show through to sendmail.
 12 #
 13
 14 # Basic system aliases -- these MUST be present
 15 MAILER-DAEMON: postmaster
%
```

Your screen continues scrolling unless you stop or abort the output.

DISPLAYING FILE CONTENTS (cont.)

The *more* command:

- Displays file contents one screenful at a time
- Has the form:

  `more` *[-c -d other_options] [-n] [+linenumber] [+/pattern] filename ...*

  The options:

c	Displays each page beginning at the top of the screen instead of scrolling
d	Integer which specifies number of lines to display
f	Do not fold long lines
+linenumber	Begins displaying at specified file line
+/pattern	Begins display 2 lines before the line containing the specified pattern
other options	See the *more*(1) manual page

  To display 10 lines at a time beginning at line 25 and draw the screen, use:

  `more -c10 +25` *filename*

- To use the help function within *more*, type: `h`

Using *more*

Display the file */usr/lib/aliases* in segments of 5 lines, beginning at the entry *bugs* and use the option to specify redraw the screen. After displaying two or three segments, abort the command. Your entry looks like this:

```
% more -c5 +/bugs /usr/lib/aliases
...skipping
Anonymous:     anonymous
anonymous: "|/usr/spool/notes/.utilities/nfmail anonmail "
bugs: "|/usr/spool/notes/.utilities/nfmail bugs"
buglist: "|/usr/spool/notes/.utilities/nfmail buglist"

--More--(3%)
%
```

Your screen will display 5 lines each time you strike the space bar. CTRL-c does not display when you type it.

Use the *more* command to display the contents of */etc/passwd*. At the end of the first screenful, display the *help* explanations for the *more* command. After displaying the *help* screen, abort the *more* command. Your entry looks similar to this:

```
% more /etc/passwd
root:MKSBXkDL6m01Y:0:10:Superuser:/:/bin/csh
daemon:*:1:1:Our friend, the daemon:/:/bin/csh
resource:xx:3:10:Owns all resources:/usr/access:/bin/csh
SCREENFUL OF LINES DISPLAYS
--More--(4%)
h (this entry calls up the help screen)

Most commands optionally preceded by integer argument k. Defaults in brackets.
Star (*) indicates argument becomes new default.
-------------------------------------------------------------------------------
<space>                 Display next k lines of text [current screen size]
z                       Display next k lines of text [current screen size]*
<return>                Display next k lines of text [1]*
d or ctrl-D             Scroll k lines [current scroll size, initially 11]*
q or Q or <interrupt>   Exit from more
s                       Skip forward k lines of text [1]
f                       Skip forward k screenfuls of text [1]
t (or ')                Go to top of file (or to where previous search started)
=                       Display current line number
/<regular expression>   Search for kth occurrence of regular expression [1]
n                       Search for kth occurrence of last r.e [1]
!<cmd> or :!<cmd>       Execute <cmd> in a subshell
v                       Start up editor at current line [/usr/ucb/vi]
h                       Display this message
ctrl-L                  Redraw screen
:n                      Go to kth next file [1]
:p                      Go to kth previous file [1]
:f                      Display current file name and line number
.                       Repeat previous command
-------------------------------------------------------------------------------
--More--(14%)
%
```

DISPLAYING FILE CONTENTS

The *less* command:

- Displays file contents
- Similar to *more* command
  - Faster
  - Allows backward and forward movement in a file
- Uses commands based on *more*, *emacs*, *vi*
- Has the form:

  `less` *[options] [-n] [+linenumber] [+/pattern] filename ...*

  Some options:

  `-Q` *less* won't beep when you enter incorrect commands

  `-p` Repaints the screen instead of scrolling

- To display a help screen, first type your command:

  `less` *filename(s)*

  then type `H`

Using *less*.

Display the file */usr/lib/aliases* using the *less* command. Then access the *less* help information. Your entry looks like this:

```
% less /usr/lib/aliases
# Don't change /usr/lib/aliases without also changing /usr/lib/aliases.common
# on the VAX or else /usr/lib/aliases.base here.
#
#
# Aliases in this file will NOT be expanded in the header from
# Mail, but WILL be visible over networks or from /bin/mail.
#
#    >>>>>>>>>>    The program "newaliases" must be run after
#    >> NOTE >>    this file is updated for any changes to
#    >>>>>>>>>>    show through to sendmail.
#

# Basic system aliases -- these MUST be present
/usr/lib/aliases h

f, SPACE     Forward one screen.
b            Backward one screen.
e, j, CR   * Forward N lines, default 1.
y, k       * Backward N lines, default 1.
d          * Forward N lines, default 10 or last N to d or u command.
u          * Backward N lines, default 10 or last N to d or u command.
r          Repaint screen.
g          * Go to line N, default 1.
G          * Like g, but default is last line in file.
=          Print current file name
/pattern   * Search forward for N-th occurrence of pattern.
?pattern   * Search backward for N-th occurrence of pattern.
n          * Repeat previous search (for N-th occurrence).
q          Exit.
"More help... (Press RETURN)"
```

DISPLAYING FILE CONTENTS

- *less* commands:

`b` or `^b`	Scroll backwards one screen; prefixing command with a number (`10b`), scrolls backward that number of lines
SPACE	Scrolls forward one screenful
RETURN	Scrolls forward one line; prefixing command with a number, scrolls forward that number of lines
`^L` or `r`	Redraws the screen
E	Examine a new file
/pattern	Search forward for *pattern*; typing /super when you are viewing a file with *less* causes *less* to scroll to the line containing *super*
`N` or `P`	Allows movement between the files, when you supply more than one filename on the command line

- Changing the *less* prompt:

`-m`	Prompt like *more*	`less -m` *filename(s)*
`-M`	Prompt like *more* with file size	`less -M` *filename(s)*
`P`*options*	Prompt with additional selections `F` filename `O` file *n* of *n* `p` percent into file	`less -PmFOp` *file1 file2*

Using *less* Commands

Experiment with the various commands that appear on the *less* help screen.

Use *less* with various prompt commands to discover how the information line (bottom of screen) changes. A colon is your prompt for entering a command; the default information line is the filename. Enter the command that will display an information line and prompt that are similar to *more* on two or more files of your choice. Your entry looks similar to this:

```
% less -m file1 file2
The CONVEX supercomputer offers the design engineer analysis
power at price and performance levels new to the industry for
increased and improved productivity. By
using a combination of widely accepted software and hardware
standards, implemented with advanced technology,
the CONVEX C-1 supercomputer offers the scientific user access to
supercomputer performance for less than $1,000,000.
...
...
....
file1 (file 1 of 2) (60%)
```

INFORMATION HANDLING

The *head* command:

- Displays the first 10 lines of a file (by default)
- Has the form:

  head [*-n-*] *filename(s)*

  % head /usr/lib/aliases

```
#
# Don't change /usr/lib/aliases without also changing /usr/lib/aliases.common
# on the VAX or else /usr/lib/aliases.base here.
#
#
# Aliases in this file will NOT be expanded in the header from
# Mail, but WILL be visible over networks or from /bin/mail.
#
#   >>>>>>>>>>  The program "newaliases" must be run after
#   >> NOTE >>  this file is updated for any changes to
```

- The form head *-number* allows you to specify the number of beginning lines to display

  % head -5 /usr/lib/aliases

```
#
# Don't change /usr/lib/aliases without also changing /usr/lib/aliases.common
# on the VAX or else /usr/lib/aliases.base here.
#
#
```

Displaying the Beginning Lines of a File

Display the default number of lines of the */usr/lib/aliases* file using the *head* command. Your entry:

```
% head /usr/lib/aliases
#
# Don't change /usr/lib/aliases without also changing /usr/lib/aliases.common
# on the VAX or else /usr/lib/aliases.base here.
#
#
# Aliases in this file will NOT be expanded in the header from
# Mail, but WILL be visible over networks or from /bin/mail.
#
#                      >>>>>>>>>>  The program "newaliases" must be run after
#                      >> NOTE >>  this file is updated for any changes to
%
```

Display only the first five lines of the file /dev/MAKEDEV. Your entry:

```
% head -5 /dev/MAKEDEV
#! /bin/sh
#
# Device "make" file.  Valid arguments:
#                      std   standard devices
#                      local  configuration specific devices
%
```

INFORMATION HANDLING (cont.)

The *tail* command:

- Displays the last 10 lines of a file (by default)
- Has the form:

  `tail` [*options*] *filename(s)*

```
% tail /usr/lib/aliases

user1: convext!user1
user2: convexw!user2
user3: convexs!user3
user4: convext!user4
user5: convexc!user5
user6: convex1!user6
user7: artemis!user7
user8: convexs!user8
user9: artemis!user9
user10: convex1!user10
```

- To display a specific number of lines, use `tail` *-number filename*

```
% tail -5 /usr/lib/aliases

user6: convex1!user6
user7: artemis!user7
user8: convexs!user8
user9: artemis!user9
user10: convex1!user10
```

Displaying Ending Lines of a File

Display the default number of ending lines of */dev/MAKEDEV*. Your entry:

```
% tail /dev/MAKEDEV
                                                                    ;;
                        esac
                        ;;
local)
                        sh MAKEDEV.local
                        ;;
*)
                        echo unknown device type $i;;
esac
done
%
```

Use another option of the *tail* command; display the last 15 lines of the */dev/MAKEDEV* file. Your entry:

```
% tail -15 /dev/MAKEDEV
/etc/mknod ${name}${unit} b $bmajor $unit;
chmod 600 r${name}${unit} ${name}${unit}
                        ;;
                        *)
echo bad unit for $name in: $i
                        ;;
                        esac
                        ;;
local)
                        sh MAKEDEV.local
                        ;;
*)
                        echo unknown device type $i;;
esac
done
%
```

INFORMATION HANDLING (cont.)

The *wc* command:

- Determines the number of words, lines, and characters of a specified file
- Has the form:

  wc [*-l -w -c*] *filename(s)*

  ```
  % wc test1
      6     38    211 test1
  ```

- To display the line count, use:

  wc -l *filename(s)*

- To display the word count, use:

  wc -w *filename(s)*

- To display the character count, use:

  wc -c *filename(s)*

Determining Word, Line, and Character Count

If you have a file named *test1*, determine the line, word, and character count. Your entry:

```
% wc test1
   6 38 211 test1
%
```

Determine only the word count of the */dev/MAKDEV* file. Your entry:

```
% wc -w /dev/MAKEDEV
   755 /dev/MAKEDEV
%
```

Determine the total line count for the files */usr/lib/aliases* and */dev/MAKEDEV*. Your entry:

```
% wc -l /usr/lib/aliases /dev/MAKEDEV
   1087 /usr/lib/aliases
    201 /dev/MAKEDEV
   1288 total
%
```

INFORMATION HANDLING (cont.)

The *cmp* command:

- Compares two files
- Determines and prints by byte and line number the first difference between two files
- Has the form:

  `cmp` *filename1 filename2*

```
% cmp states1 states2
states1 states2 differ: char 1, line 1
```

Comparing Files

Before attempting this exercise, you must have two files, *states1* and *states2*. If you do not have these files, create them now with your favorite editor. These files contain the lines:

states1	**states2**
Maine	Texas
Montana	Montana
Nebraska	Illinois
Illinois	Alabama
Iowa	Maine

Determine the first difference between the two files *states1* and *states2*. Your entry:

```
% cmp states1 states2
states1 states2 differ: char 1, line 1
%
```

INFORMATION HANDLING (cont.)

The *diff* command:

- Finds the difference between two files or directories
- Outputs changes to make file 1 identical to file 2
  - Output format:

    a = append; c = change; d = delete

    < = identifies line from first file

    > = identifies line from second file
- Has the form:

  `diff [ -l -s -r ] file1 file2`

```
% diff states1 states2
1c1
< Maine
---
> Texas
3d2
< Nebraska
5c4,6
< Iowa
---
> Alabama
> Maine
>
```

Finding File Differences

Determine what changes need to be made to *states1* to make it identical to *states2*. Your entry:

```
% diff states1 states2
1c1
< Maine
---
> Texas
3d2
< Nebraska
5c4,6
< Iowa
---
> Alabama
> Maine
>
%
```

What does each line mean?

The first entry indicates change Maine in the first file to Texas. The second entry means delete Nebraska from the first file. The third entry denotes change Iowa to Alabama and add Maine and a carriage RETURN to the first file.

MANIPULATING TEXT FILES

The *sort* command:

- Sorts text files (or standard input) alphabetically or numerically
- Sorts uppercase letters before lowercase letters
- Sorts numbers by the first digit
- Sorts and merges files together when used with more than one file
- Sends output to the screen
- Has the form:

  `sort [ -b -d -f -n -o -r -u ] filename(s)`

```
% sort states1
Illinois
Maine
Montana
Nebraska
iowa
```

Sorting Text Files

Sort the file *states1*. Your entry:

```
% sort states1
Illinois
Iowa
Maine
Montana
Nebraska
%
```

What happens when you sort two files together? Sort *states1* and *states2* together. Your entry:

```
% sort states1 states2
Alabama
Illinois
Illinois
Iowa
Maine
Maine
Montana
Montana
Nebraska
Texas
%
```

Create a file named *states3* with the following lines, using your favorite editor.

1
2
10
Illinois
iowa
Maine
montana
Nebraska

Now sort the file; notice how words with lowercase and numerals are handled. Your entry:

```
% sort states3
1
10
2
Illinois
Maine
Nebraska
iowa
montana
%
```

MANIPULATING TEXT FILES (cont.)

The *sort* command options:

`sort -b` *filename*	Ignores initial blanks
`sort -d` *filename*	Sorts in dictionary order, using only letters, digits, and blanks to determine order
`sort -f` *filename*	Ignores case differences
`sort -n` *filename*	Sorts in numeric order
`sort` *filename* `-o` *newfile*	Places sorted output in specified file
`sort -r` *filename*	Sorts in reverse order
`sort -u` *filename*	Discards duplicate lines

Using *sort* Options

Sort the file *states3* in numeric order. Your entry:

```
% sort -n states3
Illinois
Maine
Nebraska
iowa
montana
1
2
10
%
```

Sort *states1* and send the output to *states1.new*; sort *states2* and send the output to *states2.new*. View the contents of the files using *more*.

```
% sort states1 -o states1.new
% sort states2 -o states2.new
% more states1.new states2.new
::::::::::::::
states1.new
::::::::::::::

Illinois
Iowa
Maine
Montana
Nebraska

--More--(Next file: states2.new)

::::::::::::::
states2.new
::::::::::::::
Alabama
Illinois
Maine
Montana
Texas
%
```

USING PRINT QUEUES

The *lpr* command:

- Prints the specified file(s) on a printer
- Has the form (for the default line printer):

  `lpr` *filename*

  `% lpr states1`

- To remove a file after printing, use:

  `lpr -r` *filename*

  `% lpr -r states1`

- To specify a printer, use:

  `lpr -P`*printername filename*

  `% lpr -Plp states1`

Using the Line Printer

Print the *states1.new* on the default printer. Your entry:

```
% lpr states1.new
%
```

Print the file *states1* on the default printer, using the command that will remove the *states1* file after it has printed. Your entry:

```
% lpr -r states1
%
```

DISPLAYING PRINT QUEUES

The *lpq* command:

- Displays the jobs currently in a printer queue
- Has the default form:

```
lpq
% lpq
lp is ready and printing
Rank     Owner   Job   Files      Total Size
active   usrj    800   states1    44 bytes
1        smith   809   datafile   13075 bytes
```

- To specify a printer queue, use:

```
lpq -Pprintername
% lpq -Plp
```

Listing Printing Jobs

Display the jobs currently in the default print queue. Your entry:

```
% lpq
lp is ready and printing
Rank  Owner      Job Files                      Total Size
active username   420  (standard input)          37348 bytes
%
```

If no jobs are currently in the queue, `no entries` displays when you check the queue.

REMOVING JOBS FROM THE PRINT QUEUE

The *lprm* command:

- Removes the specified file from the print queue
- Has the form:

  `lprm [-P`*printername*`] `*id*

  id can be any of the following:
  username
  job number
  filename

- To remove smith's job 809 (datafile) from the default printer:

```
% lprm smith
% lprm 809
% lprm datafile
```

Removing Jobs from the Print Queue

Send the following files to the default printer: *states2*, *states1.new* and *states2.new*. Now remove them from the print queue. Your entry will look similar to:

```
% lpr states2
% lpr states1.new
% lpr states2.new
% lprm username
% dfA648 convext dequeued
cfA648 convext dequeued
dfA649 convext dequeued
cfA649 convext dequeued
dfA650 convext dequeued
cfA650 convext dequeued
%
```

You might not see all the jobs dequeued. The number of jobs displayed depends upon how quickly you type and how busy the printer queue is.

FORMATTING OUTPUT FILES

The *pr* command:

- Produces a formatted listing of one or more files
- Format of the *pr* command:

  `pr [`*option*`] . . . [`*file*`] . . .`

- A heading is generated for each page which contains the following:
  - Date file was last modified
  - Name of the file
  - Page number of the listing generated
- Sample options:

  –h Take the next argument as a page header

  ```
  % pr -h "Source Listing" main.f
  ```

  –l*n* Set the page length to be *n* lines instead of the default of 66 lines

  ```
  % pr -160 main.f
  ```

  +*n* Begin printing with page *n*

  ```
  % pr +20 prog.f
  ```

Formatting Listings

View the contents of the *states1* file using the *cat* utility. Then print the same file using the *print* utility. Finally, print the file again using the following command: *print -h "Listing of states1"*. Your entry:

```
% cat states1
Maine
Montana
Nebraska
Illinois
iowa
10
1
5
% print states1


Feb  2 16:41 1987  states1 Page 1


Maine
Montana
Nebraska
Illinois
iowa
10
1
5
. . .
% print -h "Listing of states1" states1


Feb  2 16:41 1987  Listing of states1 Page 1


Maine
Montana
Nebraska
Illinois
iowa
10
1
5
. . .
%
```

Exercises for Chapter 5

1. Print a file of your choice on the default printer. What command did you enter?

2. What is the command to remove a job from the print queue? How do you identify the job to be removed?

3. You need to determine how file1 and file2 differ. What command would you use?

4. Write a command that identifies the first difference in the files *states1.new* and *states2.new*.

5. Sort a file of your choice in reverse order and put the sorted output into *list*. What command did you use?

6. You need to know how many lines of code are included in the file *fact.c*. What would you use for your command?

7. You sent a document to the line printer. Since the printer is not close to your location, you don't want to go to the printer until you are sure it has printed. How can you determine if it has been printed?

8. What type of output does the *cmp* command produce?

9. Determine the word count of the file */usr/lib/aliases*. What command did you use?

10. Sort the */etc/passwd* file, sending the output to standard out. What command did you use.

11. Select a file of your choice; print the file on the *trip* printer. What command did you use?

12. Select one of your files and sort it in reverse order. What command did you use?

13. How can you display the help information for the command *more*?

14. Write the command that displays a file with line numbers prefixed only to each textual line—no numbers prefixed to blank numbers.

15. Write a command that displays the contents of a file named *code* and allows backward movement.

6

Advanced C Shell Usage

Objectives for Chapter 6

After completing Chapter 6, you will be able to:

1. Specify the number of command lines you want to display with *history*.
2. Display a listing of previous command lines.
3. Repeat previous command lines or parts of previous command lines.
4. Modify selected words and events of previous command lines.
5. Edit previous command lines using substitution.
6. Determine what processes are running.
7. Terminate a process.
8. Create specified aliases.
9. Run a job in the background.
10. Stop and start a job.
11. Terminate a job.

HISTORY FUNCTION

The *history* facility:

- Remembers (in an event list) the commands you issue
- Provides a facility for reissuing previous commands
- Allows you to repeat previous commands by event number or by searching for substrings of the actual command
- Allows you to display a list of command lines previously invoked by typing: `history`

ConvexOS responds with a command listing:

```
1   pwd
2   ls -l
3   cd /
4   ls
5   cd
6   ls
7   mkdir test
8   more states1.new
9   history
```

Using the *history*(1) Command

To complete the exercises in this module, you need two files, *states1* and *states1.new*. If you do not have these files, create the files with the contents of your choice. If you create the files at this time or have executed commands, please logout and then log on again.

So that you have history of events to use for the following exercise, enter the following commands:

```
more states1
mkdir practice
cd practice
cd /
cd
ls
pwd
wc states1
```

When you have entered the commands, you are ready to begin the following exercise.

Display a listing of command lines previously invoked. Your entry looks similar to this:

```
% history
1 more states1
2 mkdir practice
3 cd practice
4 cd /
5 cd
6 ls
7 pwd
8 wc states1
9 history
%
```

HISTORY FUNCTION (cont.)

- To change the number of command lines *history* use: `set history=`*n*

  ```
  % set history=50
  ```

  - If this command is typed on a command line, it is active only during the current login session.

  - If you wish to make this change permanent, add this command line to your *.cshrc* file. To cause your newly modified *.cshrc* file to be re-read during a login session, type the following:

    ```
    % source .cshrc
    ```

- To specify the number of command lines to be displayed, use `history` *n*:

  ```
  % history 10
  ```

- To print the event list in reverse order, use the *-r* option `history -r` *n*:

  ```
  % history -r 5
  ```

Changing the *history* Display

Change the number of command lines that *history* saves to 50. Then request to see only 5 lines. Your entry looks similar to:

```
% set history=50
% history 5
 7 pwd
 8 wc states1
 9 history
10 set history=50
11 history 5
%
```

List the last three commands in reverse order. Your entry looks similar to:

```
% history -r 3
12 history -r 3
11 history 5
10 set history=50
%
```

HISTORY FUNCTION (cont.)

Simple command line editing:

- Using ^ at the beginning of a line indicates you wish to modify the last command entered. The ^ allows a search/replace modification of the last command.

- Use a ^ before and after the incorrect characters, followed by the correct characters

- Substitution occurs at the first instance of the pattern

- To change *pc* to *cp* in the command:

  ```
  % pc file1 file2
  ```

  at the *shell* prompt (%), type:

  ```
  % ^pc^cp
  ```

  cp file1 file2 is displayed and the command is executed

Command Line Editing

Enter the following command (press RETURN after entering the command):

who | sort | moer

Correct the command by substituting the correct pattern; your entry looks similar to:

```
% who | sort | moer
moer: Command not found
% ^er^re
who | sort | more
user1  tty01  Sep 23 10:16
user2  tty02  Sep 23 10:13
user3  tty04  Sep 23 10:10
user4  tty05  Sep 23 08:09
...
...
%
```

HISTORY FUNCTION (cont.)

The ! character:

- Alerts ConvexOS that you want to exeucute a previous command
- Causes ConvexOS to check the history list and make appropriate substitutions in the command line
- Using the following history list:

```
1   pwd
2   ls -l
3   cd /
4   ls
5   cd
```

you can repeat the commands:

`!!`	Repeats the last command (cd)
`!4`	Repeats command 4 (ls)
`!l`	Repeats command 4, the most recent command beginning with 'l' (ls)
`!-4`	Repeats command 2; goes back four commands (ls -l)
`!?s?`	Repeats command 4, the most recent command pattern with 's' (ls)

Repeating History Events

Repeat the last command (*who | sort | more*) you entered. Your entry looks similar to:

```
% !!
who | sort | more
user1   tty01   Sep 23 10:16
user2   tty02   Sep 23 10:13
user3   tty04   Sep 23 10:10
user4   tty05   Sep 23 08:09
...
...
%
```

Change your history setting to 10; then do a history listing. Your entry looks similar to:

```
% set history=10
% history
 8 wc states1
 9 history
10 set history=5
11 history
12 who | sort | moer
13 who | sort | more
14 who | sort | more
15 set history=10
16 history
```

Repeat event 8 (*wc states1*). Your entry looks like:

```
% !8
wc states1
  5  5  37 states1
%
```

HISTORY FUNCTION (cont.)

- To make an addition to the preceding command, type: ! ! followed by additional arguments or commands

  Using `ls -l` as the last command typed:

`!! | more`	Reruns the last command and pipes the output through *more* (ls -l | more)
`!! /usr | more`	Reruns the last command, adding */usr* as an additional argument for *ls*. The output is then piped through *more* (ls -l /usr | more)

- To make an addition to a specific command, type: ! *command number* followed by the additional arguments

`!4 | more`	Reruns event 4 and pipes the output through *more*

Modifying History Events

Send the output of the previous event (*wc states1*) to a file named *mod*. Your entry looks similar to:

```
% !! > mod
wc states1 > mod
cat mod
%
```

HISTORY FUNCTION (cont.)

Specifying substitutions for a previous command:

- To make a substitution, you must indicate a search/replace pattern:
  - Type the event identifier followed by a colon
  - Specify an *s* to indicate substitution (search/replace):
  - Type a ^ followed by characters to be changed and another ^ followed by the replacement characters:

```
!5:s^search^replacement
```

- For example, change the `filename` of event 4 to *myfile*:

```
4  ls -l filename
% !4:s^filename^myfile
```

A slash can be used as a delimiter instead of ^:

```
% !4:s/filename/myfile
```

- To verify that the command is correct before executing it, use `:p` immediately following the event identifier

```
% !4:p:s/filename/myfile
emacs myfile
```

If the command is correct, to execute it, type:
`!!`

History Substitutions

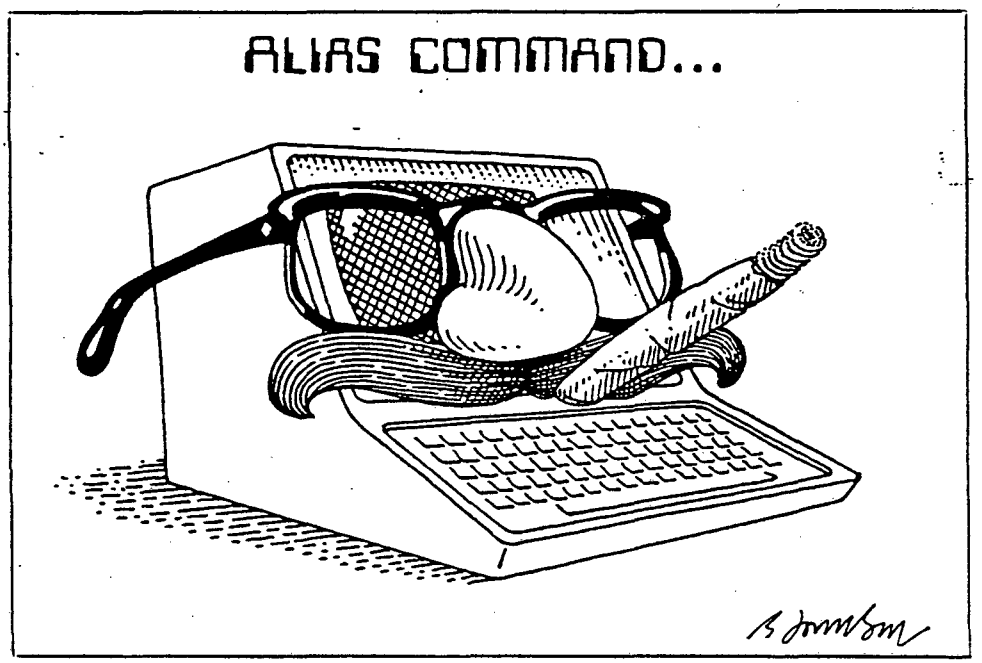
Change event 14 (*who | sort | more*) so that it prints the sorted output of *who*. Verify that your command is correct; then execute it. Your entry:

```
% !14:p:s/more/lpr
% !!
who | sort | lpr
%
```

ALIASES

The *alias* command:

- Allows command renaming or abbreviating
- To display a listing of your aliases, type: `alias` at the *shell* prompt



- To establish an alias for the current terminal session, use:

  `alias` *newname command*

  `% alias copy 'cp -i'`

- To establish a permanent alias, add the alias(es) to your *.cshrc* file or your *.login* file
- To remove an alias, use:

  `unalias` *alias_name*

  `% unalias cp`

Determining Existing Aliases

Display the aliases that are established for your account. Your entry:

```
% alias
%
```

If no aliases are established, the *shell* prompt redisplays without a listing or an explanation.

Create an alias (for this terminal session only) that assures that none of the existing files are overwritten during copying; then verify that you established the alias. Your entry:

```
% alias cp 'cp -i'
% alias
cp   cp -i
%
```

ALIASES FOR COMPOUND COMMANDS

- An alias can be composed of more than one command

- Enclose the compound command in single quotes:

```
alias newname 'command; command'
% alias status 'date; uptime'
```

Compound Aliases

Establish an alias of *nm3* for the command *nitroff -m3* and verify that you established the alias. Your entry:

```
% alias nm3 'nitroff -m3'
% alias
cp  (cp -i)
nm3 nitroff -m3
%
```

You need two files to complete this exercise. If you have two files *states1* and *states1.new*, copy *states1* to *states1.new*. If the alias was established correctly, a prompt `overwrite filename?` displays when you execute the *cp* command. Do not overwrite *states1.new*.

Your entry looks similar to this:

```
% cp states1 states1.new
overwrite states1.new?n
%
```

ALIASES FOR COMPLEX COMMANDS

- Requires the use of \!* for alias(es) using arguments:
  - !* stands for the arguments of the preceding command
  - \ prevents the immediate interpretation of the history substitution so that it takes place when the alias is invoked rather than when it is defined

- Assume you want to set-up an alias for:
  ls -l | more
  - An alias for only your working directory:

```
% alias L 'ls -l | more'
```

  - An alias for any directory:

```
% alias L 'ls -l \!* | more'
```

- The command separation symbol (;) can be used to create aliases that preform several commands in succession:

```
% alias rm 'rm - i \!*; ls'
```

Aliases for Complex Commands

Establish an alias, *L*, that prints a long listing of any directory you supply as an argument to *L*. Your entry looks like this:

```
% alias L 'ls -l \!*'
%
```

Using the alias just created, list the contents of root. Your entry:

```
%L /
total 686
drwxrwxr-x  2 root     1024   Jun 14   15:18   bin
drwxrwxr-x  2 root     3072   Aug  4   11:15   dev
lrwxrwxr-x  1 root       14   May  1   17:13   disco -> /scratch/disco
drwxr-xr-x  2 root     2560   Aug 12   09:31   etc
drwxr-xr-x 10 root      512   Jul 31   21:48   fonts
-rwxrwxr-x  1 root    77824   Jun 26   17:29   hunt.driver
drwxrwxr-x  2 root      512   Jun 25   15:55   lib
drwxr-xr-x  2 root     8192   Nov  5    1985   lost+found
drwxr-xr-x 53 root     1536   Aug  8   17:31   tac
drwxrwxrwx 15 root     1024   Aug 11   23:32   scratch
lrwxr-xr-x  1 root        8   Jul 15   11:20   ssc -> /mnt/ssc
lrwxrwxr-x  1 root        3   Jun  3   06:23   sw -> mnt
lrwxrwxrwx  1 root        7   Jun  6   18:43   sys -> usr/sys
drwxr-xr-x 56 root     1024   Aug 12   13:03   mnt
drwxrwxrwx  6 root     2560   Aug 12   15:33   tmp
drwxrwxrwx 10 root      512   Aug  8   14:57   tmph
drwxr-xr-x 26 root      512   Aug  6   16:34   usr
drwxr-xr-x 28 root     1024   Jul 14   10:23   utd
-rw-rw-r--  1 root   561152   Jun 17   10:25   vmunix
%
```

JOB CONTROL

A command may be executed in either the foreground or the background:

- Foreground jobs
  - May read to and write from the terminal
  - The *shell* waits for a command to complete before prompting the user for a new command
- Background jobs
  - May NOT read from the terminal (will be suspended)
  - May write to the terminal by default
  - The *shell* prompts for a new command without waiting for the command to complete
  - Requires an ampersand (&) immediately following the command:

```
% fc myprog.f &
```

Executing a Command in the Background

Put the command *who | sort > list* in the background. Your entry:

```
% who | sort > list&
[1] 24568
%
[1] Done    who | sort > list
%
```

JOB CONTROL (cont.)

- Background jobs (cont.)
  - Job receives a background job number (*[1]*) and process (ID *22831*); for example:

```
% nitroff -m3 myfile &
[1] 22831
% [1] Done nitroff -m3 myfile
```

  - *Done* is displayed immediately when the job is completed if *notify* is set in *.cshrc* file; otherwise, it is displayed just before the *shell* prompts you for a new command

- A job may be stopped or be switched between the background and foreground modes.
  - Stop your current job with: CTRL-z
  - Attend to other tasks
  - Start this job running again

    Running in the foreground: *fg*

    or

    Running in the background: *bg*

Stopping a Current Job

Start a mail message to yourself with the *Subject:* **Stopping Jobs** and stop the message after entering the following line:

Stop any job by

```
% mail username
Subject: Stopping Jobs
Stop any job by
^Z
Stopped
%
```

If you are not on a blank line when you stop the job (CTRL-Z), the text on the line where you invoked CTRL-Z is deleted.

Restart the job and finish the mail message:

typing CTRL-z.

Your entry:

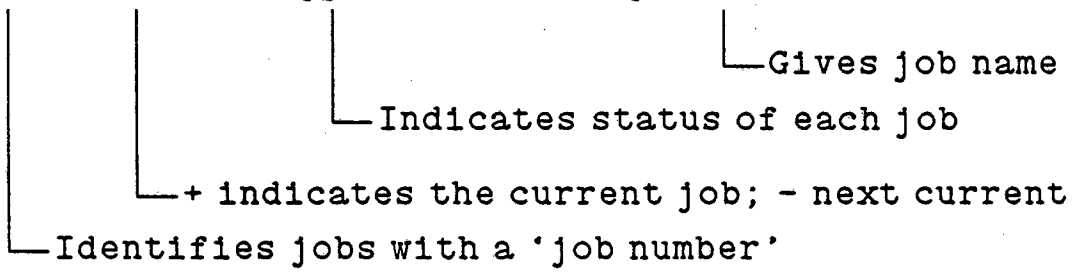
```
% fg
mail username
(continue)
typing CTRL-z.
Cc:
%
```

JOB CONTROL (cont.)

The *jobs* command:

- Lists and labels the jobs initiated at your terminal
- Has the form:

```
% jobs
[1]  +      Stopped        emacs
[2]  -      Running        nitroff -mft file1 -Pswip
[3]         Stopped        who | sort > list
[4]         Stopped        mail person
```



Using the *jobs* Commands

Start the following three processes and stop them with CTRL-z; then display the processes using the *jobs* command.

1. Using your favorite editor, create a file with the following contents:

   This is a stopped job.

   Stop the job with CTRL-z. (For *emacs*, press **ESC-x** and at the :, type **pause**.)

2. Start a mail message to yourself containing one line and then stop the process with CTRL-z.

3. Begin looking at the mail contained in your *mbox* and then stop the process with CTRL-z.

4. List the processes.

When you display the processes, your entry looks similar to this:

```
% jobs
[1]   Stopped          emacs
[2] - Stopped          mail username
[3] + Stopped          mail -f ~/mbox
```

JOB CONTROL (cont.)

- To start the current job (+), type: `fg` at the *shell* prompt

- To start a job (besides) the most recent, use:

  % *%job_number*

  % %4

- To move a job to the background, use either:

  % *job_number* &

  bg *%job_number*

  % %3 & or bg %3

- To stop a job running in background, use:

  stop *%job_number*

  % stop %2

Controlling Jobs

Bring the most recent job to the foreground and then exit from it. Your entry:

```
% fg
mail -f ~/mbox
& q
%
```

JOB CONTROL (cont.)

The *ps* command:

- Prints information about your processes currently in the system
- Outputs:
  - Process identification number - PID
  - The terminal identifier - TT
  - The CPU time used - TIME
  - The state of the process - STAT

    R - running
    T - stopped
    I - Idle
    P, D, S - temporary inactive status
  - The command - COMMAND
- Has the form: ps *[options]*

```
% ps
  PID TT STAT  TIME  COMMAND
 8016 1e  T    0:05 emacs
 8114 1e  T    0:00 mail person
 8122 1e  T    0:00 mail
 8466 1e  R    0:00 ps
```

Checking on a Process

Determine the status of your current processes. Your entry:

```
% ps
  PID TT STAT  TIME COMMAND
 3993 45  T    0:14 emacs
 4111 45  T    0:00 mail username
 5687 45  R    0:01 ps
%
```

CHECKING ON PROCESSES (cont.)

- *ps* options:

a	Displays information about all processes currently in the system
x	Displays information about processes without control terminals, i.e., daemons
l	Displays a long listing of processes
u	Displays associated login name with each process

For additional information on these options and other *ps*(1) options, see the *ConvexOS Programmer's Manual.*

Process Options

Display a listing of all the processes currently on the system, including the username. Your entry looks similar to:

```
% ps -ua
USER    PID  %CPU   %MEM  SZ   RSS  TT STAT  TIME COMMAND
user1   19321  74.4  0.4  272 212   02   R   0:02 ps -ua
user2   19320   1.9  0.1  312  52   07   S   0:00 mail user1
user3   19292   1.8  0.3  400 152   p5   S   0:00 emacs notes
user4   18820   1.6  0.1   60  16   p5   S   0:13 /etc/rlogind
user5   18339   0.8  1.2 1172 652   p4   S   0:03 emacs bonl rnet
```

JOB CONTROL (cont.)

The *kill* command:

- Terminates the specified job
- Has the form:

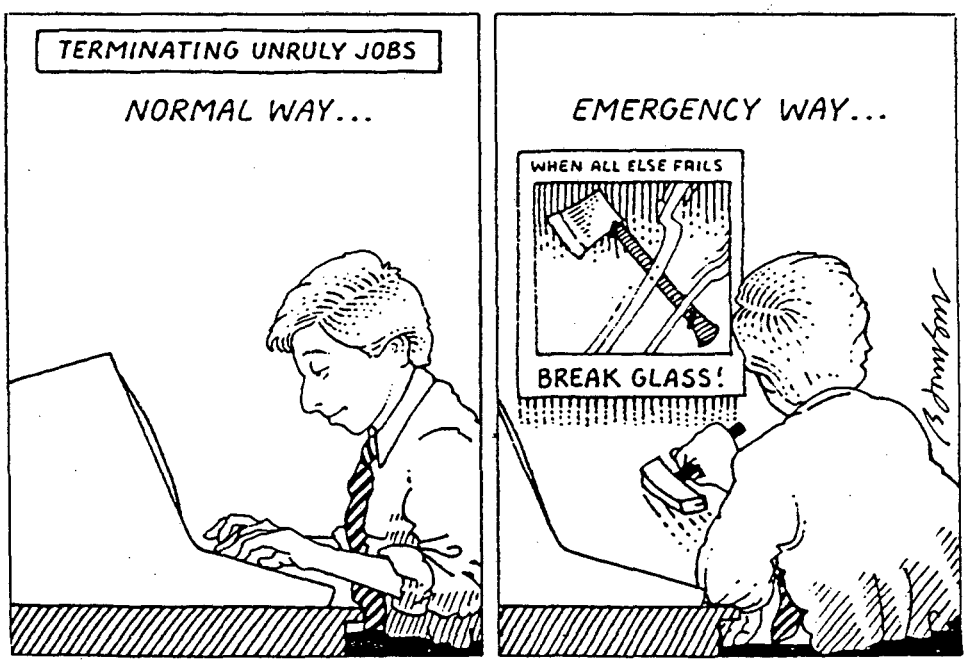
  `kill` *%job number* or *PID*

  `% kill %3`

  `% kill 8122`

- If the command *kill* fails to terminate the job, use:

  `kill -9` *job number* or *PID*



Using the *kill*(1) Command

Previously you stopped three jobs; terminate the remaining stopped jobs. Your entry:

```
% kill %3
[3]  Terminated  mail username
% kill %1
[1]  Terminated  emacs username
%
```

Exercises for Chapter 8

1. Assume your *history* listing contains the event: 10 *who | sort*. Write a command that modifies this event by sending the output to a file named *users*.

2. You made an error typing the command: *tbl file1 | nitroff -m2 -Pswip*; it should read *-m3*. Write a command that would make this change.

3. Create an alias named *cd* that not only changes the directory but also lists the contents of the directory. Verify that the command works correctly by changing your directory to root (/). After executing the alias, use the *pwd* command. If your alais is correct, the response to *pwd* will be */*.

4. Create an alias that assures that you will not accidentally remove an existing file. What command did you issue?

5. A job, number 5, fails to terminate when you use *kill %5*. Write a command that will terminate job 5.

6. Write a command that saves the last to commands you have entered in a *history* list.

7. Using the following history listing, write a command that allows you to execute the following command without entering the entire command: `vi /mnt/smith/source.files/C/gator`

   123 lf
   124 cat /mnt/smith/source.files/C/gator
   125 lpr /mnt/smith/source.files/C/gator.old

8. Write a command that lists the jobs you have working in background mode or suspended at the current time.

9. Enter the following command in the background: `sleep 50`. What did you use to place the command in the background?

10. Suspend a job that is currently executing. What command did you use?

11. Write a command that checks on all the active processes on another user's tty line; for example, communication line tty03. (Hint: use the man pages)

12. List two methods to bring a suspended job to the foreground.

13. List the command to kill job number 3.

14. Enter **alias h 'history 10'** on the command line; how long does this alias remain in effect?

15. Write a command that would change *snoppy* to *snoopy* in the following history listing: **10 cd /mnt/snoppy/prgrams/C**.

16. What does the command **!!** do?

17. Display a list of all the current aliases in effect on your terminal. What command did you use?

18. If you have the alias *cdrm 'cd \!*; rm -i *'*, what happens when you enter the command **cdrm mydir1**?

19. If you had the history event *8 cd /mnt/snoopy/programs*, and you entered this command **more !8^/C**, what happens.?

20. What would the command **stop %2** accomplish?

21. What would the command **fg** accomplish?

7

Mail

Objectives for Chapter 7

After completing this chapter, you will be able to:

1. Read mail received.
2. Send mail to specified system users.
3. Edit mail.
4. Save mail.
5. Determine what your *.mailrc* file controls.
6. Use *talk* to initiate communication with another user.
7. Determine the best way to communicate a message to other system users.

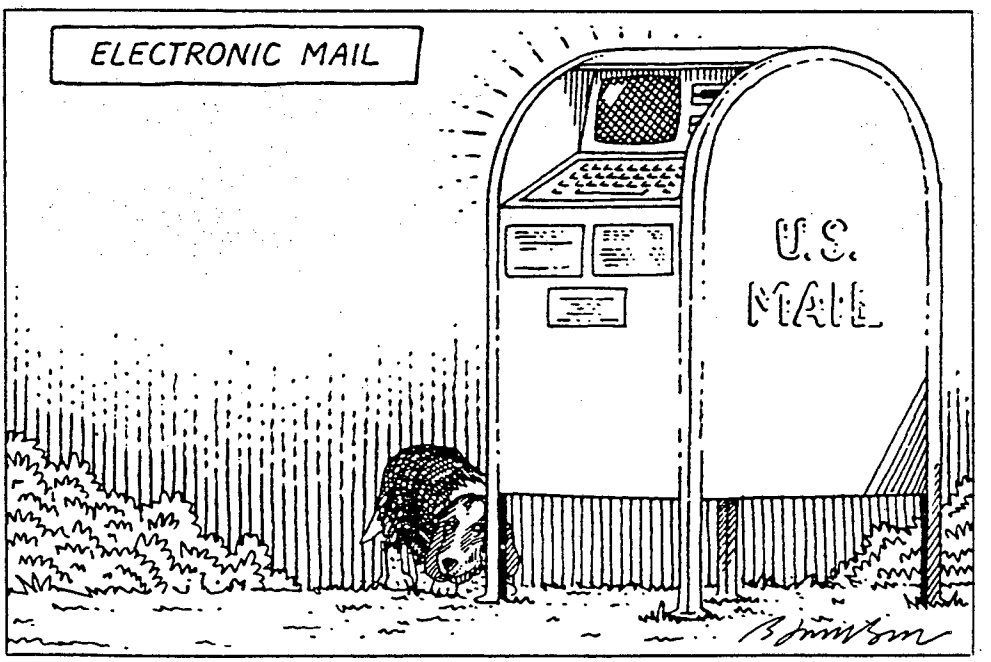
SENDING ELECTRONIC MAIL

- To send mail, use:

  `mail` *username(s)*

  `% mail andy`

- Enter subject of message at Subject prompt
- Enter message text
- To end a message, type: `CTRL-d` or a period (on a new line)
- To send a carbon copy, enter username(s) at `Cc` (if appropriate) or just press RETURN
- To abort a message, type: `CTRL-c` twice



Sending Mail

Send the following mail message (no *Cc* recipient) to yourself. Your subject will be Sending Mail.

> **Start with the mail command; then enter the subject.**
> **The mail message text can be as many lines as you wish. When you have finished the message, press RETURN and type CTRL-d on a new line.**

Your entry looks similar to this:

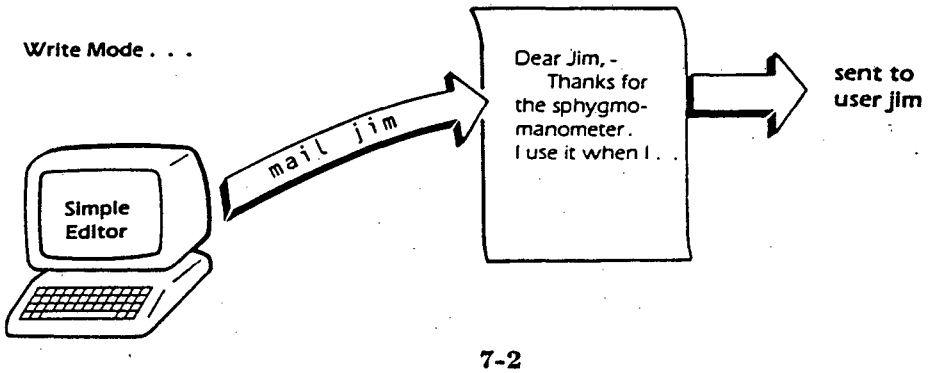
```
% mail username
Subject: Sending Mail
Start with the mail command. Then
enter the subject.
The mail message text can be as many
lines as you wish. When you have
finished the message, press RETURN
and type CTRL-d on a new line.
Cc:
%
```

Send another message to yourself with the subject **aborting mail**. After entering part of the message, abort your mail. The contents:

> **To abort a message, type 2 CTRL-c's. You will be prompted to type another CTRL-c after typing the first one.**

Your entry:

```
% mail username
Subject: Aborting Mail
To abort a message, type 2 CTRL-c's. You
will be prompted to type another CTRL-c
after typing the first one.
^C
(interrupt -- one more to kill letter)
^C%
```



EDITING THE MAIL MESSAGE BODY

- Once *mail* has been invoked, you can select a text editor by typing (at the beginning of a new line):

  `~e` or `~v`

- Interactive editing begins using either *vi* (for *~v*) or the editor specified as the environmental variable or mail editor (for *~e*)
  - To set the default (*~e*) editor, the following line can be added to your *~/.login* file:

    `setenv EDITOR` *pathname_of_editor*
  - You can also create the file *~/.mailrc* with the following line:

    `set editor=`*pathname_of_editor*

- Modify the message

- Save the text and exit the editor

- Mail prints *(continue)*

- Continue entering message (if appropriate)

- Exit mail with `CTRL-d` or period (on a new line)

Editing the Mail Message Body

Send a mail message to yourself. Enter the following text (with errors) and then correct the three misspelled words (allow - allows, editer - editor, mannor - manner).

Subject: **Revising**
Mail allow you to make changes
using your favorite editer or the
default editor. Just call up the
editor with ~e, modify the message, and exit
the editor in your normal mannor.

After you have edited the message, add this line:

See it's easy to edit.

Your entry looks like this (with *ex* as the default editor):

```
% mail username
Subject: Revising
Mail allow you to make changes
using your favorite editer or the
default editor. Just call up the
editor with ~e, modify the message, and exit
the editor in your normal mannor.
~e
"/tmp/Re17250" 5 lines, 178 characters
:1s/allow/allows/
Mail allows you to make changes
:2s/editer/editor/
using your favorite editor or the
:5s/mannor/manner/
the editor in your normal manner.
:wq
"/tmp/Re17250" 5 lines, 179 characters
(continue)
See it's easy to edit.
Cc:
You have new mail.
%
```

EDITING THE MAIL MESSAGE HEADING

Within *mail*:

- To add message recipients, type (on a new line):
  `~t` *additional name(s)*

  `~t sally andy`

- To change the *Subject*, type (on a new line):
  `~s` *new subject*

  `~s This is the new subject`

- To display the entire letter as it now appears, type (on a new line):
  `~p`

- End mail with `CTRL-d` or period (on a new line)

Revising Recipient and Subject Lines

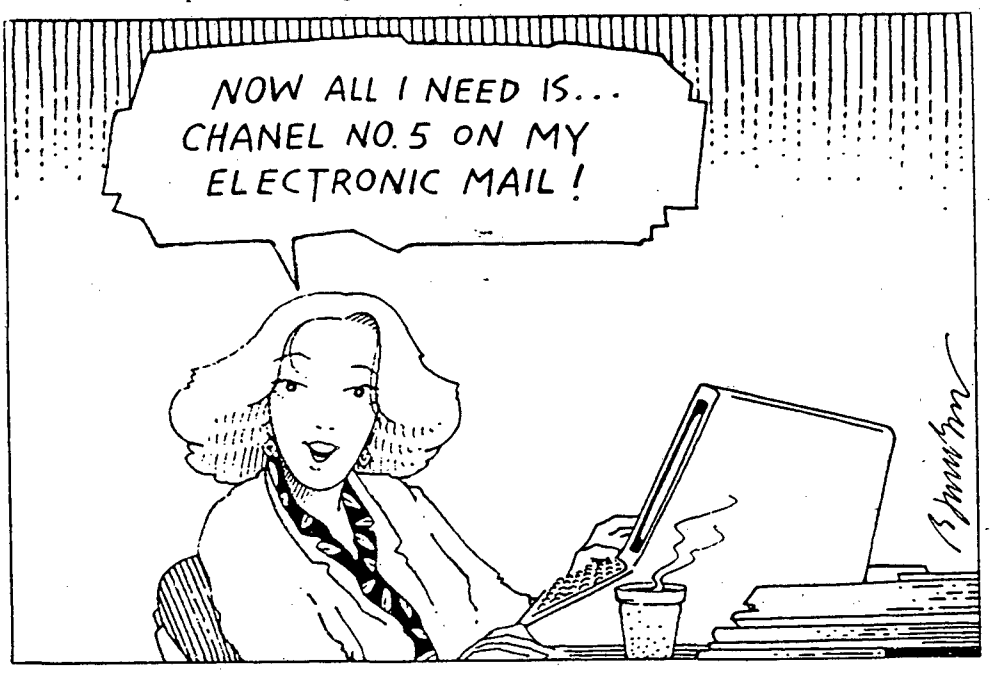
Send the following mail message to yourself with the subject: **More Changes.**

> **After I finish this message, I will**
> **be able to add message recipients**
> **and change a subject line.**

After you have entered the message text, add two recipients: use two of your colleagues' usernames and change the subject to: **More Editing**. Display the changes (before sending the mail) by typing, **~p**; don't send a *Cc*.

Your entry:

```
% mail username
Subject: More Changes
After I finish this message, I will
be able to add message recipients
and change a subject line.
~t floppsy moppsy
~s More Editing
~p
Message contains:
To: username floppsy moppsy
Subject: More Editing

After I finish this message, I will
be able to add message recipients
and change a subject line.
(continue)
Cc:
%
```



EDITING OUTGOING MAIL MESSAGE HEADINGS

- To change *To*, *Subject*, *Cc*, or add a *Bcc*, type: ~h
  - Prompts appear for:

    `To:` *username(s)*

    `Subject:` *Here's the subject*

    `Cc:` *username(s)*

    `Bcc:` *type username or press RETURN*

    `(continue)`

- Pressing RETURN retains contents as displayed
- To change contents, delete characters, words, etc., use BACKSPACE, CTRL-h, CTRL-w, or CTRL-u
- Enter *Bcc* recipient or press RETURN at `Bcc`
- Add to message contents, if appropriate, following `(continue)`
- End mail with `CRTL-d` or a period

Revising Mail Headings

Send the following message to two of your colleagues and **bane** (your instructor) with the subject **Practice Mail**. After you have entered the body, change the spelling of user *bane* to *bayne* on the *To* line; do not change any other lines. Send the *Cc* to yourself and a *Bcc* to a user in your class.

This is a test for sending mail.
Did you receive my message?

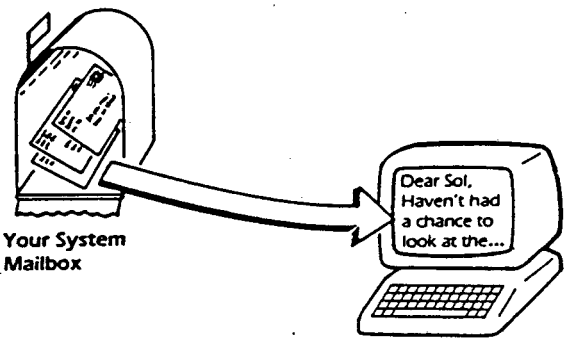
Your entry:

```
% mail username username bane
Subject: Practice Mail
This is a test for sending mail.
Did you receive my message?
~h
To: username username bane
Subject: Practice Mail
Cc: your_username
Bcc: classuser
(continue)
Cc: username
%
```

READING MAIL MESSAGES

The system notifies you of mail with a beep and a message when *biff is y*; otherwise, just a message: `You have new mail.`

Read Mode . . .



- To read mail, type: `mail`
- An informational message is displayed:
  "/usr/spool/mail/username/": 2 messages 2 new

  followed by a list of headers:

  N >1 person Fri Aug 29 20:10 10/123 "Sample Information"
  N 2 person2 Fri Aug 29 20:30 24/656 "Lunch"
- The mail prompt displays: *&*
- Type the number of the message you want to read or press RETURN to display the message with "`>`" pointing to it
  - Your screen displays the selected mail message
- To exit, type: `q`

Reading Mail Messages

Read the first two messages that appear in your mail box. Your entry looks similar to:

```
% mail
"/usr/spool/mail/user/name": 4 messages 4 new
N>1 username  Mon Aug 22 10:45 12/414  "Sending Mail"
N 2 username  Mon Aug 22 10:50 14/301  "Revising"
N 3 username  Mon Aug 22 10:55 14/301  "More Editing"
N 4 username  Mon Aug 22 11:05 14/301  "Practice Mail"

(You may have another piece of mail if you were
named as an addressee in the previous exercise).
& 1 or RETURN
From username Mon Aug 22 10:45:00 1986
Date: Mon, 22 Aug 86 10:45:00 cdt
From: username (Full Name)
To: username
Subject: Sending Mail

Start with the mail command. Then
enter the subject.
The mail message text can be as many
lines as you wish. When you have
finished the message, press RETURN
and type CTRL-d on a new line.
& 2 or RETURN
From username Mon Aug 22 10:50:00 1986
Date: Mon, 22 Aug 86 10:50:00 cdt
From: username (Full Name)
To: username
Subject: Revising

Mail allows you to make changes
using your favorite editor or the
default editor. Just call up the
editor with ~e, modify the message, and exit
the editor in your normal manner.
& q
Saved 2 messages in mbox
Held 2 messages in /usr/spool/mail/username
%
```

MAIL HELP

- To invoke *help* within *mail* (at the & prompt), type:

  ?

A listing of *help* commands display

Mail Commands

t <message list>	type messages
n	goto and type next message
e <message list>	edit messages
f <message list>	give head lines of messages
d <message list>	delete messages
s <message list> file	append messages to file
u <message list>	undelete messages
r <message list>	reply to messages
R <message list>	reply only to sender of message
pre <message list>	make messages go back to /usr/mail
m <user list>	mail to specific users
q	quit, saving unresolved messages in mbox
x	quit, do not remove system mailbox
h	print out active message headers
c [directory]	chdir to directory or home if none given

A <message list> consists of integers, ranges of same, or user names separated by spaces. If omitted, Mail uses the last message typed.

A <user list> consists of user names or distribution names separated by spaces. Distribution names are defined in .mailrc in your home directory.

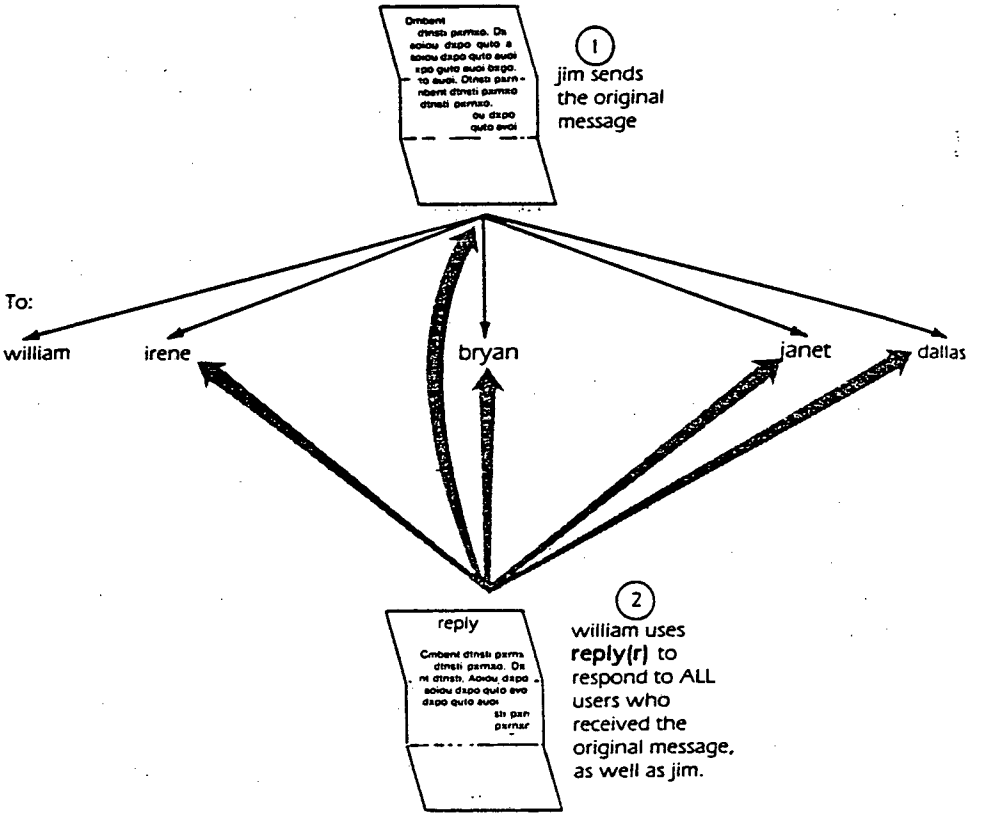
Invoking Mail Help

Display the mail *help* screen; then exit mail. Your entry:

```
& ?
  Mail Commands
t <message list>        type messages
n
e <message list>        edit messages
f <message list>        give head lines of messages
d <message list>        delete messages
s <message list> file   append messages to file
u <message list>        undelete messages
r <message list>        reply to messages
R <message list>        reply only to sender of message
pre <message list>      make messages go back to /usr/mail
m <user list>           mail to specific users
q                       quit, saving unresolved messages in mbox
x                       quit, do not remove system mailbox
h                       print out active message headers
c [directory]           chdir to directory or home if none given
A <message list> consists of integers, ranges of same, or user names separated
by spaces.  If omitted, Mail uses the last message typed.

A <user list> consists of user names or distribution names separated by spaces.
Distribution names are defined in .mailrc in your home directory.
& q
%
```

REPLYING TO MAIL

- To reply to the sender and all recipients, type: `r` at the & prompt
  - Mail displays the header
  - Enter the message
  - End mail with CTRL-d



- To insert a file into mail, type (on a new line): `~r` *filename*
- To include an existing mail message in your mail message, type (on a new line):
  `~m` *[message_number]*
  - To include the mail message marked with "`>`" or the last message read, use: `~m`

Replying to Mail - Sender and Recipients

You have two (or more) unread messages remaining in *mail*. Read mail message 1 and make a reply to the sender and all other named recipients; include the following message.

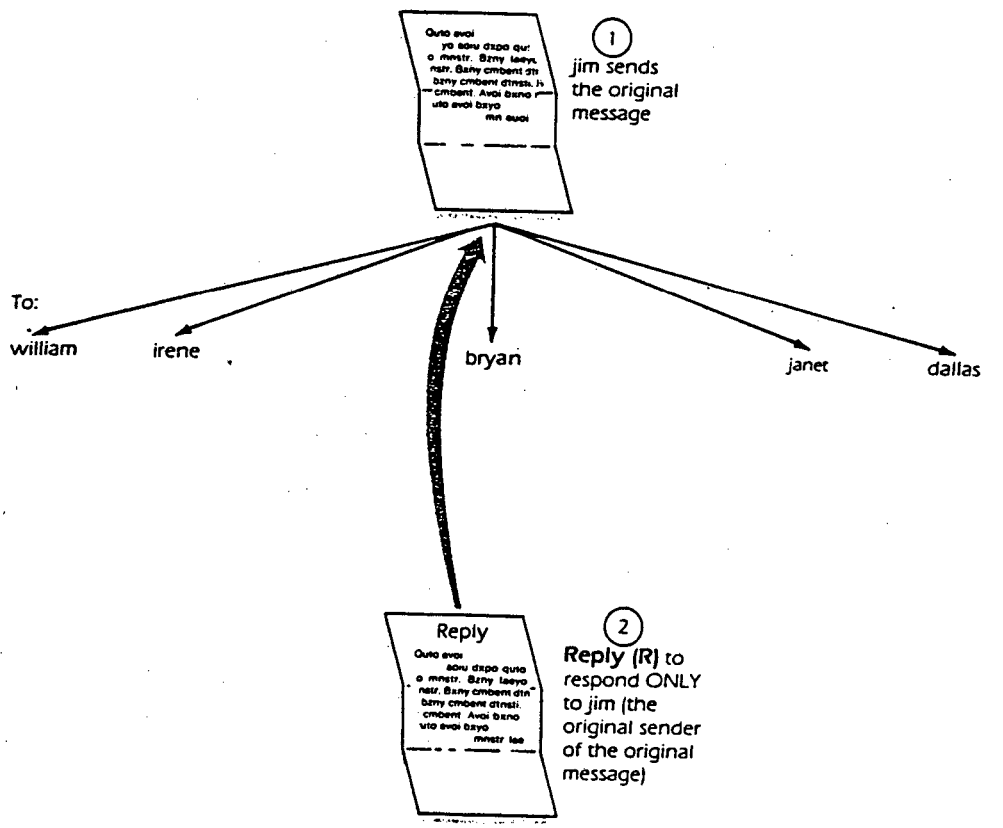
I got your message about revising mail headers.

Your entry looks similar to:

```
% mail
"/usr/spool/mail/user/name": 2 messages 2 unread
U>1 username  Mon Aug 22 10:55 14/301  "More Editing"
U 2 username  Mon Aug 22 11:05 11/213  "Practice Mail"
& 1
From username Mon Aug 22 10:55:00 1986
Date: Mon, 22 Aug 86 10:55:00 cdt
From: username (Full Name)
To: username floppsy moppsy
Subject: More Editing

After I finish this message, I will
be able to add message recipients
and change a subject line.
& r
To: username floppy moppsy
Subject: Re: More Editing
I got your message about revising mail headers.
Cc:
& q
Saved 1 message in mbox
Held 1 message in /usr/spool/mail/username
%
```

REPLYING TO MAIL (cont.)

- To reply to only the sender, type: R at the &
  - Mail displays the header
  - Enter the message
  - End mail with CTRL-d or a period



Replying to Mail - Sender Only

Read mail message 1 and send a reply to only the sender. Your message:

Yes, I received your message.

Your entry looks similar to:

```
% mail
"/usr/spool/mail/user/name": 1 message 1 unread
U 1 username  Mon Aug 22 11:05 11/213 "Practice Mail"
(You may have additional mail lines.)
& 1
From username Mon Aug 22 11:05:00 1986
Date: Mon, 22 Aug 86 11:05:00 cdt
From: username (Full Name)
To: username username bayne
Subject: Practice mail
Cc: username

This is a test for sending mail.
Did you receive my message?
& R
To: username (only sender's name appears)
Subject: Re: Practice mail
Yes, I received your message.
Cc:
& q
Saved 1 message in mbox
Held 1 message in /usr/spool/mail/username
%
```

MAIL SAVED IN MBOX

- By default, mail is saved in *~/mbox* when you exit *mail*

- To read mail from your *mbox*, type:

```
% mail -f
```

  At the & prompt, enter the number of the message you want to read

  All of the previously discussed *mail* commands can be used when accessing your *mbox*

- To retain mail in the system mailbox (not in *mbox*):

  - Change *.mailrc*

  - Add the line:

```
set hold
```

Reading Mail in *mbox*

All the mail that you have read has been saved to a file named *mbox*. Read mail message 4 from your *mbox*. Don't exit mail.

Your entry:

```
% mail -f
"/mnt/username/mbox": 4 messages
& 4
From username Mon Aug 22 11:05:00 1986
Date: Mon, 22 Aug 86 11:05:00 cdt
From: username (Full Name)
To: username username bayne
Subject: Practice mail
Cc: username

This is a test for sending mail.
Did you receive my message?
&
```

SAVING MAIL TO A FILE

- To save the *mail* message or letter you are reading to a file, type (at the & prompt):

  s *file_name*

  & s memos

- To save unread mail to a file, type (at the & prompt):

  s *mail_number file_name*

  & s 2 memos

- To read mail already saved to a file, type:

  mail -f *file_name*

  % mail -f memos

  At the & prompt, enter the number of the message you want to read

Saving Mail to a File

Save message 4, which you just displayed, to the file *memos*. Also, save message 1 to the *memos* file. With the & prompt displayed, your entry looks similar to:

```
& s memos
"memos" [New file] 17/414
& s 1 memos
"memos" [Appended] 14/493
& q
%
```

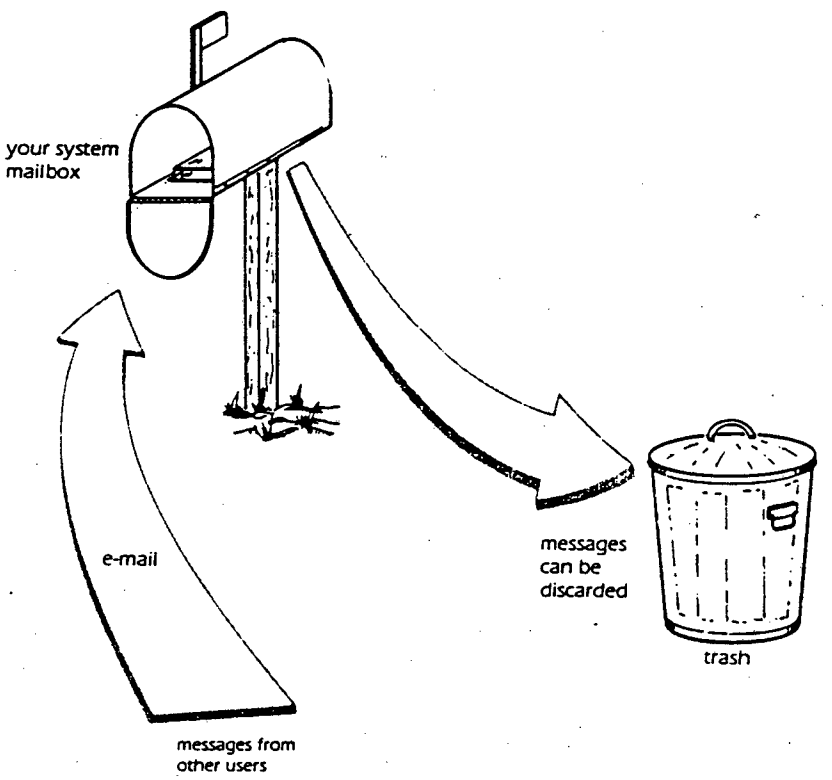
Read the mail messages that you saved to *memos*. Your entry looks similar to:

```
% mail -f memos
"memos": 2 messages
& RETURN
From username Mon Aug 22 11:05:00 1986
Date: Mon, 22 Aug 86 11:05:00 cdt
From: username (Full Name)
To: username username bayne
Subject: Practice mail
Cc: username

This is a test for sending mail.
Did you receive my message?
& RETURN
From username Mon Aug 22 10:45:00 1986
Date: Mon, 22 Aug 86 10:45:00 cdt
From: username (Full Name - Training)
To: username
Subject: Sending Mail

Start with the mail command. Then
enter the subject.
The mail message text can be as many
lines as you wish. When you have
finished the message, press RETURN
and type CTRL-d on a new line.
& q
%
```

DELETING MAIL

- To delete a mail message, immediately after reading it, type `d` at the & prompt:

  `& d`



- To cleanup *mbox* and mail files:
  - Execute `mail -f` or `mail -f mailfile`
  - At the & prompt:

    `d` *message_number(s)_to_delete*

    `& d 2`

Deleting Mail Messages

Delete the two messages in the folder *trnmsgs* and exit. Your entry:

```
& d 1 2
& q
"/mnt/username/mailsamples/trnmsgs" removed
%
```

NEW MAIL NOTIFICATION

The *biff* command allows you to specify the type of mail notification.

- To display *biff* status, type: `biff`
- If set to *n* (the default):
  - No announcement or interruption
  - `You have mail` displays at the *shell* prompt; no beep
- If set to *y*:
  - System beeps when mail arrives
  - System displays the heading and first few lines of the message when mail arrives
- Set either option for a terminal session by typing: `biff y` or `biff n` at the *shell* prompt
- To change the default setting, add the following line to to your *~/.login* file:

```
biff y
```

Displaying Mail Notification

Determine the status of your mail notification. If it is set at *n*, change it to **y** (for this terminal session) and verify that you have changed it to *y*. Your entry looks similar to:

```
% biff
is n
% biff y
% biff
is y
%
```

How would you set mail notification to be *y* for all terminal sessions?

THE *.mailrc* FILE

The *.mailrc* file:

- Can be altered to control your *mail* environment:
  - Set default mail editor

    `set editor=`*pathname_of_editor*
  - Use system mailbox

    `set hold`
  - Create *mail* aliases

    `alias bbgrp user1 user2 user3`
  - Set length of paging for message

    `set crt=22`

Revising the *.mailrc* File

You don't want long mail messages to scroll continuously. Make an entry in the *.mailrc* file that will page messages longer than 22 lines. Your entry looks similar to the following using the ex editor.

```
% ex .mailrc
".mailrc" 1 line, 23 characters
:a
set crt=22
.
:wq
".mailrc" 2 lines, 34 characters
%
```

SENDING A MESSAGE TO ALL USERS

- To send a message to all users, type:

  `mail msgs`

  - Use same procedure as *mail* to transmit message

- When you are notified that you have messages, read the message(s) by typing: `msgs`

  - A message header and (*number* lines) [ynq] are displayed

  - To read the message, press RETURN or type: `y`

  - To skip a message, type: `n`

  - To save a message, answer [ynq] with: `s` *filename*

  - To exit messages, type: `q`

Sending Messages to all Users

Send the following message to all users on the system. Since it is not urgent, send the message so the users can read the message at their leisure. The *Subject*: **Sample Message**; the message:

This is a sample of a message that is being sent to all users.

Your entry looks similar to this:

```
% mail msgs
Subject: Sample Message
This is a sample of a message that is
being sent to all users.
%
```

USING *talk*

The *talk* program:

- Allows you to converse on your screen with another user
- Invoked by typing:

  `talk` *username*

  `% talk andy`

- Rings other user by printing:

  `talk: connection requested by` *username*
  `talk: respond with: talk` *username*

- Informs sender if other person is not logged on; no connection
- Terminated by either party with `CTRL-c`
- To prevent *talk* messages, add this line to your *.login*

  `mesg n`

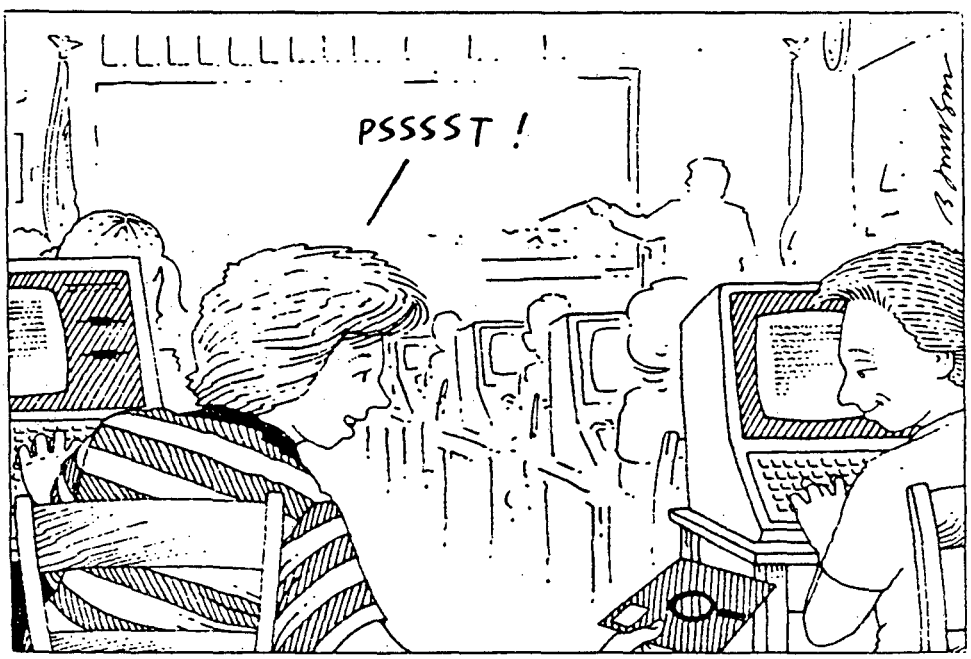
Communicating with *talk*

Using *talk* send the following message to one of your colleagues. When you have received a reply, terminate the connection.

It is 11:30. Are you ready for lunch?

Your entry looks similar to:

```
% talk username
(This is the sender's screen)
[No connection yet]
[Waiting for your party to respond]
[Ringing your party]
[Connection established]
It is 11:30. Are you ready for lunch?
(Type CTRL-c to terminate connection)


_____________________________________________
(This also displays on the receiver's screen)
Message from Talk_username@convext at 10:05
talk: connection requested by username@convext
talk: respond with:  talk username@convext
talk username
Yes, I am hungry; let's go now.

%
```



Exercises for Chapter 7

1. Write a command to send a message to all users on the system to be read at their leisure.

2. How can you display the mail *help* information?

3. What file would you edit to set your default mail editor to be *edt*? What would you add to this file?

4. What are the two ways you can end a mail message?

5. Send a mail message to your instructor containing the following information:

   a. Subject: **Mail Test**
   b. Message: **The information you requested is contained in this message. The list of states follows:**
   c. Append a file: Add the file **states1.new** to this message.
   d. Change the Subject: **Reference Information**
   e. Change the message to read: **Yes, I do have the information you need. I am enclosing the file for your reference.**
   f. Send a copy to yourself.

6. You want to communicate interactively with another user on the system. How would you accomplish this task?

7. Write a command that inhibits being interrupted when mail arrives.

8. You want to send a message to all users telling them that a party is being held on Friday at 5:00. What command would you use to send the message?

9. How do you terminate a "talk" session?

10. What is the purpose of the *biff* command?

11. Write a command that inhibits others from doing a *talk* command to your terminal.

12. You are composing a mail message and decide you need to make some revisions. What command do you enter to use the default editor?

13. Write a command that would insert mail message #4 into the mail message you are composing.

14. You are reading a mail message that was sent to several users and you want your reply to go to only the originator, what command do you use?

15. You have made several changes to a mail message and wish to see the contents before sending it; what command displays the current message contents?

8

Redirecting Input and Output

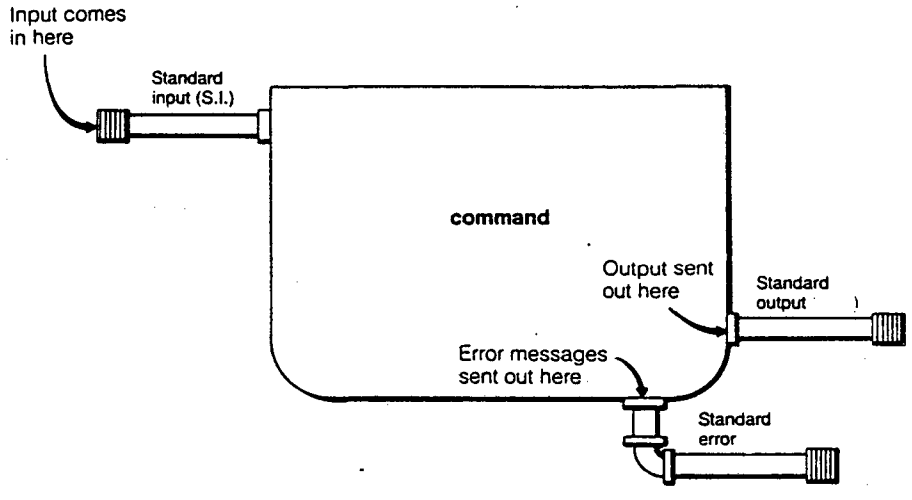
Objectives for Chapter 8

After completing this chapter, you will be able to:

1. Use the redirection command to create a file.
2. Redirect output of a command to a file.
3. Append the output of command to an existing file.
4. Redirect input of a file or command.
5. Connect two or more commands on a command line with pipes.
6. Use a filter to manipulate the contents of a file.
7. Direct the output of a command simultaneously to a file as well as to the terminal monitor.

REDIRECTING I/O

- Standard output can be redirected, usually to a file



- Use the > after the command to redirect the output:

  command > *filename*

  % `who > list`

- Redirecting output to an existing file overwrites the "existing" contents
  - Add `set noclobber` to your *.cshrc* file or on the command line to keep redirection from overwriting an existing file (can override with ! after redirection indicator)
  - To unset the noclobber option, remove it from the *.cshrc* file or use `unset noclobber` on the command line

Redirecting Output

Redirect the standard output of the *who* command to a file named *wholist*. Your entry:

```
% who > wholist
%
```

Set the *noclobber* option on the command line. Now redirect the output of the *date* command to *wholist*. What happened when you entered the command? Your entry:

```
% set noclobber
% date > wholist
wholist: File exists.
%
```

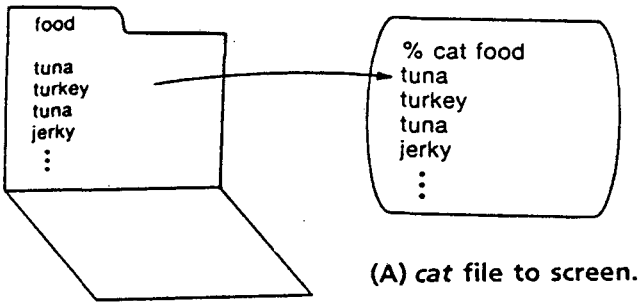
Now unset *noclobber*. Your entry:

```
% unset noclobber
%
```

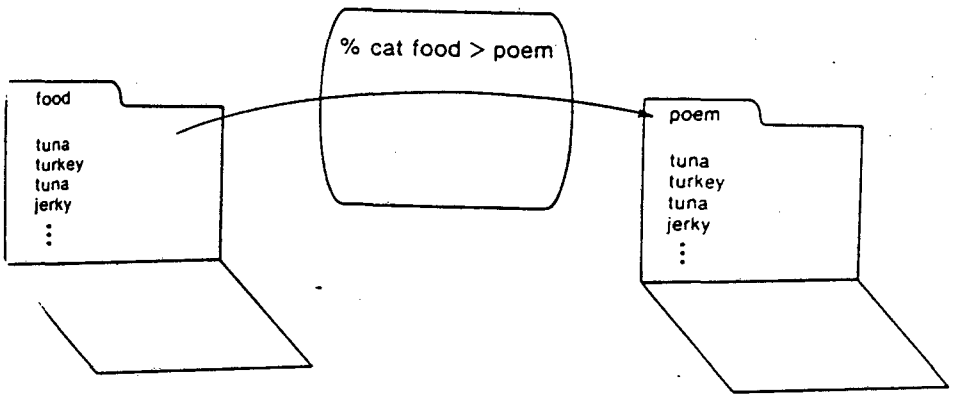
COMBINING CONTENTS OF A FILE

- The output of a file can be redirected to another file with the *cat* command

```
cat food > poem
```



(A) *cat* file to screen.



(B) *cat* file to file using redirection.

- To combine the contents of two files in a new file, use:

```
cat filename1 filename2 > filename3
% cat sample1 sample2 > example
```

Error messages (if any) are routed to the terminal

Combining Files

Write the contents of the files *wholist* and *states1* to the file named *users*. Display the contents of *users* with the *more* command. Your entry:

```
% cat wholist states1 > users
% more users
user 1  tty01  Sep 24 15:30
user 2  tty06  Sep 24 08:35
user 3  tty08  Sep 24 15:13
user 4  tty09  Sep 24 14:51
user 5  tty10  Sep 24 13:01
user 6  tty11  Sep 24 15:00
user 7  tty12  Sep 24 13:33
Maine
Montana
Nebraska
Illinois
Iowa
%
```

Now redirect the output of the *date* command to *users*. Display the contents of *users* with the *more* command. What happened to the contents of *users*? Your entry:

```
% date > users
%more users
Fri Oct 17 14:07:58 CDT 1986
%
```

REDIRECTING I/O (cont.)

- To append the standard output of a command to another file, use:

  command >> *filename*

  % date >> example

  Output of *date* is appended to *example*

  Any error messages are routed to the terminal

- To redirect standard output and standard error output to a file, use:

  cat *filename(s)* >& *filename*

  % cat sample example >& combo

  The contents of both *sample* and *example* are written to *combo* as well as any error messages

- To append both standard output and standard error output to a file, use:

  cat *filename(s)* >>& *filename*

  % cat sample example >>& combo

  The contents of both *sample* and *example* are appended to *combo* as well as any error messages

Appending Files Contents

Append the date to the *states1.new* file. Display the contents of *states1.new*. Your entry:

```
%date >> states1.new
%more states1.new
Illinois
Iowa
Maine
Montana
Nebraska
Wed Sep 24 15:57:21 CDT 1986
%
```

Redirect the standard output and standard error output of the file *fun* to *funny*; then display the contents of *funny* to verify that the standard error output was sent to the file. Your entry looks similar to this:

```
%cat fun >& funny
%more funny
fun: No such file or directory
%
```

REDIRECTING I/O (cont.)

- To redirect standard input from a file, use:

  command < *filename*

  % ex < scriptname

  scriptname is the standard input into a program named *ex*

  Error messages (if any) are routed to the terminal

- To use the current standard input file as input until the *string* is found on a line by itself

  command << *string*
  . . .
  . . .
  . . .
  string

Redirecting Standard Input

You can send the standard input of a file through mail using the command **mail -v** *username* < *filename*. Using this command, mail yourself the *states1.new* file. Your entry looks similar to this:

```
%mail -v username < states1.new
username ... Connecting to local ..
username ... Sent
%
```

The following example illustrates how you can use standard input as the input file until a specified string is found on a line by itself:

```
%more << xxFINISxx
this is a test
xxFINISxx
this is a test
%
```

In this example, when you typed the string, you were unable to continue standard input.

Use the *pr* utility to print to a file named *junk* containing two lines of text:

This is a test of the
'<<' redirection operator

Use *xxENDxx* as your terminator string. View the *junk* file with *more*. Your entry:

```
% pr << xxENDxx > junk
This is a test of the
'<<' redirection operator
xxENDxx
% more junk


Feb 4 13:23 1987  Page 1


This is a test of the
'<<' redirection operator


%
```

REDIRECTING I/O (cont.)

Using pipes:

- Pipes cause the standard output from one process to be transferred into the standard input of another process

- Use the vertical bar character (|) to specify a pipe:

  command | *command*

  To determine how many people are logged onto the system:

  % `who | wc -l`

  To print a list of your files on a line printer:

  % `ls -l | lpr`

- You can combine redirection and pipes

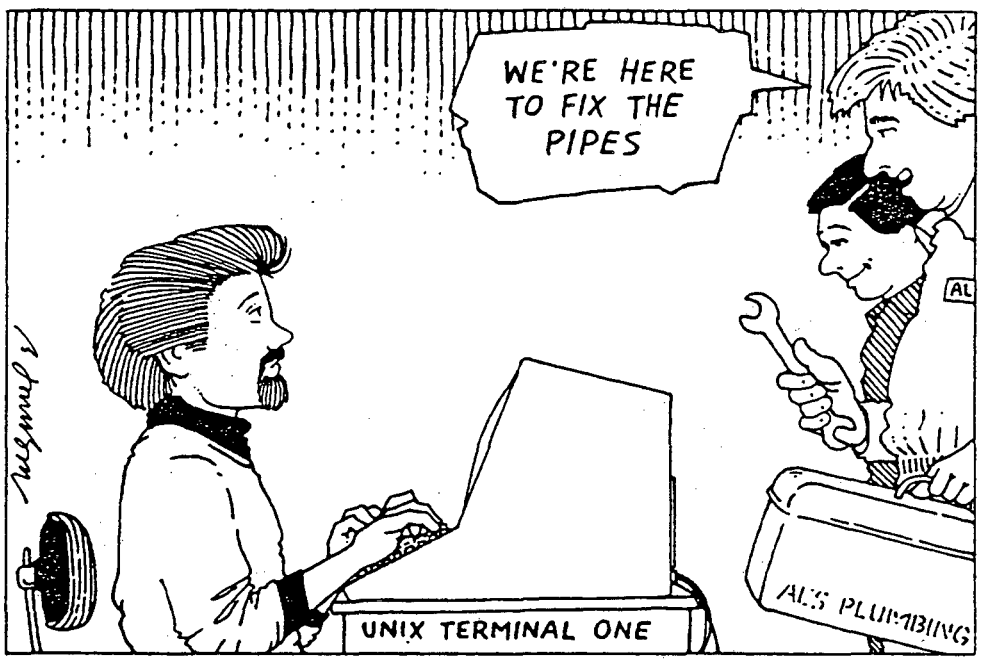
  To transfer a sorted list of users to a file:

  % `who | sort >` *filename*

Using Pipes

Transfer a sorted list of the users currently logged on the system to the file *srtusrs*; display the contents of *srtusrs*. Your entry:

```
%who | sort > srtusrs
%more srtusrs
user 1   tty15  Sep 24 13:32
user 2   tty11  Sep 24 15:00
user 3   tty13  Sep 24 15:01
user 4   tty12  Sep 24 13:33
user 5   tty08  Sep 24 15:13
user 6   tty06  Sep 24 08:35
user 7   ttyp1  Sep 24 09:12  (convex1)
...
...
%
```



REDIRECTING I/O (cont.)

- You can pipe the output of one command into another command

  command | *command* | *command*

  To list the users in alphabetical order and display a screenful of information with pauses:

  % `who | sort | more`

  To print this information:

  % `who | sort | lpr`

- You can pipe both standard output and standard error from one command into another command

  command |& *command* |& *command*

Using Pipes with Multiple Commands

Print an alphabetical listing of the current system users on the default printer. Your entry:

```
%who | sort | lpr
%
```

REDIRECTING I/O (cont.)

Using filters:

- Pipes used in conjunction with UNIX commands that alter the contents of a file are called filters

  command filename | *command(s)*

  To sort and paginate a file and send the results to the line printer:

  `sort` *filename* | `pr | lpr`

  `% sort states1 | pr | lpr`

Using Filters

Sort the contents of *states1.new* in reverse order and send the output to the line printer. (If you know the command for pagination, use it.) Your entry:

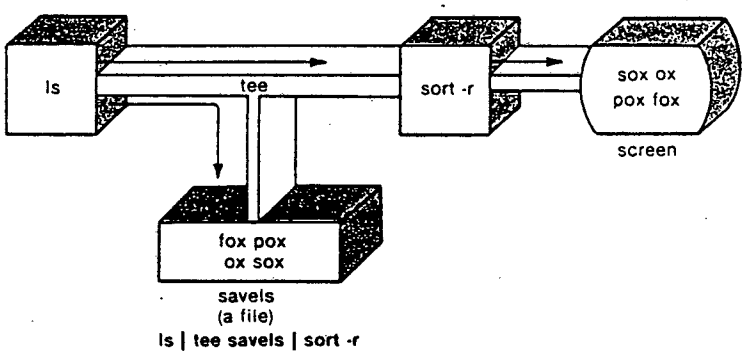
```
%sort -r states1.new | print
%
```

REDIRECTING I/O (cont.)

The *tee* command:

- Provides a display of command results
- Sends output to *stdout* and one or more files
- To send a listing of your current directory to a file and a reverse sorted list of files to *stdout*:

```
ls |tee savels | sort -r
```



- To send a sorted list of users to a file and to the *stdout* in scroll mode:

```
who | sort | tee filename | more
% who | sort | tee listing | more
```

Using the *tee* Command

Determine the current system users; save that information in a file named *listing* and also display the results a screenful at a time on your terminal. Your entry:

```
%who | sort | tee listing | more
user 1  tty01  Sep 24 15:30
user 2  tty06  Sep 24 08:35
user 3  tty08  Sep 24 15:13
user 4  tty09  Sep 24 14:51
user 5  tty10  Sep 24 13:01
user 6  tty11  Sep 24 15:00
user 7  tty12  Sep 24 13:33
%more listing
user 1  tty01  Sep 24 15:30
user 2  tty06  Sep 24 08:35
user 3  tty08  Sep 24 15:13
user 4  tty09  Sep 24 14:51
user 5  tty10  Sep 24 13:01
user 6  tty11  Sep 24 15:00
user 7  tty12  Sep 24 13:33
%
```

Exercises for Chapter 8

1. How many files are there in the */dev* directory that begin with the characters *mt*? What command did you use to obtain this information?

2. You want to append information from file1 to file2. How would you accomplish this task?

3. Explain what would happen if you entered the following command:
   who | sort -r | lpr

4. Determine the total number of files (including dot files) in your home directory. This command should give the count no matter where you are at in the directory structure. What command did you use?

5. Sort the *states1* file or a file of your choice and redirect that output to a file named *srt.states*. What command did you use?

6. Save the output of the *cal 1986* command in a file named *curcal*. What command did you use?

7. Append the output of the *date* command to the *curcal* file. What command did you use?

8. Direct the output of *cal 8 1986* to the screen and to the file named *curcal*. What command did you use? What happened to the original contents of *curcal*?

9. What happens to the contents of *smfile* if you enter the command **cat smfile bigfile > smfile**?

10. Write a command that places a copy of all your files (file1, file2, file3, and file4) into a file named *bigfile*.

11. What does the command **cat samfile** do?

12. Write a command that determines how many users are on the system and sends a sorted list of users to a file named *userlist*.

command that determines how many users are on the system and sends a st of users to a file named *userlist* and to the terminal. If you can't get this tion using one command, write two two commands (on one line) to accomplish .

command that places the contents of all the files of your current working y that end with *t* into a file named *tfiles*.

command that sorts the contents of *file1* in numerical order and places the output in a file named *file1.num*.

9

Customizing the ConvexOS Environment

Objectives for Chapter 9

After completing this chapter, you will be able to:

1. Add an entry that initializes your terminal (if appropriate) to your *.login* file.
2. Determine what variables are currently in effect in the C shell.
3. Determine environment variables in effect and modify as needed.
4. Adjust the ConvexOS environment to fit your needs.
5. Add editing options to the *.exrc* file.
6. Modify or create *.cshrc*, *.exrc*, *.login*, *.logout*, and *.mailrc* files.
7. Create/modify *.plan* and *.project* files.
8. Execute specified commands at a specific time.

THE *.cshrc* FILE

The *.cshrc* file:

- Gets its name from *csh*, the program that uses it
- Is read at the time you start the C shell, including subshells
- Provides initial instructions to C shell when you log in or start running the C shell
- Provides information to set up a new shell
- Certain variables have special meaning to the *shell* and are always set by the *shell*:
  - argv
  - home
  - path
  - prompt
  - shell
  - status
- To display the *shell* variables in effect and their values, use the command:

```
set
```

Initialized *shell* variables

Display a listing of the *shell* variables that are initialized when you log in. Your entry looks similar to:

```
% set
argv        ()
autologout  30
cdpath      /doc/username
cwd         /doc/username/training/newunix/mod4
history     20
home        /doc/username
mail        /usr/spool/mail/username
notify
old         /doc/username
path        (. /doc/username/bin /usr/convex /usr/ucb /bin /usr/bin)
prompt      %
shell       /bin/csh
status      0
term        vt100n
user        username
```

SETTING *shell* VARIABLES

- To set/modify a variable for a terminal session, use:

  `set` *variable*

  or

  `set` *variable* = *value*

  `% set ignoreeof`

  `% set prompt = "What?"`

- To set a variable permanently, add the line to your *.cshrc* file

- To turn off a variable-controlled option, use:

  `unset` *variable*

  `% unset ignoreeof`

Setting *shell* variables

You want your prompt to be personal; change it to **What's next, *your_name*?**. Your entry:

```
% set prompt="What's next, Floppsy?"
What's next, Floppsy?
```

Reset your prompt to %. Your entry:

```
What's next, Floppsy?set prompt=%
%
```

CUSTOMIZING THE *.cshrc* FILE

- Examples of variables you can set:

`set autologout=`*number*	Specifies number of minutes an interactive *shell* can be idle before the *shell* automatically terminates itself
`set ignoreeof`	Prevents accidental logouts when you type CTRL-d
`set noclobber`	Prevents unintentional overwriting of files with redirection (>)
`set history=`*number*	Records last *number* of commands entered
`set savehist=`*number*	Saves the last *number* commands and uses them as starting history list at next login session
`set prompt=`*string*	Sets an alternate prompt

- Aliases can be added to your *.cshrc* file:

  `alias` *newname command*

- To invoke the new variables and/or aliases, type: `source .cshrc` after entering the variable in your *.cshrc* file

Modifying Variables in *.cshrc*

Change your history setting to save 25 events. Your entry (using *ex*):

```
% ex .cshrc
".cshrc" 3 lines, 45 characters
:/history/
set history = 20
:s/20/25
set history = 25
:wq
".cshrc" 3 lines, 45 characters
% source .cshrc
%
```

Change your prompt permanently to **What do you want to do,** *your_name*? Your entry (using the *ex*) editor:

```
% ex .cshrc
".cshrc" 3 lines, 45 characters"
:a
set prompt="What do you want to do, Sally?"
.
:wq
".cshrc" 4 lines, 90 characters
% source .cshrc
What do you want to do, Sally?
```

Add an alias to your *.cshrc* file; alias *history* to *h*. Your entry (using the *ex* editor):

```
% ex .cshrc
".cshrc" 4 lines, 90 characters
:a
alias h history
.
:wq
".cshrc" 5 lines, 104 characters
What do you want to do, Sally?source .cshrc
What do you want to do, Sally?
```

CUSTOMIZING THE *.login* FILE

- Commands that you might want to add/modify in your *.login* file:

`msgs -q`	If there are new messages, displays "There are new messages" at login time
`msgs -p`	Pipes message output through *more*
`stty erase ^e`	Sets the command sequence for erase to control e
`tset -Q`	Initializes the terminal settings silently

Customizing the *.login* File

Add the command line to your *.login* file that pipes message output through *more*. Your entry (using the *ex* editor):

```
% ex .login
"login" 7 lines, 168 characters
:a
msgs -p
.
:wq
".login" 8 lines, 175 characters
%
```

ENVIRONMENT VARIABLES

Environment variables:

- Initialized by login
- Maintained by the shell
- Passed to all commands and programs run from within the current *shell*
- Store information that certain programs need to know about
- Usually in uppercase (to distinguish them from ordinary *shell* variables):

PATH	HOME
TERM	TERMCAP
SHELL	USER
PRINTER	EDITOR
PAGER	LESS

- To display a list of environment variables and their values, type: `printenv`

Environment Variables

Display a listing of the environment variables. Your entry:

```
% printenv
HOME=/mnt/username
SHELL=/bin/csh
PATH=.:/mnt/username/bin:/usr/convex:/usr/ucb:/bin:/usr/bin
TERM=vt100n
USER=username
TERMCAP=d0|vt100n:cr=^M:do=^J:nl=^J:bl=^G: .....
%
```

ENVIRONMENT VARIABLES (cont.)

- To set an environment variable for a terminal session, use:

  `setenv` *ENV_VARIABLE_NAME value*

  `% setenv PRINTER swip`

- To permanently set an environment variable, place the instruction in your *.login* file

- To remove the definition of an environment variable for a terminal session, use:

  `unsetenv` *ENV_VARIABLE_NAME*

Setting Environment Variables

Set your editor for this terminal session to emacs; you will need to use the pathname: */usr/convex/emacs* to specify the editor. Your entry:

```
% setenv EDITOR /usr/convex/emacs
%
```

Verify that the editor is emacs by displaying the environment variables.

```
% printenv
HOME=/mnt/username
SHELL=/bin/csh
PATH=.:/mnt/username/bin:/usr/convex:/usr/ucb:/bin:/usr/bin
EDITOR=/usr/convex/emacs
TERM=vt100n
USER=username
TERMCAP=d0|vt100n:cr=^M:do=^J:nl=^J:bl=^G: .....
%
```

CUSTOMIZING THE *.mailrc* FILE

- When you run *mail*, the program looks for initial options in *.mailrc*
- To set *mail* options for a terminal session, type: `set` *option* at the mail prompt
- To permanently setup an option, add the line to your *.mailrc* file
- Example of available options:

`set autoprint`	Prints the next message automatically after a delete command
`set hold`	Keeps current messages in system mailbox until removed
`set crt=`*n*	Determines how long a message must be before *more* is used to read it
`set editor=`*ed_pthnm*	Specifies name of default editor to use when editing mail messages
`set record=`*pathname*	Specifies where to record all outgoing mail; otherwise, outgoing mail is not saved

- See *mail*(1) in the *CONVEX UNIX Programmer's Manual* for a complete listing of options

Customizing the *.mailrc* File

Add a line to your *.mailrc* that forces the next message to print automatically in mail after a delete command. Your entry (using the *ex* editor):

```
% ex .mailrc
".mailrc" 1 line, 21 characters
:a
set autoprint
.
:wq
".mailrc" 2 lines, 32 characters
%
```

If you have mail that you can delete, verify that you did set *autoprint*.

CUSTOMIZING THE *.logout* FILE

The *logout* file:

- Is read when you exit from the *login* shell
- Provides instructions for the C *shell* to perform when you log out
- Is best suited for commands displaying information about the session just ending
- Can contain commands that run in the background (after logout)
- Examples:

`clear`	Clears the terminal screen
`echo Goodbye`	Displays "goodbye"

Customizing the *.logout* File

Add the lines to your *.logout* file that cause "so long" to display when you logout. Your entry (using the *ex*) editor:

```
% ex .logout
".logout" 1 line, 5 characters
:a
echo "so long"
.
:wq
".logout" 2 lines, 13 characters
%
```

Verify that this option is set by logging off the system.

CUSTOMIZING THE *.exrc* FILE

Whenever you run *vi*, the editor looks in the *.exrc* file for initial commands and option settings.

- To determine which options are set (while in *vi*), type:

```
:set
```

A listing is displayed at the bottom of the screen.

- To display a list of all options available (while in *vi*), use:

```
:set all
```

- A complete listing of options and their descriptions can be found in the *CONVEX Tutorial Papers, Ex Reference Manual.*

Modifying the *.exrc* File

Do the following exercises if *vi* is your favorite text editor. To complete this exercise, you will need at least one file to edit: *states1*. Determine which *vi* options are set and display a list of the *vi* options available. Your entry (using the *vi* editor):

```
% vi states1
Maine
Montana
Nebraska
Illinois
Iowa
~
~
"states1" 5 lines, 37 characters
:set
autoindent redraw shell=/bin/csh/term=vt100n nowarn
:set all
autoindent     nonumber      noshowmatch
autoprint      open          noslowopen
noautowrite    nooptimize    tabstop
. . .
. . .
nomodelines    noshowinsert  nowriteany
:q!
%
```

CUSTOMIZING THE *.exrc* FILE (cont.)

- Options can be added to the *.exrc* file or set while in *vi*

- Within *vi*, to specify an option, include the option as an argument to the *:set* command:

  `:set` *option*

  `:set autoindent`

- For options having a value, use:

  `:set` *option=value*

  `:set window=30`

- To change a setting, use *set* to set a new value:

  `:set wrapmargin=0`

- To change your terminal type for an edit session while in *vi*, type:

  `Q`
  `:set term=vt100`
  `:vi`

Setting *vi* Options

To complete this exercise, *vi* should be your favorite text editor. While editing the file *states1* (using *vi*), set the *number* option to display a line number preceding each line of text. Your entry (using *vi*):

```
% vi states1
Maine
Montana
Nebraska
Illinois
Iowa
~
~
"states1" 5 lines, 37 characters
:set number
1 Maine
2 Montana
3 Nebraska
4 Illinois
5 Iowa
:q!
%
```

CREATING/MODIFYING THE *.project* FILE

- To create or change your "Project" information for the *finger* utility:

  - Use a text editor to create a *.project* file (if it does not exist) in your home directory

  - Enter the message that you want to display as your "Project"

  - Save the file

  - View your user information by typing:

```
f your_username
```

Creating a *.project* File

Create a *.project* file in your home directory with a line or two describing or naming your position. Your entry (using the *ex* editor):

```
% ex .project
".project" [new file]
:a
Information about yourself goes in here.
Be creative.
.
:wq
".project" [new file] 2 lines, 52 characters
%
```

Now view the information about yourself, using the *finger* utility. Your entry:

```
% f username
Login name: person                  In real life: First Person
Office: 86/485, x489                 Home phone: 9520200
Directory: /mnt/user                  Shell: /bin/csh
On since Dec 11 08:42:13 on ttyOf
Project: Never ending ...
%
```

CREATING/MODIFYING THE *.plan* FILE

- To create or change your "Plan" information in the *finger* utility:
  - Use a text editor to create a *.plan* file (if it does not exist) in your home directory
  - Enter the message that you want to display as your "Plan"
  - Save the file
  - View your user information by typing:

```
f your_username
```

Creating the *.plan* File

Create a *.plan* file with a line or two describing one of your goals for the future. Your entry (using the *ex* editor):

```
% ex .project
".plan" [new file]
:a
Information about your goal goes in here.
Be creative.
.
:wq
".plan" [new file] 2 lines, 52 characters
%
```

Verify your entry by viewing the information about yourself. Your entry:

```
% f username
Login name: user                    In real life: First Person
Office: 86/485, x489                  Home phone: 931-3159
Directory: /tac/user                   Shell: /bin/csh
On since Dec 11 08:42:13 on tty0f
Project: Never ending ...
Plan: I would rather be skiing.
```

CREATING THE *.crontab* FILE

- The *cron* daemon executes commands at specified dates and times according to instructions found in *~/.crontab*
- *cron* checks *~/.crontab* files for modification once an hour; use tellcron to force *cron* to check immediately
- Use a text editor to create a *.crontab* file in your home directory
- *crontab* entries have the following format:

  minute hour day_of_month month_of_year day_of_week command

  The first 5 fields are integer patterns separated with spaces or tabs:

  minute (0-59)
  hour (0-23)
  day of the month (1-31)
  month of the year (1-12)
  day of the week (1-7; 1 = Monday)

  Asterisks may be used for any pattern of the 5 fields to represent all legal values
- The *.crontab* entry:

```
30 9 * * 2 mail friends < /mnt/username/testcase
```

  sends a mail message every Tuesday at 9:30 a.m. to the mail alias *friends* using the file *testcase* as the input

Creating a *.crontab* File

Assume that you have a file named *status* in the directory */mnt/yourusername/bin* containing a report that needs to be sent to user *jones* every Friday afternoon at 5:00. Create an entry in your *.crontab* file that will accomplish this task. Your entry (using the *ex* editor):

```
% ex .crontab
".crontab" [new file]
:a
00 17 * * 5 mail jones < /mnt/username/bin/status
.
:wq
".crontab" [new file] 1 line, 49 characters
%
```

Exercises for Chapter 9

1. Write a command that will automatically log you off the system if there is 30 minutes of inactivity. In what file would you put this command?

2. You keep accidentally logging off the system when you type CTRL-d. Write a command that will keep this from happening. In what file would you put this command?

3. Edit the appropriate file to cause the last 30 commands you have executed to be saved for your future execution. What command did you enter and in what file did you put this command?

4. Set the default printer to *trip* for this terminal session. What command did you use?

5. Write a command that displays the message: `It is about time you got to work` when you log on.

6. Edit the appropriate file to cause mail messages to remain in your system mailbox rather than being saved to *mbox*. What file did you edit and what command did you enter?

7. You want your terminal to display the line: `See you tomorrow` when you log off the system. Edit the appropriate file. Verify that it works. What file did you edit and what command did you enter?

8. No information displays for *project* or *plan* when you use the *finger* utility. Add a project and plan of your choice to the appropriate files. Verify. How did you accomplish this task?

9. Why would you want to set *ignoreeof* in your *.login* file?

10. Add a command to the *.cshrc* that protects your files from accidentally being overwritten by redirection commands. Verify that it works. What is the command you entered?

11. Write a command that you can use within *vi* that allows you to change the terminal.

12. Write a command that sets *autoindent* in *vi*.

13. When are the *.login* and *.cshrc* files read?

14. In what file can you change/set the default permissions for new files?

15. Write a command that sets your default printer to *tacip* for all terminal sessions.

16. Display a listing of your environment variables. What command did you use?

17. Write a command that allows you to save five history events across terminal sessions. To what file would you add this information?

18. Explain how to change information in the *finger* utility for *project* or *plan*.

19. In what file would you put a mail alias, such as docgrp for users snoopy, sally, and fred. Give the format of the command.

20. In what file would you place a command to remove all the files beginning with .## in your home directory the first day of every month. Write the command that you would use to accomplish this task.

10
Notesfile

Objectives for Section 10

After completing this chapter, you will be able to:

1. Display a listing of notesfile topics.
2. Display *help* for the notesfile.
3. Scroll through a specific notesfile index.
4. Create notes using the *notes* system.
5. Utilize the commands for reading a note/response in the *notes* system.
6. Respond to an existing note.

notes - A NEWS SYSTEM

notes(1)

- Functions as an electronic bulletin board
- Supports computer-managed discussion forums
- Coordinates access to various data bases of notes and their responses
- Updates the data bases of notes and their responses
- A single *notesfile* contains an ordered list of *base notes*, each of which may have an ordered list of responses associated with it
- A *note string* consists of a *base note* and all its responses
- Separate *notesfiles* contain discussion on separate topics

notes - A News System

The *notes* system provides users with the abilities to read notes and responses, to write their own files, and to sequence through a set of notesfiles, viewing all text or new text only.

Almost all notesfile commands (within notes) require exactly one character and no carriage return. The commands were chosen to be easy to remember, such as *i* for index, *j* for jump, *s* for save, etc.

notes TOPIC LIST

- Format of *notes* command:

  `notes [`*options*`] [`*topic1 . . .*`]`

- To display a list of available topics, use:

  `% notes`

  A message is displayed; press the RETURN key to view a list of topics.

- To select a specific topic, use:

  `% notes` *topic_from_list*

  For example, to access the topic *general*:

  `% notes general`

- To use a list of topics contained in a file:

  `% notes -f` *filename*

- To use a topic list placed in the environment variable *NFSEQ*:

  `% setenv NFSEQ "`*topic,topic*`"`

  For example:

  ```
  % setenv NFSEQ "general,practice"
  % notes -s $NFSEQ
  ```

  The *notesfiles* in the topic list are scanned from left to right; upon finishing the first topic, the second is entered

Getting Started with *notes*

NOTE: The machine you are currently on may have a different list of topics than what you see here. If you find a topic named *practice* or *students*, please use either of those *notesfiles* for your practice work.

Request a list of the topics in the *notes* system and pipe that list through *more*. Your entry:

```
% notes | more
Usage: notes [-s] [-t ttytype] [-f file] [-a seqname] topic [...]
Hit <return> to continue
comp.sys.amiga
general
mod.amiga
net.micro.amiga
net.micro.apple
net.rec.photo
practice
students
%
```

ACCESSING THE NOTESFILE

- The shell's metacharacters **?[]* are recognized within a topic

  % notes "*topic*.*"

- *notesfiles* may be excluded from the topic list by prefixing their names with a *\!*

  % notes "*", "\!*topic*"

- To show the index of new notes, responses, and text since your last entry into that *notesfile*:

  % notes -i *topic*

- Users can add notes or reply to notes; set either environmental variable, *EDITOR* or *NFED* to establish a default editor.

  For example, to set these to *emacs*, you can add either of the following lines to your *.login* file:

```
setenv EDITOR /usr/convex/emacs
setenv NFED /usr/convex/emacs
```

- You can exit *notes* at any time with *q* or *Q*. However, if you are reading a series of *notes*, using *q* causes the next *topic* index to display; exit at anytime with *^D*.

Displaying the *notes* Index

Read the *practice* index; use the *q* command to exit *notes*. If you have never read the *practice notesfile* before, you will be placed at the end of the index page for that topic. A sample entry:

```
% notes -i practice

practice                          11:15 am Dec 11, 1989


5/23/89
   6 Blank common lengths                    user@convext
   7 no repeated Holleriths in DATA          user@convext
   8 adjustable array <-> ENTRY statement        user@convext
   9 dusty decks and -q AGAIN (sigh)         user@convext
  10 BIG BASIC BLOCKS                     user@convext
  11 VMS/FORTRAN LOGICAL IFs                user@convext
  12 Disappearing loops                    user@convext
  13 INDUCTION VARIABLES in EXPRESSIONS     user@convext
  14 isolating bugs to loops               user@convext
  15 breaking up big subroutines            user@convex.UUCP
  16 Blank common lengths, again            user@convex.UUCP
  17 TWS directives                        user@convext
    **** End of Notes ****
         ----------------
q
%
```

Since you just read the *practice notesfile* and no new entries have been added, you will not see any entries when you request the *practice* topic again. Your entry.

```
% notes -i practice
practice...
%
```

An index for any topic may be viewed by entering `notes` followed by the topic name. For example:

```
% notes general

practice                          11:15 am Dec 11, 1989


5/23/89
6 Car Problems   username
7 Volleyball     username
8 Dr. needed     username
******End of Notes****
q
%
```

MOVING AROUND IN *notes*

- When *notes* is invoked, an index of topics is displayed. Enter the number of the note and press the RETURN key.

- *help* can be accessed from within *notes* by entering a ?

- The following commands allow movement within the *notesfile* system:

space	Show the next page of the note, response, or index page
return	Show the next note or next index page
number	When in an index page, read the *number*th note listed When reading a note, skip *number* responses
+	Go forward one page of the current note, response, or index
-	Go to the previous page of the current note, response, or index
;	Go to the next response; if there are no more responses, go to the next note
*	Skip to the last response

notes Commands

Enter the *practice notesfiles* or the *general notesfile.* Enter a – to see the previous page of the index; then enter a + to return to the last page of the index. A sample entry:

```
% notes practice

practice                          11:45 am Dec 11, 1989

5/23/89
   6 Vectorize Character Strings          user@convex
   . . .
  17 Recursive Subroutines               user@convex
    **** End of Notes ****
         - - - - - - - - - - - - - - -
-

practice                          11:45 am Dec 11, 1989

8/14/89
   1 Multiple Printers On One Queue       user@convex
   . . .
   5 Other Laser Printer                 user@convex

         - - - - - - - - - - - - - - -
+

practice                          11:45 am Dec 11, 1989

5/23/89
   6 Vectorize Character Strings          user@convex
   . . .
  17 Recursive Subroutines               user@convex
    **** End of Notes ****
         - - - - - - - - - - - - - - -
```

Read one of the notes listed in the index; pick any one you wish. Your entry:

```
5
Read note > 5
/* Written 10:07 pm Nov 4, 1989 by user@convext.UUCP in convext:gen */
/* ---------- "testing of notes" ---------- */
Are we having fun yet ?
/* End of text from convext:gen */
```

notes COMMANDS (cont.)

i	Return to the index from a note/response page
j	Jump to a new note or response
J	Skip the rest of this note; jump to the next note/response
k	Leave the *notesfile* and update the sequencer information
m	Mail a message to someone
M	Mail a message to someone; include the text of the note
p	Write a personal note to the author
P	Write a personal note to the author; include the text of the note
q	Leave the *notesfile* and update the sequencer information
Q	Leave the *notesfile* without updating the sequencer information
s	Save the currently displayed note/response at the end of a (prompted for) file

Saving *notes*

Save a copy of the note your are reading with the *s* command; name the new file *junk1*. Your entry:

```
s
File name: junk1
        Saved 8 lines in junk1
```

Return to the index by means of the *i* command. Your entry:

```
%i

practice                          11:45 am Dec 11, 1989

5/23/89
   6 Vectorize Character Strings          user@convex
   . . .
  17 Recursive Subroutines               user@convex
    **** End of Notes ****
         ----------------
```

notes COMMANDS (cont.)

S Save the currently displayed note string at the end of a (prompted for) file

w When issued from the index page, allows user to write a *new* note.

When issued from a note/response display, enters a response to the current *note*.

The default editor is *vi*; to select a different editor, set the *NFED* or *EDITOR* environment variable to the desired editor.

W Includes the text of the currently displayed note/response in the new response which is written by means of the selected editor

x Search for a note with the (prompted for) string in its title

X Search for a note with the previous search string in its title

notes Commands

Read another note and write a response to it. Select any note you wish.

```
5
Read note > 5
/* Written 10:07 pm Nov 4, 1989 by user@convext.UUCP in convext:gen */
/* ---------- "testing of notes" ---------- */
Are we having fun yet ?
/* End of text from convext:gen */
w
Edit Response Text:
Yes we are!
Director Message (y/n): n
```

View the index pages of all the *notesfiles* having something to do with *amiga*. Again, use **q** to exit *notes*. Your entry:

```
% notes -x "*.amiga"

comp.sys.amiga                    11:30 am Dec 11, 1989

5/23/89
   6 Beginning O.S.                user@convex
   . . .
  17 New Problems                 user@convex
    **** End of Notes ****
         - - - - - - - - - - - - - - - -
q

mod.amiga                        11:35 am Dec 11, 1989

8/14/89
   6 Modifications                 user@convex
   . . .
  17 Other Laser Printers             user@convex
    **** End of Notes ****
         - - - - - - - - - - - - - - - -
q

net.micro.amiga                    11:40 am Dec 11, 1989

12/10/89
   1 Hello World                   user@convex
    **** End of Notes ****
         - - - - - - - - - - - - - - - -
q
%
```

Exercises for Chapter 10

1. What is a *note string* composed of?

2. What is the purpose of the *-i* option on the *notes* command line?

3. How do you set the default *notes* editor?

4. Can you use shell metacharacters on the *notes* command line? If so, how?

5. How do you exit a *notes* session?

6. What does the *x* command in the *notes* system do?

7. Write a command that allows you to write a personal note to the author of the "note" you are reading.

8. Write a command that saves the current note to the file *note.file*.

9. Write a command that allows you to make a reponse to the current "note".

10. Submit a practice note on a topic of your choice.

11. Make a reply to an existing note.

A

VI Commands

Command	Description
h	Back 1 character
(	Back 1 sentence
B/b	Back 1 word
s	Change current letter then insert
C	Change entire line (cc)
r	Change current letter Only
~	Change case one character
cb/cw	Change word backward/forward
Q	Change to ex editor
:vi	Change back to vi
ctx	Change through letter "x"
R	Change multiple characters
:'a,'bt.	Copy Block after current line
:'a,'bd	Delete block a - b
dd	Delete line
"xndd	Delete n lines into buffer x (a-z)
d(/ d)	Delete to beginning/end of sentence
D	Delete to end of line
db/dw	Delete word backward/forward
X/x	Delete character before/under cursor
:set all	Display all options
z-	Display current line at bottom
z.	Display current line at center
z	Display current line at top
:set	Display current options
^G	Display file information
(cr)	Down 1 line - first character
j	Down 1 line - same column
YP/Yp	Duplicate current line before/after
:e <file>	Edit different file
Fx/fx	Find previous/next "x" same line (; and , = repeat)
l	Forward 1 character
W/w.	Forward 1 word
)	Forward 1 sentence
L	Go to bottom of screen

Command	Description
n|	Go to character n (same line)
G	Go to end of file
$	Go to end of line
0	Go to first character of line
M	Go to middle of screen
nG	Go to line n (1G = Line 1)
H	Go to top of screen
A/a	Insert after EOL/cursor
I/i	Insert before line/cursor
O/o	Insert before/after current line
DEL	Interrupt (Stop)
:set nu	Line number enable
:set nonu	Line number disable
mx	Mark Position (x= a-z)
:'a,'bm.	Move Block (a thru b)
^B/^U	Full/half page up
^F/^D	Full/half Page Down
:q	Quit editor no mods made
:q!	Quit editor abandon changes
:r <file>	Read file (insert)
^l	Refresh screen
.	Repeat last command
P/p	Restore deleted text Before/After
"np	Restore deleted buffer n (1-9)
"xP/xp	Restore from buffer x (a-z)
`x	Return to mark x (a-z)
``	Return to previous position
ZZ	Save File & Quit editor
^Y/^E	Scroll down/up one line
?^<strng>	Search backward for <string> beginning of line only
?<strng>$	Search backward for <string> end of line only
?<string>	Search file backward

Command	Description
/<string>	Search file forward
/^<strng>	Search forward for <string> beginning of line only
/<strng>$	Search forward for <string> end of line only
N/n	Search repeat opposite/same direction
---->	:1,$s/original/changed to/g Global search & replace
!}fmt	format
:n	goto next file<L one shiftwidth lef

Command	Description
>L	Shift remaining lines on screen one shiftwidth right
:set sw=n	Set shiftwidth n spaces
!}sort	Sort until next blank line
U	Undo all changes to one line
u	Undo last command
k	Up 1 line - same column
:'a,'bw!	Write block (a-b) to existing file
:'a,'bw	Write block (a-b) to new file
:w <file>	Write file
Yp/YP	Yank buffer
"xny	Yank into buffer x, n lines

Entering/Leaving *vi*

The following table illustrates the basic ways of entering and exiting *vi*:

Table A-1: Entering/Leaving *vi* Commands

Command	Explanation
vi *filename*	edit *filename*
vi +*n filename*	When the text appears on the screen, the cursor is positioned on the *n* line.
:wq	Writes buffer contents to permanent storage; returns you to the *shell*
:q!	Exits the editor and aborts all modifications to the buffer (file)
:w *filename*	Writes buffer contents to permanent storage to new *filename* without exiting to *shell*; will not write to an existing file
:w! *filename*	Writes buffer contents to an existing *filename*; overwrites contents of *filename*

Command Modes in *vi*

There are two modes in *vi*:

Table A-2: *vi* Command Modes

Mode	Explanation
command	Normal and initial startup state; can enter commands but no text; to return to command mode press the ESC key
text input	Entered by typing a command (**a**/A—append, **i**/I—insert, **o**/O—open line, **c**/C—change, **s**/S—overtyping, **R**—replaces) followed by the text; must use ESC to return to command mode

Specifying a Terminal for *vi*

Use the command **set term**=*terminal* in your *.exrc* file. You can also set the environmental variable *TERM* to your *terminal* type, i.e., **setenv term** *terminal*.

B

ConvexOS Command Summary

The following table lists commonly used ConvexOS commands by category of use.

Table B-1: Common ConvexOS Commands

Category	Commands	Description
Batch	`qstat`	list jobs in queue
	`qdel` *jobid*	remove *job* from batch queue
	`qsub` *f1*	submit executable file to batch queue
Communication	`biff`	determine type of mail notification
	`mail`	send/receive electronic mail
	`mesg`	permit or deny messages from others
	`talk`	start two-way terminal communication
Directory	`cd` *d1*	change directory to *d1*
	`mkdir` *d1*	make new directory named *d1*
	`pwd`	print working (current) directory
	`rmdir` *d1*	remove empty directory *d1*
Editing	`awk` *commands*	string processing language
	`emacs` *f1*	*emacs* full-screen text editor
	`ex` *f1*	*ex* line editor
	`sed` *commands*	batch line editor
	`vi` *f1*	*vi* full-screen text editor
File	`cat` *f1 f2*	list both files *f1* and *f2* to standard out
	`chmod` *opt f1*	change permission bits for file *f1*
	`cp` *f1 f2*	copy file *f1* to *f2*
	`find` *options*	find and act on designated files
	`head` *f1*	show top portion of file *f1*
	`less` *f1*	show page with forward and backward scrolling
	`ln` [`-s`] *f1 f2*	link file *f2* to existing file *f1*
	`ls` [`-alF`]	list files in directory
	`more` *f1*	show page with forward scrolling
	`mv` *f1 f2*	move file *f1* to *f2*
	`rm` [`-r`] *f1 f2*	remove files *f1* and *f2* (-r recursively)
	`tail` *f1*	show tail portion of file *f1*
	`tee`	split output into two files

Table B-1: Common ConvexOS Commands (cont.)

Category	Commands	Description
Filter	`cmp` *f1 f2* `dd if=`*f1 of=f2* `diff` *f1 f2* `grep` *string f1* `sort` *f1* `wc` *f1*	display first difference in files *f1* and *f2* copy data from *f1* to *f2*, converting data show differences between files *f1* and *f2* search for string in text file sort file *f1* count the words/lines in file *f1*<

Help	`info`
	`learn`
	`man [-k]` *command*
		access information by topic or menu selection
		access tutorial
		display manual page on selected topic

Interactive	`fg`
	`bg`
	`jobs`
	`kill`
	`ps`
		bring background job into foreground
		put foreground job into background
		list jobs in background
		kill job in background mode
		list all processes running on your behalf

Language	`as` *f1.s* `cc` *f1.c* `fc` *f1.f*	assemble file containing assembly code compile/link C language files compile/link FORTRAN files

Miscellaneous	`alias` *al string*
	`cal` *mo yr*
	`date`
	`echo` *string*
	`finger` *user*
	`history`
	`stty` *options*
	`time` *command*
		alias *al* to command found in string
		display a calendar
		print the date
		echo *string* to standard out
		provide user information
		view history of commands you have entered
		set terminal options
		time the command

Printer	`lpr` *f1*
	`pr` *f1*
	`lpq`
	`lprm`
	`nroff -ms` *f1*
		send *f1* to line printer
		add headings, etc., to printed output
		display printer request queue
		remove request from printer queue
		formats text to be printed

Tape	`ansitar`
	`mt`
	`tar`
	`tpalloc`
	`tpdealloc`
	`tpmnt`
	`tpq`
	`tprm`
	`tpumnt`
		ANSI labeled tape utility
		magnetic tape utility
		tape archive utility
		allocate tape drive
		deallocate tape drive
		request tape mount
		display tape request queue
		remove request from tape queue
		request tape unmount

User	`login username`
	`logout`
	`passwd`
	`who`
	`whoami`
		login in to system
		logout of system
		change user password
		list users currently logged in
		display username

Index

D

E

X

Y

Z

ConvexOS Student Course Materials

Training Course Critique

You are invited to submit your comments concerning the clarity and quality of the training course. Constructive critical comments are most welcome and will help us continue in our efforts to provide quality training.

What part of the training course was most beneficial to you?

What suggestions do you have for improving the course content?

Do you have any comments regarding the instructor, course content, or class environment? Were questions adequately answered? Did the course content meet your expectations?

Instructor_____ Date_____

Company you represent_____