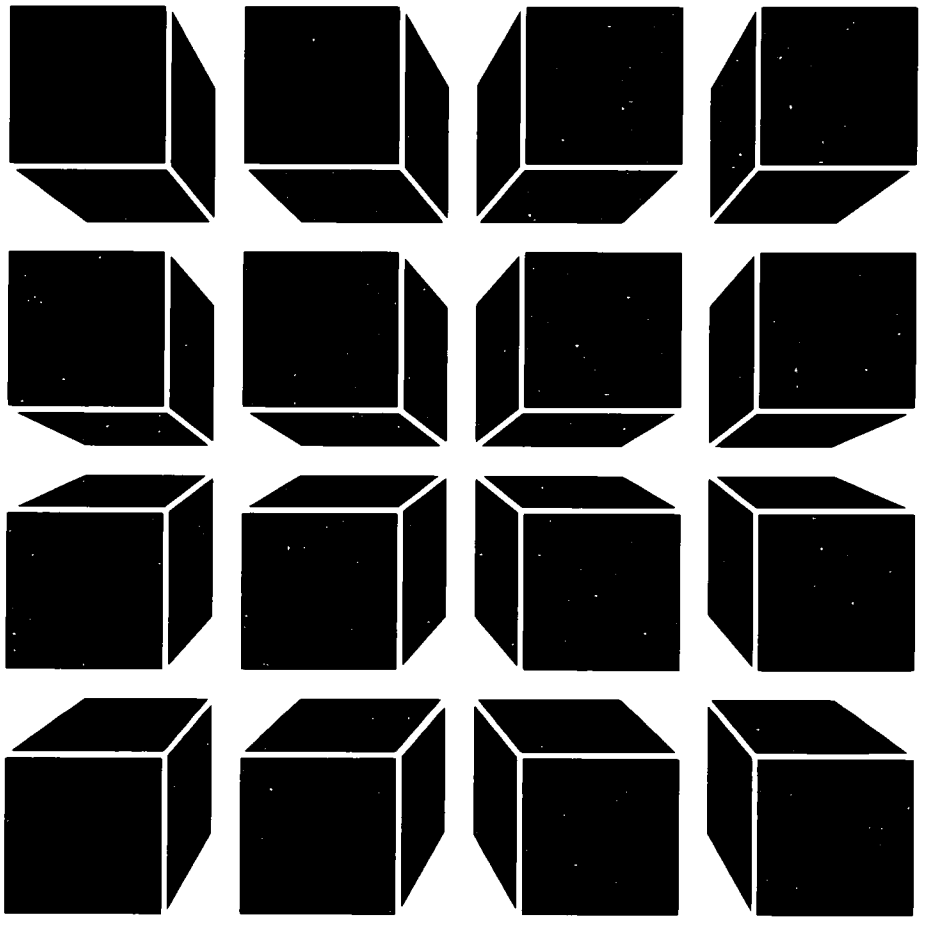
AT&T VIDEOTAPE LIBRARY

FUNDAMENTALS OF THE UNIX® SYSTEM - INTERMEDIATE LEVEL



WORKBOOK

AT&T

CONTENTS

VOLUME 3—File Access and Manipulation

VOLUME 4—The vi Screen Editor: Part II

AT&T COMPUTER SYSTEMS EDUCATION PROGRAM INTRODUCTION TO STUDENT WORKBOOK

To the Student,

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

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

This course, on **Fundamentals of the UNIX® System, Intermediate Level**, has been designed for users who already understand the basics of the UNIX System and want to learn more. You will be introduced to the shell command language, as well as learning more advanced techniques in communications and the vi screen editor.

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

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

VOLUME 1

The Interactive Shell and Shell Commands

WHAT YOU'LL BE ABLE TO DO AFTER VOLUME 1

- Display and change your terminal setup.
- Display and use standard shell variables.
- Enter more than one command on a line.
- Run shell procedures from any directory.
- Create a *.profile* to customize your UNIX System environment.
- Use a pipeline to pass output from one command to another command.

WHAT HAPPENS WHEN YOU LOG ON?

When you log on the UNIX System, the following automatically takes place:

— The **login** program:

1. Checks the login id and password you enter against the information in the file */etc/passwd* for validity. The file */etc/passwd* contains all the valid logins for your system, the **encrypted** password for each login, and other information.
2. Sets your HOME, MAIL, PATH and LOGNAME variables. These will be discussed shortly.
3. Starts up your shell.

— The **shell**:

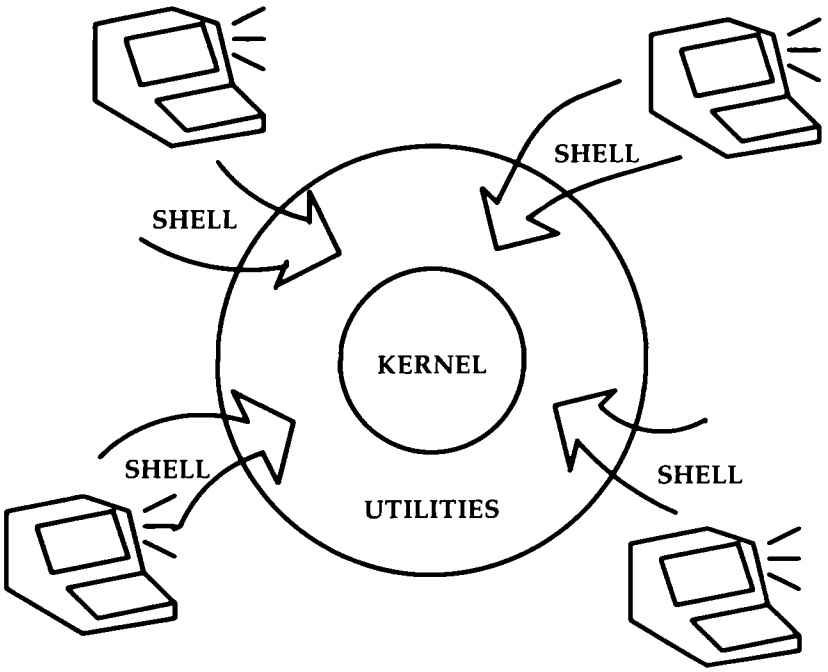
1. Executes the system setup file */etc/profile*, which:
   - — displays the message of the day
   - — notifies you of any news or mail
   - — sets other variables
   - — performs other setup (included by the system administrator)
   - — executes your own setup file called *.profile*, if such a file is in your home directory.

WHAT HAPPENS WHEN YOU LOG ON?

The System profile and your .profile:

Both the system profile (*/etc/profile*) and your *.profile* are shell programs. The most basic shell program is simply a list of commands in a file. The commands in the system profile are executed for everyone, every time they log on the UNIX System. The system administrator maintains the system profile for all users. On the other hand, your *.profile* is a way of customizing your own setup. It is executed after the system profile, meaning that you have the last say over your session setup. Your *.profile* contains commands and variable assignments that you want automatically executed every time you log on the UNIX System.

THE SHELL



- User interface
- Command line interpreter
- Programming language
- Establishes an environment

DISPLAYING YOUR TERMINAL SETUP

```
$ stty<CR>
speed 2400 baud; evenp hupcl
erase = ^h; swtch = ^';
brkint -inpck icrnl onlcr
echo echoe echok
$
```

- Default erase character is #
- Usually changed in */etc/profile* or *.profile*

DISPLAYING SHELL VARIABLES AND THEIR VALUES

A *variable* is a name that refers to a temporary storage area in computer memory

All Variables

```
$ set<CR>
HOME=/instr/jrs
MAIL=/usr/mail/jrs
LOGNAME=jrs
TERM=5425
PATH=:/bin:/usr/bin:
MAILCHECK=600
PS1=$
PS2=>
$
```

Environment Variables

```
$ env<CR>
HOME=/instr/jrs
MAIL=/usr/mail/jrs
LOGNAME=jrs
TERM=5425
PATH=:/bin:/usr/bin:
MAILCHECK=600
$
```

SOME STANDARD SHELL VARIABLES

PATH When you enter a command like **date**, **who**, etc., you are asking the shell to execute a program. The shell must first locate the file containing the program before it can execute it. How does the shell know where to look for this file? The PATH variable contains a list of directories where the shell should look for commands you enter. The directories in the list are separated by colons. The PATH variable's default value is **:/bin:/usr/bin**, which means:

```
Current directory (colon with no
directory name before it)
/bin
/usr/bin
```

The PATH variable on your system may list other directories that have been added by the system administrator.

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

PS1 stores the value of your primary shell prompt. The default is "$" (with a space after the dollar sign). Your prompt can be personalized by assigning PS1 a different value.

SOME STANDARD SHELL VARIABLES

PS2 The shell displays the secondary prompt, **PS2**, when an incomplete command line, such as one with an opening quote but no closing quote mark, is entered. If this prompt appears (its default value is ">"), either complete the command line or press <DELETE> to abort the command.

EXINIT is used by **vi**. It contains a list of **vi** options to be set every time **vi** is used. For example, this variable can be set with the line number option so that every time you use **vi**, line numbers are displayed.

LPDEST contains the name of your default printer.

MAIL The **mail** command reads a file containing all your incoming mail. How does the **mail** command know which file to read? The MAIL variable contains the path name of your mail file (*/usr/mail/login*).

MAILCHECK While logged on the UNIX System, you will periodically receive the message **you have mail** whenever new mail for you is available. The variable MAILCHECK contains, in seconds, how often your mail file should be checked for incoming mail. By default, it is 600 seconds (10 minutes). If, for example, MAILCHECK is reset to 0, your mail file will be checked every time the shell is ready to display the prompt.

A SAMPLE .profile

```
 1  date
 2  PS1='Speak to me: '
 3  TERM=5425
 4  EXINIT='set showmode nu'
 5  LPDEST= _____
 6  stty  erase  '^h' echoe
 7  echo  Reminders for today:
 8  calendar
 9  echo  The people on the system are:
10  who  -q
11  banner  get to work
12  export  TERM  EXINIT  LPDEST
```

To activate:

log off and then back on

OR

$ . .profile<CR>

ASSIGNING A SHELL VARIABLE A VALUE

variable=value

- Value must be a single word
- NO white space around equal sign

---

a sample **.profile**

variable →
assignments →
→

```
umask 077
echo Welcome!
PS1='Enter command: '
EXINIT='set showmode nu'
TERM=5425
stty erase '^h' echoe
banner get to work
export TERM EXINIT
```

$ **PS1='Enter command: '**<CR>
$ **TERM=5420**<CR>
$ **EXINIT='set showmode nu'**<CR>

VARIABLE SUBSTITUTION

Using the Contents of Variables

- Precede the variable name with a $
- The shell substitutes **$variable** with its contents

---

a sample **.profile**

```
→  echo Welcome $LOGNAME!
   PS1='Enter command: '
   EXINIT='set showmode nu'
   TERM=5425
→  echo your terminal type is $TERM
   stty erase '^h' echoe
→  banner get to work $LOGNAME
   export TERM EXINIT
```

$ echo $TERM<CR>
5425
$

THE READ COMMAND

read variable

- Assigns a value to the variable
- Value is read from *standard input*
- **read** does NOT prompt

---

a sample **.profile**

```
umask 077
echo Welcome $LOGNAME!
PS1='Enter command: '
EXINIT='set showmode nu'
→ echo enter your terminal type:
→ read TERM
echo your terminal type is $TERM
stty erase '^h' echoe
export TERM EXINIT
```

THE READ COMMAND

Another way to assign a value to a variable is with the **read** command. The **read** command is frequently used in shell procedures to allow information to be entered from the terminal and assigned to a variable. Since the **read** command does NOT prompt for input, it is frequently preceded by an **echo** command.

The next page shows an example of using the **read** command in a *.profile*. Below is an example of what happens when the user logs on:

```
login: jrs
Password:<CR>
Welcome to UNIX System V.   ← message of the day
Have a great day!
Welcome jrs!   ← welcome message from .profile
enter your terminal type:   ← prompt (echo) before read
5425<CR>   ← read waits here for input
your terminal type is 5425
        .
        .
        .
Enter command:
```

Above, the **read** command causes execution to stop and wait for input. After the user enters a terminal type, in this case **5425**, followed by a <CR>, execution continues. In the example above, the TERM variable is assigned the value **5425**. If just a <CR> is entered, TERM is assigned the *null* value.

CHOOSING NAMES FOR USER SHELL VARIABLES

- Should describe the contents
- Only upper or lower case letters, numbers, or underscores
- Cannot begin with a number

Valid variable names:	What's wrong with these?
directory	2ndfile
FILENAME	file-name
Item_3	file name
answer	acct#

USING VARIABLES — AN EXAMPLE

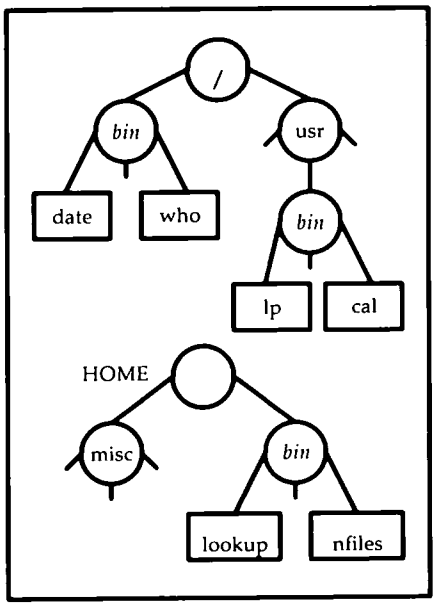
contents of **lookup** *file*

```
1  file=$HOME/info/friends
2  echo
3  echo  Welcome  $LOGNAME!
4  echo  You are running the lookup program.
5  echo
6  echo  Enter a name to look up:
7  read  name
8  echo
9  grep  -i  $name  $file
```

```
$ lookup<CR>

Welcome imr!
You are running the lookup program.

Enter a name to look up:
linda<CR>

Linda Moad  Sunnyvale,CA . . . June 20
linda Barry Hopewell,NJ  . . . July 29

$
```

EXECUTING COMMANDS — THE PATH VARIABLE

```
$ cd <cr>
$ nfiles <cr>
nfiles: not found
$ cd bin <cr>
$ nfiles <cr>
 the number of files:
      5
$ echo $ PATH <cr>
:/bin:/usr/bin:
```



- Contains a colon-separated list of directories
- Used by the shell to locate commands

I/O REDIRECTION SUMMARY

command > *file*	output of command
$ ls -p > msg	will be stored in *file*
command >> *file*	output of command
$ date >> msg	will be appended to *file*
command < *file*	input to command
$ mail liz < msg	will be contents of *file*

- Redirection allows storage of output or use of stored data as input
- Redirection symbol always appears between a command and a file

PIPES

THE BUILDING BLOCK APPROACH

command1 ¦ command2

$ cal ¦ mail carl<CR>

$ ls ¦ pg<CR>

- UNIX System commands are the blocks.
- Pipes are the connectors.
- Output of command1 becomes input to command2.
- Allows data to be processed by a sequence of commands.
- Eliminates the need for temporary files.
- Pipe symbol always appears between commands.

SEQUENTIAL COMMANDS

- Commands separated by a semicolon
- Executed in sequential order
- Time saver

```
$ cd; ls  -l<CR>
total 12
drwx------  2 jrs doc    208 Sep 18 10:32 budget
-rw-r--r--  1 jrs doc    173 Jul  7 14:01 calendar
-rw-r--r--  1 jrs doc    613 Jul 22 15:56 friends
drwxr-xr-x  2 jrs doc    112 Jul  7 15:20 fun
drwxr-x---  3 jrs doc    288 Sep 16 16:33 misc
drwx------  2 jrs doc    160 Jul  9 13:48 oldletters
-rw-r-----  1 jrs unixc 355 Jul  8 14:35 parts
-rw-------  1 jrs doc    641 Aug 13 08:28 poem
drwxr-x---  4 jrs doc     64 Aug 28 10:48 project
drwx------  2 jrs doc    240 Aug 29 09:46 work
$
```

VOLUME 1 EXERCISE

1. What are the two ways we assign a value to a variable?

2. How can we change a value of a shell variable and look at the new value?

3. What is the pipe symbol and how does it work?

4. What does the PATH shell variable do?

5. What does the dot profile do?

VOLUME 1 EXERCISE - ANSWERS

1. Page 1-11, 1-14
2. Page 1-11, 1-12
3. Page 1-19
4. Page 1-8
5. Page 1-4

VOLUME 1 SUMMARY

SETTING UP YOUR LOGIN SESSION

SESSION SET UP		
Command	Description	Example
stty	Displays terminal settings erase=^h : erase character backspace No "erase =' : erase character #	$ stty<CR> speed 2400 baud; evenp erase = ^h; swtch = ^',..
	Sets erase character to backspace Removes erased characters from screen	$ stty erase '^h' echoe
set	Displays all variables set in shell	$ set<CR> HOME=/usr/jdg ....
env	Displays environment variables	$ env<CR>
export	Variables must be exported to be used by programs	$ export TERM<CR>
. .profile	Executes .profile without logging off	$ . .profile<CR>
DISPLAYING INFORMATION		
Command	Description	Example
echo	Prints its arguments Useful in shell programs	$ echo welcome<CR> welcome
calendar	Reminder service - Searches the file *calendar* for today's or tomorrow's dates and prints reminder.	$ calendar<CR> 2/14 Valentine's Day

VOLUME 1 SUMMARY

SETTING UP YOUR LOGIN SESSION

STANDARD SHELL VARIABLES		
Variable	Description	Assignment
LOGNAME	login name	
HOME	path name of home directory	
TERM	terminal type	TERM=5425
PS1	primary prompt $	PS1='Speak to me: '
PS2	secondary prompt >	read TERM
PATH	directories for locating commands	
EXINIT	default options for vi	echo $TERM
LPDEST	default printer name used by lp	
MAIL	path name of mail file	
MAILCHECK	seconds between checking for arriving mail	

VOLUME 2

Advanced User Communications on the UNIX System

WHAT YOU'LL BE ABLE TO DO AFTER VOLUME 2

- Talk to another user on the UNIX System.
- Use the advance features of mailx.
- Send mail to users on other UNIX Systems.
- Send files to and receive files from other UNIX Systems.

TALKING TO ANOTHER USER

The **write** command allows interactive communication between two users logged into the same UNIX System, provided neither has denied "write permission" to their terminal screen. The **who** command (with the **-T** option) displays all the users currently on-line and each user's write permission to their terminal screen:

```
$ who  -T<CR>
mike       + tty40      Sep 9   12:54
jan        + tty42      Sep 9   12:58
imr        - tty12      Sep 9   13:04
$
```

A plus sign preceding the terminal number indicates write permission, while a minus sign indicates that write permission to that user's terminal is denied. To write to mike, enter:

```
$ write  mike<CR>
Is it LUNCH yet? <CR>
-o- <CR>
```

Once you have successfully initiated the write, you will not see the shell prompt (**$**), and each line you type will be printed on the other user's terminal as soon as you press <CR>. It is customary to indicate when you are through "talking" for the moment, so the other person can reply without interrupting you. A common convention, from walkie-talkie usage, is to type **-o-** for "over." If the recipient, in this case mike, wishes to reply, he can invoke the **write** command with your login id, and from that point on, whatever either of you types will be printed on the other's terminal.

When you are ready to end the conversation, you can type **-oo-** for "over-and-out." To exit the write program and get the shell prompt back, type a single <**CTRL/d**>. An end-of-transmission message (*EOT*) will then be displayed on the other user's terminal, showing that you have exited the write program. The other user should then exit in the same manner.

A word of warning about the **write** command — if multiple users initiate simultaneous writes to the same terminal, that terminal screen may become garbled.

The write command is best used sparingly. Most users eventually become involved in editing and other time-consuming tasks, during which they don't want to be interrupted. To display the current write permissions to your terminal screen, execute the **mesg** command without any arguments:

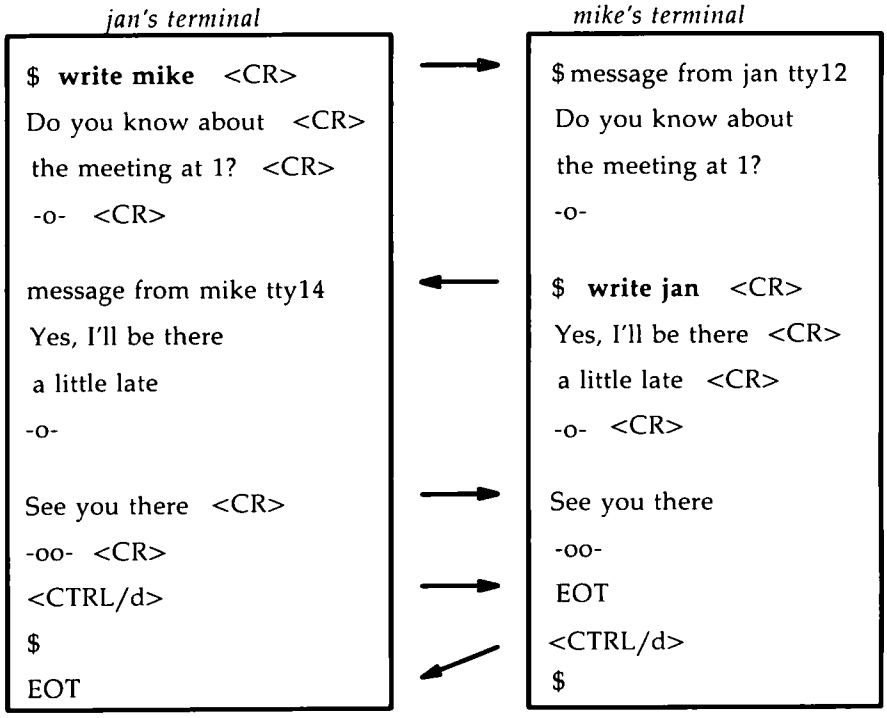
```
$ mesg<CR>
is y
$
```

The **mesg** command displays "**is y**" or "**is n**" depending on whether write permission is permitted or denied. The default is **y**, although the system administrator may change this in the system profile. Write permission can be set using the **mesg** command with the argument **y** or **n**.

```
$ mesg  n<CR>     ← denies write permission
$
```

This command can be put in your *.profile* so that every time you log on, write permission is set to **y** or **n**.

TALKING TO ANOTHER USER



SENDING MAIL TO A USER ON A REMOTE SYSTEM

The first step in communicating with a user on a remote system is to find out the *nodename* for that user's system. Then you'll need to determine whether your system can communicate with the remote system.

Users can determine the *nodename* of their system by executing the **uname** command:

```
$ uname -a<CR>
wham  wham  2.0v3  0819  3B-20A
  ⇑     ⇑     ⇑     ⇑     ⇑
system node release version hardware
 name  name
```

To determine if your system can communicate with a remote system, execute the **uuname** command. This command lists the nodenames of all the other UNIX Systems that your system can access. If the system in question is listed, you can communicate directly.

SENDING MAIL TO A USER ON A REMOTE SYSTEM

If the system in question is not listed, you will need to find a nodename in common and then communicate with one another through that machine. For example, suppose that you want to send mail to user **jim** on the **eagle** system. This system, however, is not listed when you execute **uuname**. Both you and user **jim** will then need to execute the **uuname** command on your respective systems and find a system in common. For example, suppose that the system **kingbee** appears on both lists. Your mail can then be sent through the **kingbee** system to the **eagle** system:

```
$ mail  kingbee!eagle!jim<CR>
Jim, <CR>
The stuff is in the mail. <CR>
Liz <CR>
. <CR>
$
```

If communication must be done through one or more intermediate machines, it will, of course, take a little longer before the mail arrives.

SENDING FILES TO A REMOTE UNIX SYSTEM USING uuto

- **Sending files:**

```
$ uuto file(s) nodename!login
$ uuto  poem  wham!rob<CR>
```

- **Requesting status:**

```
$ uustat<CR>
whamN6a1f  11/26-
14:56  .  .  .  /users/sue/poem
$
```

- **Canceling requests:**

```
$ uustat  -kwhamN6a1f<CR>
Job: whamN6a1f successfully killed
$
```

RECEIVING FILES FROM A REMOTE SYSTEM

```
$ mail<CR>
From uucp Wed Nov 26 13:01 EST 1996
/usr/spool/uucppublic/receive/sar/aloha/stats
from aloha!ep arrived
? d
$
```

```
$ uupick<CR>
from system aloha:  file stats
?
```

↑

m<CR>	**moves file to current directory**
a<CR>	**moves all files sent from system to current directory**
p<CR>	**prints contents of file**
d<CR>	**deletes file**
<CR>	**skips file; remains in public directory**
q<CR>	**quits**

MAIL VERSUS UUTO

- **mail** or **mailx**

```
$ mail zorro!doug < poem
```

  — ASCII files

  — File contents sent to recipient's mailbox

  — Accessed by recipient using **mail** or **mailx**
  (type **s** or **w** to save)

- **uuto**

```
$ uuto search zorro!doug
```

  — ASCII or binary files

  — File(s) received in public directory

  — Accessed by recipient using **uupick**
  (type **m** or **a** to move)

SENDING MAIL USING mailx

The **mailx** command, available since UNIX System V Release 2, is more powerful and flexible than the **mail** command. To send mail using **mailx**, type the command followed by *logins* or *alias groups*:

```
$ mailx tct kal docs<CR>
Subject: Lunch <CR>
How about getting together on Friday for lunch? <CR>
Steve <CR>
. <CR>
$
```

By default, **mailx** prompts for the subject of your mail message. Above, mail will be sent to the logins *tct* and *kal*, and the alias group *docs*. An alias group is a name that represents a list of logins. If you frequently send mail to the same logins, an alias group is a shortcut to repeatedly typing the list of logins. To define alias groups, create a file named *.mailrc* in your **HOME** directory. The *.mailrc* file contains personal parameter and variable settings used by **mailx**. There is also a system setup file for **mailx** (*/usr/lib/mailx/mailx.rc*) that is maintained by the system administrator and used to set up parameter and variable settings for everyone using **mailx**. Don't confuse *.mailrc* with your *.profile*. The *.profile* is used to set up your shell environment while the *.mailrc* file is used by **mailx**. Below is a sample *.mailrc* file:

```
group docs jwh ep jrs liz
set metoo
set dot
set sign=Ella
set autoprint
set askcc
set DEAD=Deadmail
```

Line 1 This defines an alias group named *docs*. Whenever *docs* is used as an argument on the **mailx** command line, it represents the logins *jwh*, *ep*, *jrs*, and *liz*.

Line 2 This allows your login to appear on the **mailx** command line if you want to send yourself the mail also.

Line 3 When using the **mail** command, either ".<CR>" or <CTRL/d> on a line by itself sends the message (if it was typed interactively). By default, only <CTRL/d> will send the message when using **mailx**. **set dot** permits a ".<CR>" on a line by itself to send the mail also. If this is not set in the system setup file, you may want to add this line to your *.mailrc* file.

Line 4 The set sign command is used to create an autograph string for use with the "tilde a" and "tilde A" commands.

Line 5 To have the next message automatically printed to the screen after you delete a message, use the "set autoprint" command.

Line 6 If you would like to be automatically prompted for "copies to" after you have completed each message, use the "set askcc" command.

Line 7 By default, mail that can't be delivered is saved in dead.letter file in your HOME directory. If you would like to change the name of the file in which "dead letters " are filed, use the set DEAD=filename

Refer to the **mailx** manual page for other **mailx** parameter and variable settings.

SOME mailx COMMANDS

~t	add names to the "to" line for our message
~s	revising the subject line
~c *names*	carbon copy (Cc) list
~b *names*	sending "blind copies" of the message
~h	edit "to","subject" line or add "carbon/blind copies"
~v	edit message with **vi**
~d	read the "dead.letter" contents in to your message
~A	autograph string

Commands must be typed at beginning of line.

READING YOUR MAIL USING mailx

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

n<CR>	**displays message number** *n*
d<CR>	**deletes current message from your UNIX mailbox**
s<CR>	**saves the current message in the file** *mbox*
<CR>	**skips the message; next message is displayed**

READING YOUR MAIL USING mailx

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

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

CALLING UP ANOTHER SYSTEM

If you have logins on more than one UNIX System, you may be able to call up and log onto one of them while logged onto your local system. The **cu** (call UNIX) command is frequently restricted to members of a particular group. If this is the case, see the system administrator to add your login to the appropriate group that has permission to use **cu**. It is even possible to log onto a non-UNIX System using this command (refer to the manual page for the necessary options).

If the other system's nodename is known to your local system, this can be specified as the argument to **cu**. Otherwise, specify the telephone number of the remote machine along with an indication of the modem speed:

```
$ cu  -s1200  9=1415-555-1212<CR>
```

In the above example, a speed of 1200 baud is specified using the **-s** option (the default is 300 baud). The **9=** indicates that the call is to go outside and that the system should wait for the second dial tone. The hyphens in the telephone number are optional. After typing the **cu** command and entering a <CR>, some dialing information may be displayed. If the connection succeeds, the words **Connected** followed by the **login:** prompt will appear on the screen. If only the word **Connected** appears, simply enter a <CR> and then the **login:** prompt should appear.

CALLING UP ANOTHER SYSTEM

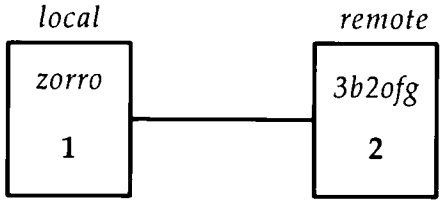
The first system that the user logs into is called the *local* system; the second system (the system accessed via **cu**) is called the *remote* system. To execute a command on the remote system, simply type the command line. To execute a command on the local system, preface the command line with ~**!** as the following shows:

```
$ ls  -Cp<CR>     ← on the remote system
calendar  junk/  scores  stats  schedule

$ ~[zorro]!ls  -Cp<CR> ← on the local system
bin/  budget/  calendar  misc/  poem
```

Note:
After the ~ is typed, *[system]*, where system is the name of the local system, is automatically displayed.

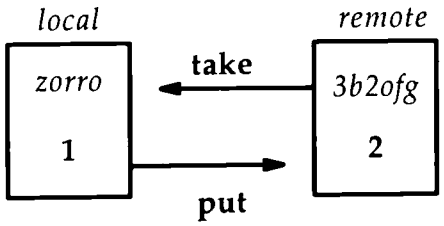
CALLING UP ANOTHER SYSTEM

```
$ uname  -a<CR>
zorro zorro 2.0v3 0619 3B-20A

$ cu  3b2ofg<CR>
Trying modem - ... 9=14155551212
Connected
login: aer<CR>
Password: <CR>
Welcome to UNIX System V.
Have a great day!
$
```




```
$ ls  -Cp<CR>
calendar junk/ scores stats schedule

$ ~[zorro]!ls  -Cp<CR>
bin/ budget/ calendar misc/ poem
```

COPYING FILES BETWEEN SYSTEMS
~%take and ~%put




```
$ ~[zorro]%take scores<CR>
stty -echo ; mesg n . . .
10 lines/112 characters

$ ~[zorro]!ls -Cp<CR>
bin   budget/   calendar   misc/   poem
scores

$ ~[zorro]%put poem fpoem<CR>
stty -echo ; cat - > . . .
30 lines/895 characters

$ ~[zorro].<CR>
Disconnected

$
```

VOLUME 2 EXERCISE

1. How can you communicate with remote systems?

2. What type of information does the uname and uuname command provide for you?

3. Using the mailx command, what do ~a, ~t, ~s, and ~v do?

4. When is it better to use uuto vs. mail?

5. How can you transfer files while logged on another system to your UNIX System?

VOLUME 2 EXERCISE - ANSWERS

1. Page 2-6, 2-7
2. Page 2-6
3. Page 2-13
4. Page 2-10
5. Page 2-17

VOLUME 2 SUMMARY
COMMUNICATING ON THE UNIX SYSTEM

Command	Description	Example
write	write with another user. To exit write, type a <CTRL/d>.	$ write jan<CR>
mesg	Reports the write permission to your terminal screen.	$ mesg<CR> is y
	Changes the write permission to your terminal screen.	$ mesg n<CR> $ mesg y<CR>
	Cancels a file transfer request.	$ uustat -kwhamN6a1<CR>
mailx	Sends mail to other user(s).	$ mailx bob<CR> Subject: Meeting meeting a 2PM .<CR>
	Read your mail.	$ mailx<CR>
uname	Reports the local system name and nodename. Second name is nodename.	$ uname -a<CR> wham *wham* ... 3B-20A
uuname	Lists all remote systems that the local system can access.	$ uuname<CR>
uuto	Sends files to remote systems.	$ uuto poem wham!jill<CR>
uustat	Reports file transfer status.	$ uustat<CR>
	Cancels a file transfer request.	$ uustat -kwhamN6a1<CR>
uupick	Used to retrieve files that were sent via uuto.	$ uupick<CR>
cu	Used to call up another system. Either use the nodename as an argument or specify the speed and phone number.	$ cu rz3bc<CR> $ cu -s1200 9=13124560968
~%take	If logged into more than one system, **take** can be used to copy a file from the remote to the local system. In the example, the copy will have the same name as the original file.	$ cu rz3bc $ ~%take scores
~%put	If logged onto more than one system, **put** can be used to copy a file from the local to the remote system. In the example, the file name of the copy is changed to *report*.	$ cu rz3ba $ ~%put rep report

VOLUME 3

File Access and Manipulation

WHAT YOU'LL BE ABLE TO DO AFTER VOLUME 3

- Print the contents of files.
- Use file name generation characters to execute commands on more than one file.
- Control who has permission to access files.

PRINTER REQUESTS

The **pg**, **cat**, and **pr** commands print the contents of a file on your terminal screen. The **lp** command is used to get a *hardcopy* from a printer. For example:

```
$ lp  poem<CR>
```

where **lp** prints the contents of the file *poem* on a printer. Although **lp** can print files on a printer, it does no formatting, and the output appears as if **cat** had been used. Also, files with restricted permissions cannot be printed this way. Therefore, files are usually formatted with **pr**, and that output is piped to **lp**:

```
$ pr  poem | lp<CR>
```

If your system has more than one printer, one of them will be the "default" or the printer that **lp** uses, unless directed otherwise. To use a printer different from the default, either:

1. Use the **-d** option with the printer name. For example:

   ```
   $ pr  poem | lp  -dlp1<CR>
   ```

   where *lp1* is the name of the printer. Ask the system administrator for the names of the printers on your system.

2. Add the following lines to your *.profile* so that every time you log on, the LPDEST variable is set to the name of the printer you prefer. This then becomes your default printer.

```
LPDEST=lp1
export LPDEST
```

   Remember, **export**ing a variable places it in your environment, where it is accessible to commands such as **lp**.

The **lp** command queues output for printing and responds with a job number. The **lpstat** command is used to report the status of **lp** requests. The **cancel** command is used to cancel a printer request. The job number, displayed by **lp**, is used as the argument to the **cancel** command.

```
$ pr poem | lp -dlp1<CR>
request id is lp1-7099 (standard input)
$ lpstat<CR>
lp1-7099      jmg     743     Aug 28 11:45  on lp1
$ cancel lp1-7099<CR>
request "lp1-7099" canceled
$
```

The **cancel** command may be used by any user to cancel a job. The argument to **cancel** can also be the printer name, for example *lp1*. If the argument is a printer name, the currently printing job is canceled.

FACTORS IN NAMING FILES

- Should describe the contents
- Maximum of 14 characters
- Recommend upper or lower case letters, numbers, underscore(_), or dot(.)
- Using *file name generation characters* to represent groups of file names with a pattern

The bracketed set can consist of a list of characters, any one of which can occur in that position of the file name, or a hyphenated range of characters, with the lowest ASCII character specified first.

```
$ budget9[023] <CR>          $ ls budget9[1-3] <CR>
budget90                     budget91
budget92                     budget92
budget93                     budget93
$                            $
```

Any number of bracketed sets may be specified anywhere in the pattern; each will match a single character. When an **!** is the first character inside the brackets, any character that is NOT a member of the set will match. Below, a NEGATED character set is used to avoid listing the budget for 91:

```
$ ls budget9[!1] <CR>
budget90
budget92
budget93
$
```

Using Combinations of ?, *, [ ]:
Any combination of file name generation characters can be used to generate file names as arguments to commands. For example, suppose a shopkeeper has accounts stored in a directory called

receivable. The shopkeeper wishes to list all the accounts receivable for 1985 and 1986, for customer names beginning with A through L:

```
$ ls  rec*/8[56]_[A-L]*<CR>
receivable/85_Allen
receivable/85_Glitchman
receivable/86_Anderson
receivable/86_Fogarty
receivable/86_Kludge
$
```

Notice that file name generation characters can be used anywhere in a path name! Above, the first asterisk saved typing out the entire directory name *receivable*.

Other Commands:

The file name generation characters are similar to some of the editor metacharacters. Remember, commands that match file names use file name generation characters; commands that match patterns of text use editor metacharacters. Thus, the **find** command, that searches directories for file names, uses file name generation characters. For example:

```
$ find . -name 'temp?' -print<CR>
```

This **find** command searches for file names beginning with *temp* followed by ONE MORE CHARACTER, such as *tempx, temp8, tempL*. The pattern must be quoted to prevent the shell from using it to match file names in the current directory.

FILE NAME GENERATION CHARACTERS

*	matches zero or more characters
?	matches any single character
[ ]	matches one of the enclosed characters

```
$ ls<CR>
budget87
budget88
budget89
budget8tot
budget90
budget91
budget92
budget93
budget9tot
budgettot
tot
$
```

examples:

```
$ ls  *tot
$ pr  * | lp
$ pr  budget*
$ pr  budget*tot | lp
$ pr  budget?tot | lp
$ cat  budget8? > total8
$ ls  budget??
$ pr  -d  budget9[023]
$ cat  budget9[1-3]
$ ls  budget[89]*
$ find .  -name 'junk*' -print
```

FILE PERMISSIONS

It is important to take steps to protect your files. As owner of your files, you control who has permission to access your files. For instance, you probably will want to prevent other users from modifying your files. You may also want to prevent some or all users from reading your files. For every file, there are three classes of users who may access the file:

- **Owner** — Every file on the UNIX System has an *owner*. Initially, the owner is the user who created the file.
- **Group** — Every file on the UNIX System belongs to a *group*. Initially, the group is the same group as the user who created the file. When many users, perhaps working on a related project, need to share files, the system administrator can set up a *group identity* for these users. The **id** command displays a user's group identity.
- **Others** — All other users.

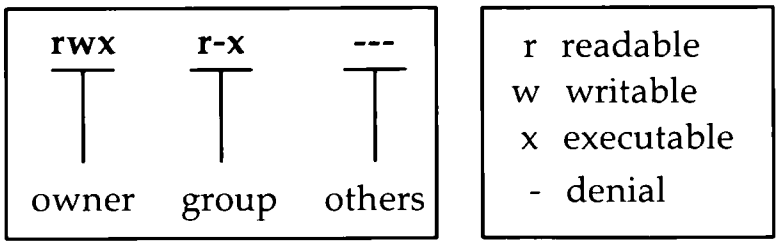
Each class of users can have a different set of permissions on a file. The **ls -l** command displays file permissions. The following are the permissions and their meanings.

Permission	Regular File	Directory
Read	Allows reading the file with **cat, pr, cp**, etc.	Allows listing the files within a directory.
Write	Allows changing the file contents by overwriting, editing, etc.	Allows new files to be added, removed, or renamed. ***need execute also**
Execute	Allows executing a file as a command (if the file is a program).	Allows changing to a directory (**cd**) and accessing files below it.

Note — Even when a file's permission prohibits writing, if the parent **directory** has write and execute permission, the file can be removed. Therefore, the best rule for protecting files is to deny **write** permission on your **directories** to users in the **group** and **other** classes.

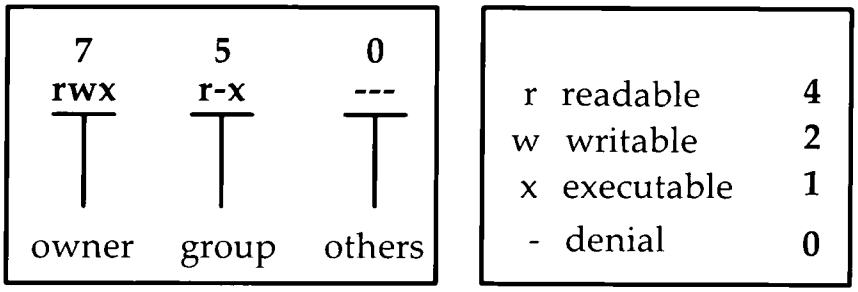
FILE PERMISSIONS

```
$ id<CR>
uid=248(jrs) gid=17(doc)

$ ls  -l<CR>
total 11
drwx------  2 jrs doc    128 Jun  9 12:58 bin
drwx------  2 jrs doc    128 Jun  9 13:58 budget
-rw-r--r--  1 jrs doc    173 Jul  7 14:01 calendar
-rw-r--r--  1 jrs doc    613 Jul 22 15:56 friends
drwxr-xr-x  2 jrs doc    112 Jul  7 15:20 fun
drwxr-xr-x  3 jrs doc    288 Aug  6 08:29 misc
drwx------  2 jrs doc    160 Jul  9 13:48 oldletters
-rw-r-----  1 jrs unixc 355 Jul  8 14:35 parts
drwxr-x---  4 jrs unixc  96 Jul 22 09:38 project
drwx------  2 jrs doc    240 Jul 28 08:32 work
$
```



CHANGING FILE PERMISSIONS

chmod *mode* file(s)




```
$ ls  -l  calendar<CR>
-rw-r--r--  1 jrs doc 173 Jul 7 14:01 calendar
$ chmod  640  calendar<CR>
$ ls  -l  calendar<CR>
-rw-r-----  1 jrs doc 173 Jul 7 14:01 calendar
$
```

```
$ ls  -ld  misc<CR>
drwxr-xr-x  3 jrs doc 288 Aug 6 08:29 misc
$ chmod  750  misc<CR>
$ ls  -ld  misc<CR>
drwxr-x---  3 jrs doc 288 Aug 6 08:29 misc
$
```

WHAT ARE THE CORRECT MODES?

```
$ chmod _____ friends<CR>
$ ls -l friends<CR>
-rw-r-----  1 jrs doc   613 Jul 22 15:56 friends
$

$ chmod _____ parts<CR>
$ ls -l parts<CR>
-rw-rw-r--  1 jrs unixc 355 Jul  8 14:35 parts
$

$ chmod _____ c*<CR>
$ ls -l c*<CR>
-rw-r--r--  1 jrs doc   173 Jul  7 14:01 calendar
$

$ chmod _____ work<CR>
$ ls -ld work<CR>
drwxr-x---  2 jrs doc   240 Jul 28 08:32 work
$

$ chmod _____ misc/p*<CR>
$ ls -l misc/p*<CR>
-rw-------  1 jrs other 96 Sep 8 16:20 misc/poem
-rw-------  1 jrs other 96 Sep 9 14:22 misc/phone
$

$ chmod _____ .<CR>
$ ls -ld<CR>
drwx------  9 jrs other 112 Jan  8 16:30
$
```

ANSWERS

```
$ chmod 640 friends<CR>

$ chmod 664 parts<CR>

$ chmod 644 c*<CR>

$ chmod 750 work<CR>

$ chmod 600 misc/p*<CR>

$ chmod 700 .<CR>
```

THE umask COMMAND

So, how do files get their original modes? There are two default modes, one for directories and one for regular files. Directories, by default, are readable, writable, and executable. In the absence of any restrictions then, a newly created directory will have a mode of **777**. Regular files, on the other hand, are NOT normally executed, so their default mode is **666**.

Frequently, the system administrator (in the system profile */etc/profile*) sets up a *mask* for regular file and directory modes. This mask is used to **take away** certain default permissions when new files are created. To see the mask on your system, execute the **umask** command:

```
$ umask <CR>
0002
$
```

The **umask** command displays a 4-digit number. The first digit is a zero and means that this is an octal number. The last three digits correspond to the three user classes: owner, group, and others. Each digit indicates the permissions, if any, that will be DENIED to that user class. The mask does not affect the permissions of existing regular files and directories, only newly created files. For example, a mask of **002** would deny write permission to **others** on new files. With this mask, the mode of newly created regular files would be 664 and the mode of newly created directories would be 775.

Suppose the system mask is **000** but you would like a mask of **022**? This would ensure that new regular files and directories would NOT have **write** permissions for the **group** and **others**.

THE umask COMMAND

To establish a mask value that is different from the system mask, execute **umask** with a three-digit argument. The three digits correspond to the permissions that will be DENIED to the owner, group, and others, respectively. For example, to establish a mask of **022** execute:

```
$ umask  022<CR>
$
```

The absence of any error messages indicates success! To establish this mask for all your login sessions, include this command line in your *.profile*. Remember, this mask only affects the mode of newly created files. The **chmod** command must be used to change the mode of existing files.

The following are popular mask values and their effect on the mode of regular files and directories:

mask	mode	
	regular files	directories
000	666	777
022	644	755
027	640	750
077	600	700

Note: A regular file, created with an editor, will NOT have execute permission. Execute permission must be added explicitly with the **chmod** command.

THE umask COMMAND
Used to set default file permissions

default	regular files	directories
umask 000	rw-rw-rw-	rwxrwxrwx
masks		
umask 022	rw-r--r--	rwxr-xr-x
umask 027	rw-r-----	rwxr-x---
umask 077	rw-------	rwx------

---

put **umask** command in *.profile*

CHANGING GROUPS

```
$ ls  -l  stats<CR>
-rw-r----- 1 tmm doc 173 Jul 7 14:01 stats
$ cat  stats<CR>
cat: cannot open stats
$ id<CR>
uid=248(jrs) gid=21(unixc)

→ $ newgrp  doc<CR>
  $ id<CR>
  uid=248(jrs) gid=17(doc)
  $ cat  stats<CR>
  This report contains proprietary
  information about our project status:
        .
        .
        .
$
```

VOLUME 3 EXERCISE

1. How many classes of users can access files?

2. How can you change the default permission on your files and directories?

3. The permission 750 implies ...?

4. How can you determine your group id and your own id?

5. How can you change your group id if you belong to more than one group?

VOLUME 3 EXERCISE - ANSWERS

1. Page 3-9
2. Page 3-11
3. Page 3-11
4. Page 3-17
5. Page 3-17

VOLUME 3 SUMMARY
MANAGING FILES

PRINTING FILES		
Command	**Description**	**Example**
cat	Displays the contents of a file.	$ cat poem<CR>
pg	Displays the contents of a file one screen at a time.	$ pg poem<CR>
pr	Displays the contents of a file formatted with a header and page numbers.	$ pr poem<CR>
	Displays the lines double-spaced.	$ pr -d poem<CR>
	Displays the lines with line numbers.	$ pr -n poem<CR>
lp	Prints files on a printer. Responds with a job number. In this example, lp5-4810 is the job number.	$ pr poem|lp<CR> request id is lp5-4810
	Prints files on a particular printer. The -d option is used to specify the printer named lp1.	$ pr poem|lp -dlp1<CR> request id is lp1-4811
lpstat	Prints information about jobs sent to the printer. Displays job numbers that can be used with the cancel command.	$ lpstat<CR> lp1-4811 jrs ...
cancel	Cancels jobs sent to the printer. Must specify the job number. The lp command responds with a job number.	$ cancel lp1-4811<CR>

VOLUME 3 SUMMARY
MANAGING FILES

FILE PERMISSIONS		
Command	**Description**	**Example**
chmod	Changes the permissions of files. In the example, 644 are the new permissions of the file named schedule. r = 4 w = 2 x = 1	$ chmod 644 schedule<CR>
id	Reports your user id and group id.	$ id<CR> uid=248(fjp) gid=1(unixc)
ls	Displays the permissions of all the files in the current directory.	$ ls -l<CR> drwxr-xr-x 2 jrs ... programs -rw-rw-r-- 1 jrs ... tests
	Displays the permissions of a regular file(s).	$ ls -l poem friends<CR> -rw-r--r-- 1 jrs ... poem -rw-r----- 1 jrs ... friends
	Displays permissions of all the files in a directory.	$ ls -l memos<CR> -rw-r--r-- 1 jrs ... memo1 -rw-r--r-- 1 jrs ... memo2
	Displays the permissions of a directory.	$ ls -ld memos<CR> drwxr-x--- 2 kal ... memos
newgrp	Changes your current group.	$ newgrp src<CR>
umask	Reports the current umask setting.	$ umask<CR> 0027
	Changes umask setting. Usually done in .profile.	$ umask 077<CR>

VOLUME 4

The vi Screen Editor: Part II

WHAT YOU'LL BE ABLE TO DO AFTER VOLUME 4

- Change text.
- Search for string of characters.
- Move and copy text.
- Split and join lines.
- Copy the contents of a file into your text.
- Run a shell command while you are in vi.
- Refresh your screen.
- Recover files.
- Set vi options.

CHANGING TEXT

r*c*	replace character
s*text*<ESC>	substitute character with *text*
*n***s***text*<ESC>	substitute *n* characters with *text*
cc*text*<ESC>	change line
*n***cc***text*<ESC>	change *n* lines
c*object text* <ESC>	change object
cw*text*<ESC>	change word
c$*text*<ESC>	change to end of line
r<**CR**>	split line
*n***J**	join *n* lines (default 2)

The **c** and **s** commands switch into input mode.

STRING SEARCHING

/string<CR>	forward search
?string<CR>	backward search
n	repeat search

Search wraps around beginning or end of buffer.

SUBSTITUTE COMMAND

:s/*string*/*newstring*/<CR>	substitute **first** occurrence of *string* with *newstring* on the **current** line
:s/*string*/*newstring*/**g**<CR>	substitute **all** occurrences of *string* with *newstring* on the **current** line
:5,25s/*string*/*newstring*/**g**<CR>	substitute **all** occurrences of *string* with *newstring* on lines 5 through 25
:1,$s/*string*/*newstring*/**g**<CR>	substitute **all** occurrences of *string* with *newstring*

REGULAR EXPRESSION METACHARACTERS

Metacharacter	Meaning
^	beginning of line
$	end of line
.	any single character
*	zero or more occurrences of preceding character

Examples:
```
/^is
:1,$s/^/<TAB>/
/it$
:50,$s/it$/it,/
:1,$s/.//
:1,$s/John.*Smith/John Smith/g
```

Above commands all end with <CR>.

COPYING AND MOVING TEXT

- Most recently deleted or yanked text goes to *unnamed buffer*.
- The put command retrieves the contents of *unnamed buffer*.

p Puts text to *right* of cursor
If lines, puts text *below* cursor

P Puts text to *left* of cursor
If lines, puts text *above* cursor

Move Text	Copy Text
1. delete	1. yank
2. move cursor	2. move cursor
3. p or P	3. p or P

THE YANK COMMAND

The yank command, **y**, is used to copy text. It saves text in the unnamed buffer **WITHOUT** removing the text, allowing you to duplicate portions of the file buffer. For example:

yw yanks a copy of current word and the following space (i.e., up to next word) into the unnamed buffer.

y5w yanks the next five words into the unnamed buffer.

*n***yy** yanks *n* lines into the unnamed buffer.

Copying (duplicating) text from one location in the buffer to another is performed in the following three steps:

1. Yank the text with a single command. You will not see the effect of this command on the screen.
2. Move the cursor to the location in the buffer where you want the new copy of the text to appear.
3. Use the put command (**p** or **P**) to place a copy of the unnamed buffer contents into the desired location in the buffer.

MISCELLANEOUS COMMANDS

xp	transpose two characters
:**r** *file*<CR>	read contents of *file* into buffer after current line
:**!***command*<CR> :!ls<CR>	execute shell command
:**!***command* **%**<CR> :!spell %<CR>	execute command with file name as argument
<CTRL/l>	clear and redraw screen (control 'el')
$ **vi -r** *file*<CR>	invoke **vi** to recover *file*

SETTING vi OPTIONS DURING YOUR EDIT SESSION

Line-numbering:

Frequently, you will want to reference particular lines. For example, you may want to do a substitute command on a range of lines (e.g., **:5,25s/Mr/Ms/g**). The command **:set number**<**CR**> (can be abbreviated **:set nu**<**CR**>) will redraw your screen with line numbers. These line numbers only appear while you are in **vi**. If you save the buffer, the line numbers are NOT saved in the file. As lines are added or deleted from the buffer, the line numbering is automatically adjusted. The command **:set nonumber**<**CR**> (abbreviated **:set nonu**<**CR**>), redraws the screen and removes the line numbers.

Displaying Tabs and End of Lines:

Sometimes you may need to determine whether your file contains <TAB>'s or spaces. For example, you may have columns of data and need to know if the columns are <TAB> separated or space separated. The command **:set list**<**CR**> will display tabs as ^I. In addition, the list option will mark the end of lines with a $. This may be useful to show any trailing <TAB>'s or spaces at the end of lines. The command **:set nolist**<**CR**> will redraw your screen, removing the dollar signs marking the end of lines and changing ^I back to a <TAB>.

SETTING vi OPTIONS DURING YOUR EDIT SESSION

Indicate Input Mode:

:set showmode<CR>

This option is useful, especially if you are new to **vi**. Since it is easy to forget whether you are in command or input mode, this option, if set, indicates on the bottom of your screen when you are in the input mode.

Define Automatic Right Margin

wm is short for **wrapmargin**. This option is useful when you are adding a large amount of text and you don't want to enter a <CR> to break lines. When this option is set to a number (0 turns wm off), lines will automatically be broken *n* spaces from the right side of the screen. On the next page, **wm=10** sets the wrapmargin to 10 spaces.

SETTING vi OPTIONS DURING YOUR EDIT SESSION

:set number	turn on line-numbering
:set nonumber	turn off line-numbering
:set list	display tabs and end of lines
:set nolist	turn off list
:set showmode	indicate input mode
:set noshowmode	turn off showmode
:set wm=10	define automatic right margin
:set wm=0	turn off wm
:set	see what options are set
:set all	see the entire option list

VOLUME 4 EXERCISE

1. How can we change a whole line?

2. How can we split or join two lines of text?

3. How can we change a string throughout the file?

4. How can we refer to the beginning of a line or end of a line?

5. How can we copy a part of a file from one place to another?

6. How can we set the line number on the buffer and see what is already set for us?

VOLUME 4 EXERCISE - ANSWERS

1. Page 4-3
2. Page 4-3
3. Page 4-5
4. Page 4-6
5. Page 4-7, 4-8
6. Page 4-10

VOLUME 4 SUMMARY
vi SCREEN EDITOR

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

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

2. *Display Text*

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

3. *Cursor Movement*

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

VOLUME 4 SUMMARY
vi SCREEN EDITOR

---

4. *Text Creation*

a*text*<ESC>	append after cursor
i*text*<ESC>	insert before cursor
o*text*<ESC>	open line below
O*text*<ESC>	open line above

5. *Delete Text*

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

6. *Searching*

/string	find next string
?string	reverse search
n	repeat last / or ?

7. *Change Text*

r*c*	replace char with *c*
*n*s*string*<ESC>	substitute *n* chars with *string*
cc*text*<ESC>	change line with *text*
*n*cc*text*<ESC>	change *n* lines
cw *text*<ESC>	change word
c$ *text*<ESC>	change to end of line
*n*J	join *n* lines
r<CR>	split line
:1,$s/string/newstring/g	substitution

VOLUME 4 SUMMARY
vi SCREEN EDITOR

8. *Copying Text*

yy	yank entire line
*n*yy	yank n lines from current line
yw	yank word
y$	yank to end of line
p	put after char (line)
P	put before char (line)

9. *Move Text*

use delete instead of yank

10. *Miscellaneous*

xp	transpose 2 characters
:r file	read file into buffer
:!spell %	run shell command on current file
<CTRL/l>	redraw screen
$ vi -r file	recovery

11. *Setting Options*

:set number	number lines
:set nonumber	turn off numbers
:set list	display tabs and end of lines
:set nolist	turn off list
:set showmode	indicate input mode
:set noshowmode	turn off showmode
:set wm=10	define automatic right margin
:set wm=0	turn off wm

12. *Variables*

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