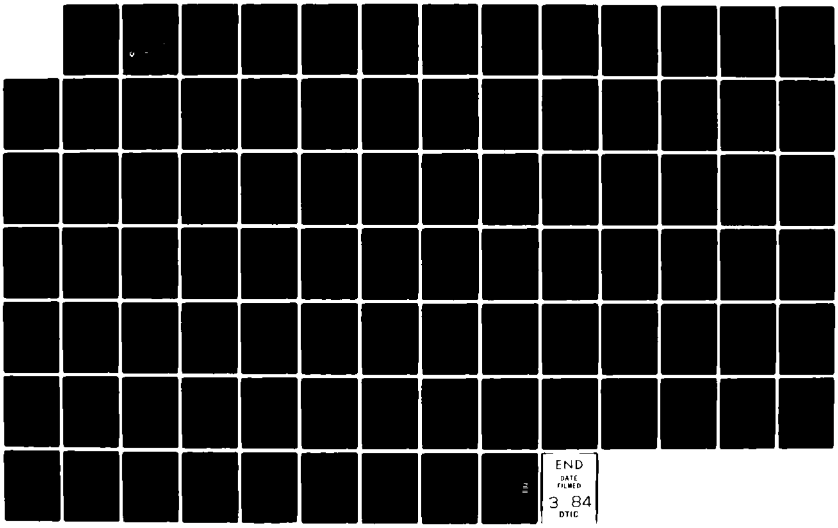
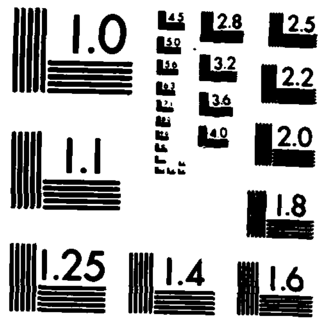
```
AD-A137 264   BEGINNERS' TUTORIAL FOR JHU UNIX AT BRL (BALLISTIC        1/1
              RESEARCH LABORATORY)(U) ARMY ARMAMENT RESEARCH AND
              DEVELOPMENT CENTER ABERDEEN PROVIN..  N L CIDER DEC 83
UNCLASSIFIED  ARBRL-SP-00036 SBI-AD-F300 373           F/G 9/2      NL
```



END
DATE FILMED
3 84
DTIC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0

SPECIAL PUBLICATION ARBRL-SP-00036

# BEGINNERS' TUTORIAL FOR JHU UNIX AT BRL

Nancy L. Cider

December 1983

US ARMY ARMAMENT RESEARCH AND DEVELOPMENT CENTER
BALLISTIC RESEARCH LABORATORY
ABERDEEN PROVING GROUND, MARYLAND

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

| REPORT DOCUMENTATION PAGE | | READ INSTRUCTIONS BEFORE COMPLETING FORM |
|---|---|---|
| 1. REPORT NUMBER<br>SPECIAL PUBLICATION ARBRL-SP-00036 | 2. GOVT ACCESSION NO.<br>AD-A137 264 | 3. RECIPIENT'S CATALOG NUMBER |
| 4. TITLE (and Subtitle)<br>BEGINNERS' TUTORIAL FOR JHU UNIX AT BRL | | 5. TYPE OF REPORT & PERIOD COVERED<br>FINAL |
| | | 6. PERFORMING ORG. REPORT NUMBER |
| 7. AUTHOR(s)<br>NANCY L. CIDER | | 8. CONTRACT OR GRANT NUMBER(s) |
| 9. PERFORMING ORGANIZATION NAME AND ADDRESS<br>US Army Ballistic Research Laboratory, ARDC<br>ATTN: DRSMC-BLT(A)<br>Aberdeen Proving Ground, MD 21005 | | 10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS<br>1L162618AH80 |
| 11. CONTROLLING OFFICE NAME AND ADDRESS<br>US Army AMCCOM, ARDC<br>Ballistic Research Laboratory, ATTN: DRSMC-BLA-S(A)<br>Aberdeen Proving Ground, MD 21005 | | 12. REPORT DATE<br>December 1983<br>13. NUMBER OF PAGES<br>94 |
| 14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) | | 15. SECURITY CLASS. (of this report)<br>UNCLASSIFIED |
| | | 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE |

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

UNIX Tutorial
Computer
Software
Introduction

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This paper is meant to help beginners get started with the UNIX operating system. It is by no means all inclusive; however, an attempt was made to touch on most topics. It will cover, first, the basics needed for day-to-day use of the system. Included will be: typing conventions, special character meanings, logging in and out, the file system, file manipulation, and inter-user communication. Second, it will provide a brief description of the available document preparation programming software. Emacs, the "ed" editor, C, Basic, Nroff and Troff will be covered. Third, a section of examples and a command summary are

20. included. Finally, there is a list of terms, for those of you who are unfamiliar with some of the language and meanings used in computing, and a reading list of UNIX documents. The reading list contains documents which I have on hand.

Dist A1

## Table of Contents

## I. INTRODUCTION

In many ways, UNIX represents the state of the art in computer operating systems. From the user's point of view, it is easy to learn and use, and its tool-oriented structure makes it very productive.

It is hard, however, for the beginner to know where to start and how to make the best use of the facilities available. The purpose of this tutorial is to point out high spots for new users, so they can get used to the main ideas of UNIX and start making good use of it quickly. It is not an attempt to rewrite the UNIX Programmer's Manual; instead, it should suggest in what order to read the manual and collect together things that are stated only indirectly there. Throughout, references to the manual will look like this: see msg(1), which means "see the article titled 'msg' in section 1 of the Programmer's Manual." If you don't have a copy of the Programmer's Manual, there is one on-line. You give the command 'man msg' to get information on the 'msg' command. You can also do a 'man man' to see how 'man' works.

There is also an on-line program called 'learn' you can use to help you get started. It is a self-paced course on various topics. It may be worth your while to spend some time going through this course as an introduction to the system.

There are four main sections to this manual:

II. Getting Started: How to type to UNIX, using simple commands, and how to log in and out.

III. Day-to-day Use: Things you need every day to use UNIX effectively; generally useful commands and the file system.

IV. Document Preparation: Preparing manuscripts is one of the most common uses for UNIX. This section contains an introduction to the simplest text processing program, and some information about more sophisticated software.

V. Programming: UNIX is an excellent system for program development. This section talks about some of the tools, but again is not a tutorial in any of the programming languages that UNIX provides.

## II. GETTING STARTED

### A. Typing To UNIX

Typing on a computer terminal is similar to using a typewriter, with the exception that some keys have special meanings to the computer. You should become familiar with these keys because some are essential and some may cause unexpected things to happen. Typing characteristics can be changed using the 'stty' command to suit various kinds of terminals and user preferences. See stty(1) for more information.

1. The Backslash Character (\). This is the UNIX escape character. It usually means that if the following character has a special meaning (see

below), the special interpretation is to be turned off. Because of this, it is a lengthy task to precisely define what this key will do; see tty(4) for more information. This key is also used extensively in the 'ed' editor.

2. The RETURN and LINEFEED Keys. Each computer terminal will have one or both of these keys, used to terminate the current line. You should use the LINEFEED key, although the computer will accept RETURN and immediately convert it to LINEFEED. On output, LINEFEEDs or RETURNs are converted to a LINEFEED and RETURN. They will hereafter be referred to collectively as RETURN. Most of the time, the computer will not process what you have typed until you have terminated it with a RETURN. If you expect something to happen, and nothing does, make sure you have ended your command with a RETURN.

3. The CONTROL Key. This key is similar to the shift key, and is used to provide extra characters which are usually nonprinting. To type the 'control C' character, for example, type the 'C' key while holding the CONTROL key depressed. Hereafter, control characters will be represented by a '^', so that 'control C' looks like '^C'.

a. ^R. This character causes UNIX to retype the current line of input as it now stands. This is useful because some special characters (e.g., BACKSPACE) may alter the appearance of the line, and because the computer may interrupt you with a message while you are typing. After the ^R, you may continue working on the line as before.

b. ^H, RUBOUT, BACKSPACE. These keys are used to correct typing mistakes on the current line. You will have to experiment with your terminal to see which of these keys your terminal accepts. They will be referred to collectively as BACKSPACE. Each time BACKSPACE is pressed, it will cause the last character typed to be deleted. On some terminals, the last character will actually disappear from the screen. On others the BACKSPACE character may do one of a few things. It may appear as an underscore, or the terminal may back up and overwrite what was there before, or it may do something entirely different. You'll just have to get used to what your terminal does.

c. ^U. If repairing the line you are working on would require extensive use of the BACKSPACE key, you may prefer to throw away the entire line by typing a ^U and then starting over. When this key is used, UNIX will give you a fresh line.

d. ^S and ^Q. These two keys have a special meaning when 'stall' is turned on using stty (on our system, 'stall' is turned on for you automaticaly). Typing a ^S will cause all output to the terminal to be stopped (stalled). This is useful for fast CRT terminals, where you may want to look closely at the output before it scrolls off the screen. Output is resumed by typing a ^Q. You can set your page length with the stty command in the form 'stty pagexx' where 'xx' is a number less than 63. This will automatically insert a ^S after 'xx' lines, so you will have to type a ^Q to start the screen moving again.

e. ^C. Most programs and commands may be halted by typing a ^C. There are exceptions; for example, the 'ed' editor will stop what it is doing, but will not go away.

f. ^I and TAB. These two characters are identical; both cause the current line position to be advanced to the nearest tab stop. These are conventionally set every eight spaces. They may be reset by using the 'stty' command.

g. ^D. This character is used to indicate an 'end-of-file' from the terminal. That is, some commands or programs which read multiple lines of input from the terminal expect input terminated by a ^D, which must be the first character typed on a new line. It is also used as a fast exit off the system.

## B. Readahead

UNIX has full readahead, which means that you can type as fast as you want, whenever you want, even when some program is typing at you. If you type during output, your input characters will appear intermixed with the output characters on your terminal, but they will be stored away by UNIX and used in the order you typed them. Thus, you can type two commands, one after the other, without waiting for the first to finish or even begin. If you get a message while you're typing, just act as if it did not occur. You don't need to back up, or type anything to get rid of the message, it will be ignored when you hit your RETURN. If you want to see what you've typed so far, use ^R.(See above.)

## C. Login

Turn on the terminal, and activate the modem if necessary. Press RETURN, until the computer asks you to 'enter class'. If the computer does not respond, press BREAK, then RETURN again. When it does respond, type the class for the computer you wish to connect with, followed by RETURN. Classes in BRL are as follows:

| Class code | Computer |
|---|---|
| bmd | BRL-BMD |
| vld | BRL-VLD |
| cyber | mfa |
| mfa | mfa |
| mfb | mfb |
| tac | BRLNET Tac |
| vgr | BRL-VGR |
| aunix | AMSAA Unix Vax |
| avms | AMSAA VMS Vax |

After you enter the class, press RETURN again, and the computer should ask for your 'User Name'. Type in the name you were given, followed by RETURN. Then the computer should ask for your 'Password'. Type in the password you were given with your User Name, followed by RETURN. You will notice that your password does not appear on the screen. This is normal, and is necessary for security purposes. Then the computer will print a message, followed by a new line and a '$'. This is the UNIX prompt. It lets you know that you may type commands. From now on the things the computer types to you will be in boldface, and the things you must type will be in normal typeface.

## D. Typing Commands

Once you've seen the prompt, you may type commands, which are requests that UNIX do something. In the rest of the manual, if you see "$ command', this means that whenever you see the '$', you can type your command. Try typing

```
$ date
```

followed by RETURN. You should get back something like

```
Tue Oct 12 10:52:29 EST 1982
```

Don't forget the RETURN after the command, or nothing will happen. If you think you're being ignored type a RETURN; something should happen. (The carriage returns aren't shown, but they have to be there.)

One very important exercise you should attempt now is changing your password. A password known only by you is your major protection against other people. Keep in mind that if someone logs on as you, he can do anything he wants with the information you have put into the system. To change your password, do the following

```
$ passwd
Your login name:
Old password:      Type in your old password
New password:      Type in what you want for the new
                   password (at least 6 characters)
Confirm:           Retype your new password
Password changed
$
```

From now on, whenever the system asks for your password, you will type in your new password. Two things to remember: you can change your password as often as you'd like, and regulations require you to change it at least once a year. Also, if someone else has been using your account for some reason, (usually, it's until they get their own) change your password after they've stopped using it.

Another command you might try is

```
$ who
```

which tells you the 'names' of everyone who is currently logged in, and the ID of their terminals. It gives something like

```
harry  ttya 041
cider  ttyJ 641
bobs   ttyb 102
```

If you would like to find out what a user's real (as opposed to login) name is, type

```
$ who is cider
```

which will respond with

```
        Nancy Cider (TBD/PMB) <cider@bmd70>
```

If you need to know a person's user name and you know their real name, you can use a command called 'grep' as follows

```
        $ grep 'Cider' /etc/password
```

You need to type their name in upper and lower case as in my example, or the 'grep' program won't be able to find them. The output should be something like

```
        cider::882:202:Nancy Cider (TBD/PMB) <cider@bmd70>:/other/cider
```

The user name you are looking for is the name at the beginning of the line, if the user is on another machine you will need both the user name and the site they are at. The name then is what is contained in the angle brackets, like "<cider@bmd70>" in my listing. When you type it, you shouldn't type the angle brackets, just use "cider@bmd70".

If you make a mistake typing a command name, UNIX will tell you. For example, if you type

```
        $ whom
```

you will be told

```
        whom: not found
```

## E. Mail

When you log in, you may sometimes get the message

```
        You have mail.
```

UNIX provides a postal system so you can send and receive letters from other users of the system. To read your mail, issue the command

```
        $ msg
```

The 'msg' program will say that it is reading your mailbox file, and will tell you how many messages 'mailbox' contains. It then gives you the mail prompt, which is '->'. To read your first message, type 't' followed by RETURN. As you type the 't', the program will finish the word by adding 'ype', to make the word 'type'. When you press RETURN, it prints out the current message. You can also read message number 1 by typing 't 1'. To read your next message, type 'n', and the program will print it. To type messages 1 through 5, type 't 1-5'.

If you want to get a list of messages, their authors, and the subject, type 'ha'. Then you can type just the one you want, by using 't n' where 'n' is the number of the mesage you wish to see (e.g., 't 5' types the 5th message).

To type any of the msg commands, simply type the first letter, the program will add the rest by itself. To delete a message, type 'd', and then the message number or RETURN for the current one. Messages are not actually deleted until you leave the 'msg' program; until then you can undelete messages with the undelete command. To back up to the previous message, type 'b'; and to jump to a known message number, type 'g' followed by the message number and RETURN.

To send a message, type 's'. The 'msg' program will respond with 'To: ' to which you reply with one or more valid user's names, on the same line separated by commas. Then it asks 'cc: ' (carbon copies); it is usually a good idea to keep a copy of a message, in case of mishaps. So you would reply with your name, and anyone else you feel should get a copy again separated by commas. It then asks 'Subject: ', to which you reply with the general idea of the message, which can be up to 30 characters in length. The fields may be left blank, if so desired, by simply hitting RETURN. Then you may begin typing your message. Each line must end with a RETURN, and the message ends with a ^D.

The send prompt will come up after you type the ^D, and is 'command or ?'. You can change your message or headings after you finish writing the message. To see what the entire message looks like, type an 'r' (review). In send, the rest of the command does not appear on the screen as it did in the 'msg' program. However, you still only need to type the first letter of the command. To edit the body of the message, you can type 'ed' (edit), or 'x' (emacs). To edit the headers on the message, type 'h' (header edit).

You can add a file to the message by typing 'f' (file include). You can also save a message in a file, by using the 'p' command. The 'send' program will ask you the name of the file you wish to save the message in. If the system goes down while you are creating a message, it will often be saved as 'draft'. If you type 'msg' and 's', you will be notified that a draft exists, and will be given a chance to delete it or append new information to it.

There is also a command '?', in both 'msg' and 'send', which is very handy. It gives a summary of the commands that you can give either program (depending on which you are in), with a short explanation for each. Finally, there are two commands you can give to exit the program. The first is 'e' (exit and update), which will update your files, and put undeleted mail in 'mbox'. To read the mail in your file 'mbox' you would have to type 'msg mbox'. It is a good idea to print out the messages in your 'mbox' that you wish to have a copy of, but don't need on the computer. After you've printed them, you can delete them. If you ever delete all files from your 'mbox' you'll get a message asking if you want 'mbox' to be deleted. Reply with a 'y', when you need a new 'mbox' to put undeleted mail in, 'msg' will create a new one for you. The second is a fast exit that does not update your files. To do this you type 'q' (quit).

If you want to send some mail to someone on the Arpanet, you have to know their login name and their host name. Then you give their name as 'name@site', when the 'msg' program asks 'To:'. For example, my name would be 'cider@brl-bmd'. There are a few walkthrough examples in the example appendix that you can use to get the general idea behind msg. This should be enough to get you started, and you can use '?', or msg(1) for further details.

## F. Writing to Other Users

At some point in your UNIX career, out of the blue will come a message like

```
        Message from joe @ ttyf
```

accompanied by a startling beep. It means that Joe wants to carry on a conversation with you, but unless you take explicit action you won't be able to write back. To respond, type the command

```
        $ write joe
```

This establishes a two-way communication path. Now whatever Joe types on his terminal will appear on yours, and vice versa. The path is rather slow. If you are in the middle of something, you have to get to a state where you can type a command. Normally, whatever program you are running has to terminate or be terminated. If you're editing, you can escape temporarily from the editor -- see ed(1).

A protocol is needed to keep what you type from getting garbled up with what Joe types. Typically, it's like this

> Joe types 'write smith', then 'Are you there?' and waits. Smith types 'write joe' and waits. Joe now types his message (as many lines as he likes). When he's finished, he signals it by typing 'o', which stands for 'over'.
>
> Now Smith types a reply, also terminated by 'o'. This cycle repeats until someone gets tired; he then signals his intent to quit with 'o+o' or 'o/o', for 'over and out'.
>
> To terminate the conversation, each side must type a ^D alone on a line. When the other person types his ^D, you will get the message 'EOT' (end of transmission) on your terminal.

If you write to someone who isn't logged in, or who doesn't want to be disturbed, you'll be told. To avoid getting any messages yourself you can type 'mesg'. 'mesg' changes writing permission to the opposite of what it was (e.g., if others were able to write to you before, they won't be able to now). If the target person is logged in, but doesn't answer after a decent interval, simply type ^D.

## G. Ending Your Session

To end a session with the computer, you must first terminate any programs or commands you may be running. When you have a prompt, type a ^D (end of file). If you wish to see some accounting information as well, you can type 'logout' instead. If you don't logout, someone else can turn on the terminal and be you, affecting what you have done. If after the terminal responds to your logout command you get a message back saying 'quota exceeded', that means you don't have enough room to store all the things you've put on the computer. You must get rid of enough files to allow you to logout, do so, then come and see me. I will try to get your quota increased so that you don't have that problem in the future. This problem is often caused by not deleting messages from your mailbox. If your mailbox file becomes large enough you may exceed quota. The solution to this is to periodically print out messages you want copies of, then delete them from your mailbox.

# III. DAY-TO-DAY USE

## A. Creating Files - The Editors

If you have to type a paper, a letter, or a program, how do you get the information stored in the machine? Most of these tasks are done with the UNIX 'text editor' called 'ed', or just 'e'. Since 'e' is thoroughly documented in ed(1) and explained in 'A Tutorial Introduction to the UNIX Text Editor', we won't spend any time here describing how to use it. All you want it for right now is to create some files. (A file is just a collection of information stored in the machine, a simplistic but adequate definition.)

To create a file with some text in it, do the following

```
$ ed, or e  (invokes the text editor)
?           (means that the file did not already exist)
a           (command to append text)
now type in
whatever text you want
.           (stops appending)
```

At this point you could do various editing operations on the text you typed in, such as correcting spelling mistakes, rearranging paragraphs and the like. Finally, you write the information you have typed into a file with the editor command

```
w junk
```

You could have replaced 'junk' with any other name you chose. 'e' will respond with the number of characters it wrote into the file called junk.

You have probably noticed by now that there are no prompts in the editor. The editor is a 'quiet' program, which means that it doesn't say anything unless something goes wrong. This means that you really have to remember if you were entering text, or editing. If you are appending and you type editor commands, the editor will take them as part of the text. Just type a period on a line by itself and edit out the extra commands.

It is usually a good idea to write your file about every page worth, just in case the computer goes down, or you make a terrible mistake. For example, you're typing along and want to get an idea of how things look. But instead of typing '1,$p' to print what you have, you type '1,$d'! This means you have lost you entire file! But if you have been writing the file every page, then you have only lost what you had entered since the last write. That's not quite as catastrophic!

If the system goes down however, usually there is a file created for each user that was editing, called 'ed-hangup'. This file is created from what you were doing when the computer went down. However, it is not always in exactly the form you had it when the computer died, so you may not want to depend on that. To edit this file, type 'ed ed-hangup'. After you make your changes, change the name of the 'ed-hangup' file, so that if the computer goes down again, you will not lose the changes you just made.

Suppose you now add a few more lines with 'a', terminate them with '.', and write the whole thing out as 'temp', using

```
a
new lines of text
to be added
.
w temp
```

We should now have two files, a smaller one called 'junk' and a bigger one (bigger by the extra lines) called 'temp'. Type a 'q' to quit the editor. If you forgot to write out your file, the editor will say 'Write file!'. To quit the editor without writing the file, type 'q!'.

You might also like to know about another editor called 'emacs'. This is a screen editor, that is, instead of editing one 'line' of a file at a time, you edit one 'screen' of a file at a time. A page is displayed on the terminal, and you use various control characters to move you around and enter text. To get into emacs, give the command

```
$ emacs filename     (to edit an existing file)
```
or
```
$ emacs RETURN       (to create a new file)
```

For more information, and a command summary, refer to the hand-out on emacs, as well as the appendix of examples. One of the examples is an emacs test file that you can copy and use, with commands you type to get yourself familiar with this screen editor. There is a new command called emacs-teach that will lead you through a comprehensive list of the possibilities available with emacs. Not all of the information is accurate with regard to our emacs, some of the commands are not implemented, and some are handled differently.

## B. <u>What Files Are Out There?</u>

The 'ls' or 'l' (for list) command lists the <u>names</u> (not contents) of any of the files that UNIX knows about. If you type

```
        $ l
```

the response will be something like

```
        junk
        temp
```

They are sorted into alphabetical order automatically, but other variations are possible. For example, if you add the optional argument 't',

```
        $ l -t
```

they are listed in the order in which they were last changed, most recent first. The '-l' option gives a 'long' listing and will produce something like

```
        -rw-r--r--   1 cider    41 Oct 12 12:56 junk
        -rw-r--r--   1 cider    78 Oct 12 12:58 temp
```

The date and time are of the last change to the file. The 41 and 78 are the number of characters in the file. The '-rw-r--r--' tells who has permission to read and write the files, and this particular grouping is the default. There are three classes of users that permission is granted for: 'owner', 'group', and 'other'.

'owner' is the user who created the file, and that person's name should be the user name in the long list. 'Cider' is the owner in the listing above, and cider's permissions appear first in the '-rw-r--r' grouping. The second in the grouping is the 'group' users; usually those users in your branch or team. The 'other' classification is last in the grouping, and is all other users. In the grouping listed above, the 'owner cider' has read and write permission, and the 'group' and 'other' users have only read permission.

To change the permissions you need to know who should have what permissions for each file, and they are set using the 'chmod' command. Permissions are represented by numbers in the command. Read permission is designated with a 4, write permission is 2, execute permission is 1, and no permission at all is a 0. To give someone more than one permission you add the appropriate numbers together. For example, if you want to have read, write, and execute permissions your number would be a 7. To give yourself read, write, and execute permissions, your group read permission, and everyone else no permissions on the 'junk' file you've created, the command would be

```
        $ chmod 740 junk
```

And then a 'ls -l' would produce

```
        -rwxr-----   1 cider    41 Oct 12 12:56 junk
        -rw-r--r--   1 cider    78 Oct 12 12:58 temp
```

Your directories can also be set for other people to have access to all or none of the files in your directory. For more information on setting permissions and resetting the default permissions, see chmod(1) and umask(1). Use only that information in the manual that pertains to the absolute mode, ignore information on the symbolic mode.

Getting back to the 'l' command, options can be combined: 'l -lt' would give the same thing as the 'l -l' above, but sorted into time order. You can also name the files you are interested in, and 'l' will list the information about them only. More details can be found in ls(1). It is generally true of UNIX programs that 'flag' arguments like '-t' precede filename arguments like 'junk'.

There is another variation of the 'l' command available. This is 'lss', which lists your files in 6 columns across the screen. This becomes especially useful when you have many files to list. The 'lss' command given now, would produce

```
junk      temp
```

C. Printing Files

Now that you've got a file of text, how do you print it so that people can look at it? There is a host of programs that do just that. One simple thing is to use the editor, since printing is often done just before making changes anyway. You can say

```
$ e junk
1,$p
```

or more simply,

```
$ e junk
1,$
```

'e' will reply with the count of the characters in 'junk' and then print all the lines in the file. After you learn how to use the editor, you can be selective about the parts you print.

There are times when it's not feasible to use the editor for printing. For example, there is a limit on how big a file 'e' can handle (about 124,000 characters or 7,000 lines). Secondly, it will only print one file at a time, and sometimes you want to print several, one after another. So here are a couple of alternatives.

First is 'type', the simplest of all the printing programs. 'type' simply copies all the files in a list onto the terminal. So you can say

```
$ type junk
```

or, to print two files,

```
$ type junk temp
```

The two files are simply concatenated and typed (hence the name 'type') onto the terminal in the order listed. The command 'cat' does a similar job, but it puts in the control characters that may be stripped by 'type'.

'pr' produces formatted printouts of files. As with 'type', 'pr' prints all the files in a list. The difference is that at the top of each page it produces headings with date, time, page number and file name for each file specified and extra lines to skip over the fold in the paper. Thus

```
$ pr junk temp
```

will list 'junk' neatly, then skip to the top of a new page and list 'temp' neatly.

'pr' will also produce multi-column output; as in the following

```
$ pr -3 junk
```

which prints 'junk' in 3-column format. You can use any reasonable number in place of '3', and 'pr' will do its best.

It should be noted that 'pr' is <u>not</u> a formatting program in the sense of making new paragraphs and justifying margins. The true formatters are 'nroff', 'troff', and 'vroff', which you will get to in the section on document preparation.

There are also programs that print files on a high-speed printer. To print 'junk' on the line printer, type

```
$ lpr junk
```

and the cursor will come back when 'lpr' is finished See lpr(1) for more information.

## D. Shuffling Files About

Now that you have some files in the file system and some experience in printing them, you can try bigger things. For example, you can move a file from one place to another (which amounts to giving a file a new name), like this

```
$ mv junk precious
```

This means that what used to be 'junk' is now 'precious'. If you do an 'l' command now, you will get

```
precious
temp
```

Beware that if you move a file to another one that already exists, the contents of the file before the 'mv' command was executed are lost forever.

If you want to make a copy of a file (that is, to have two versions of something), you can use the cp command.

```
        $ cp precious temp1
```

makes a duplicate copy of 'precious' in 'temp1'. This is a good idea when you want to make many changes to a file. Copy the file to another file with a new name, then edit the new file. This way, if anything weird happens to new file, you still have the old one to fall back on.

Finally, when you get tired of creating and moving files, there is a command to remove files from the file system

```
        $ rm temp temp1
```

will remove the files named. You will get a warning message if one of the named files wasn't there.

## E. What's in a Filename

So far we have used filenames without ever saying what's a legal name, so it's time for a couple of rules. First, filenames are limited to 14 characters, which is enough to be descriptive. Second, you must refer back to these files later, so pick a name that means something to you. You must type this name in every time, so it would be worthwhile to keep it short. Keep in mind that if you make it too short, you won't remember what was in the file, so you must compromise somehow. Third, you must never use a filename that has a space in it anywhere! This will cause many unexpected and undesired results! Instead of a space, use a period, hyphen, or underscore.

Finally, although you may use almost any character in a filename, common sense says you should stick to ones that are visible, and that you should probably avoid characters that might be used with other meanings. We already saw, for example, that in the 'l' command, 'l -t' meant to list in time order. So if you had a file whose name was '-t', you would have a tough time listing it by name. There other characters which have special meaning either to UNIX as a whole, or to numerous commands. To avoid pitfalls, you would probably do well to use only letters, numbers, and the period. (Don't use the period as the first character of a filename, for reasons too complicated to go into here.)

On to some more positive suggestions. Suppose you're typing a large document like a book. Logically this divides into many small pieces, like chapters and perhaps sectons. Physically it must be divided too, for 'e' will not handle big files. You might have a separate file for each chapter, called

```
        chap1
        chap2
        etc...
```

Or, if each chapter were broken into several files, you might have

```
        chap1.1
        chap1.2
        chap1.3
        ...
        chap2.1
        chap2.2
        etc...
```

You can now tell at a glance where a particular file fits into the whole.

There are advantages to a systematic naming convention which are not obvious to the novice UNIX user. What if you wanted to print out the whole book? You could say

```
        $ pr chap1.1 chap1.2 chap1.3 ...
```

but you would get tired pretty fast, and would probably even make mistakes. Fortunately, there is a shortcut. You can say

```
        $ pr chap*
```

The '*' means 'anything at all', so this translates into 'print all files whose names begin with 'chap'', listed in alphabetical order. This shorthand notation is not a property of the 'pr' command, by the way. It is system-wide, a service of the program that interprets the commands (the 'shell', 'sh'). *Using that fact, you can see how to list the names of files* of the book

```
        $ l chap*
```
produces
```
        chap1.1
        chap1.2
        chap1.3
        ...
```

The '*' is not limited to last position in a filename -- it can be anywhere. Thus

```
        $ rm *junk*
```

removes all files that contain 'junk' as any part of their name. As a special case, '*' by itself matches every filename, so

```
        $ pr *
```

prints all the files (alphabetical order), and

```
        $ rm *
```

removes <u>all</u> files. (You had better be sure that's what you wanted to say!)

The '*' is not the only pattern-matching feature available. Suppose you want to print only chapters 1 through 4, and 9 of the book. Then you can say

```
        $ pr chap[12349]*
```

The '[...]' means to match any of the single characters inside the brackets. You can also do this with

```
        $ pr chap[1-49]*
```

'[a-z]' matches any single character in the range 'a' through 'z'. there is also a '?' character, which matches any single character, so

```
        $ pr ?
```

will print all files which have single-character names. Of these niceties, '*' is probably the most useful, and you should get used to it. The others are neat extras and worth knowing, but will probably take a while to remember.

If you should ever have to turn off the special meaning of '*', '?', etc., enclose the entire argument in quotes (double), as in

```
        $ l "next?"
```

## F. What's in a Filename, Continued

When you first made that file called 'junk', how did UNIX know that there wasn't another 'junk' somewhere else, especially since the person in the next room may also read this tutorial? The reason is that generally each user of UNIX has his own 'directory', which contains only the files that belong to him. When you create a new file, unless you take specific action the new file is made in your own directory, and is unrelated to any other file of the same name that might exist in any other directory.

The set of all files that UNIX knows about are organized into a (usually big) tree structure, with your files located several branches up into the tree. It is possible for you to 'walk' around this tree, and to find any file in the system, by starting at the root of the tree and walking along the right set of branches.

To begin, type

```
        $ lss /
```

'/' is the name of the root of the tree (a convention used by UNIX). You will get a response something like this

```
        backup   boot    etc     ofc     tmp     usr
        bin      d       lib     other   unix    vld
        tbd      dev     mnt     sys
```

This is a collection of the basic directories that UNIX knows about. Here, '/bmd' could be a directory which contains tbd users' files. Now try,

```
        $ lss /bmd
```

This should list a long series of names, among which is your login name. Finally, try

```
        $ l /bmd/yourname
```

You should get what you get from a plain

```
        $ l
```

By the way, '/bmd/yourname' is your home directory. That is, when you sign on, that is where you will be. You can do anything with these files as well as any subdirectories you make, but there is more to directories than this.

Try

```
        $ type /bmd/yourname/junk
```

(if 'junk' is still around). The name '/bmd/yourname/junk' is called the 'pathname' of the file that you normally think of as 'junk'. 'Pathname' has an obvious meaning: it represents the full name of the path you have to follow through the tree of directories to get to a particular file. It is a universal rule in UNIX that anywhere you can use an ordinary filename, you can use a pathname.

This isn't too exciting if all the files of interest are in your own directory, but if you work with someone else or on several projects concurrently, it becomes handy indeed. For example, your friends can type your book by saying

```
        $ pr /bmd/yourname/chap*
```

Similarly, you can find out what files your neighbor has by saying

```
        $ l /bmd/neighborname
```

or make your own copy of one of his files by

```
        $ cp /bmd/neighborname/hisfile yourfile
```

If your neighbor doesn't want you poking around in his files, or vice versa, privacy can be arranged. Each file and directory has read-write-execute permissions for the owner, a group, and everyone else, to control access. Go back to part B of this section, the examples in the first appendix, or ls(1) and chmod(1) for details. As a matter of observed fact, most users find openness of more benefit than privacy most of the time.

As a final experiment with pathnames, try

```
        $ lss /bin /usr/bin
```

Do some of the names look familiar? When you run a program, by typing its name after a '$', the system simply looks for an executable file of that name. It looks first in your directory (where it typically doesn't find it), then in '/bin' and finally in '/usr/bin'. These are the two default directories that contain command files. There is nothing magic about commands like 'type' or 'l', except that they have been collected into two places to be easy to find and administer. You may also look into other directories if they are accessible.

What if you work regularly with someone else on common information in his directory. You would save time and effort if you could access his files without typing the entire path name. This is done by changing the directory that you are currently in, by

```
        $ cd /bmd/yourfriend
```

Now when you use a filename in something like 'type' or 'pr', it refers to the files in your friend's directory. Changing directories doesn't affect permissions associated with the file -- if you couldn't access a file from your directory, changing to another directory won't alter that fact. If you forget what directory you're in, use

```
        $ pwd
```

('print working directory') to find out.

It is often convenient to arrange one's files so that all the files related to one thing are in a directory separate from other projects. For example, when you write your book, you might want to keep all the text in a sub-directory called book. So make one with

```
        $ mkdir book
```

then go to it with

```
        $ cd book
```

then start typing chapters. The book is now found in (presumably)

```
        /bmd/yourname/book
```

To delete a directory, see rmdir(1).

You can go up one level in the tree of files (in the computer business, trees are always upside down; thus 'up' means 'toward the root') by saying

```
        $ cd ..
```

'..' is the name of the parent of whatever directory you are currently in. For completeness, '.' is an alternate name for the directory you are in.

## G. Using Files Instead of the Terminal

Most of the commands you have seen so far produce output on the terminal; some, like the editor, also take their input from the terminal. It is universal in UNIX that the terminal can be replaced by a file for either or both of input and output. As one example, you could say

```
        $ l
```

to get a list of files. But you can also say

```
$ l > filelist
```

to get a list of your files and put it into the file 'filelist'. ('filelist' will be created if it doesn't already exist, or overwritten if it does.) The symbol '>' is used throughout the UNIX command system to mean 'put the Standard Output on the following file, rather than the terminal'. Nothing is produced on the terminal. As another example, you could concatenate several files into one by capturing the output of 'type' in the file 'temp' by

```
$ type f1 f2 f3 > temp
```

This is especially handy when you know you are going to want more than one copy of the output of a program, and you don't want to rerun the program every time you want a copy. It is also helpful when you want the output to be used by someone else on another terminal.

Similarly, the symbol '<' means 'use as Standard Input the contents of the following file, instead of the terminal keyboard'. Thus, you could make up a script of commonly used editing commands and put them into a file called 'script'. Then you can run the script on a file by saying

```
$ ed file < script
```

There is a file in the system called '/etc/passwd' that contains a list of user and login names, the division and branch, and other accounting information for each user. If you wanted to print out this file, you might have problems reading the output, because the fields are separated by colons. You can change the colons to tabs to make the output more readable by using the 'tr' command. 'tr' translates characters from Standard Input, and can be used here in the form

```
$ tr ':' 'TAB' < /etc/passwd
```

Where I have put the word TAB, you would press the TAB key, or use ^I. This will give you a cleaner listing on the terminal, in a form that is much more readable.

## H. Pipes

One of the novel contributions of UNIX is the idea of a pipe. A pipe is simply a way to connect the output of one program to the input of another program, so the two run as a sequence of processes -- a pipeline.

For example,

```
$ pr f g h
```

will print the files 'f', 'g', and 'h', beginning each on a new page. Suppose you want them run together instead. You could say

```
$ type f g h > temp
$ pr temp
$ rm temp
```

but this is more work than necessary. Clearly what you want is to take the output of 'type' and connect it to the input of 'pr'. So let's use a pipe, as follows

```
        $ type f g h | pr
```

The vertical bar means to take the output from 'type', which would normally have gone to the terminal, and put it into 'pr', which formats it neatly. Since 'pr' uses characters as they appear in the pipeline, this version runs faster than the one using the 'temp' file, because you don't have the time lag for creating a new file.

Pipes are also handy for counting things, eg. the number of files you have, the number of users logged in, or how many instances a certain sequence of characters occurs. The program used for counting is 'wc' (word count). To count the number of users logged-on, use the command

```
        $ who | wc
```

To count the number of files you have, use

```
        $ ls | wc
```

To count the number of filenames that contain a sequence of characters, you could use

```
        $ ls | grep ".bas" | wc
```

This example gives as its result the number of BASIC files, since BASIC programs must have a '.bas' extension on them.

Many UNIX programs are written so that they will take their input from one or more files if file arguments are given; if no arguments are given they will be read from the terminal. Any program that reads from the terminal can read from a pipe instead; any program that writes on the terminal can drive a pipe. You can have as many elements in a pipeline as you wish.

## I. The Shell

We have already mentioned once or twice the mysterious 'shell', which is in fact 'sh'. The shell is the program that interprets what you type as commands and arguments. It also looks after translating '*', etc., into lists of filenames. It handles the redirection of input and output, pipes, and some new things besides. For example, you can start two programs with one command line by separating the commands with a semicolon; the shell recognizes the semicolon and breaks the line into two commands. Thus

```
        $ date ; who
```

does both commands before returning with a '$'.

You can also have more than one program running simultaneously if you wish. For example, if you are doing something time-consuming, like formatting a document using 'nroff', and you don't want to wait around for the results before starting somethin else, you can say

```
        $ nroff -msBRL manuscript > newman &
```

The ampersand at the end of a command line says 'start this command running, then take further commands from the terminal immediately'. Thus the nroff will begin, but you can do something else at the same time. This is referred to as 'running in the background'. The reason for putting the results in a new file is to avoid having the results from the nroff command printed while you're working on something else.

You can also say

```
        $ (command 1; command 2; command 3) &
```

to start these commands in the background, or you can start a background pipe-line with

```
        $ (command 1 | command 2) &
```

When you initiate a command with '&', UNIX replies with a number called the process number, which identifies the command in case you later want to stop it. If you do, you can say

```
        $ kill process #
```

If you forget the process number, you can use 'ps' to print out a list of the processes currently running and their corresponding process numbers. For more information, see ps(1).

Just as you can tell the editor or some similar program to take its input from a file instead of from the terminal, you can tell the shell to read a file to get commands. For instance, suppose you want to set tabs on your terminal, and find out the date and who's on the system every time you log in. Then you can put the three necessary commands ('stty tabs'; 'type /usr/pub/tabs'; 'date'; 'who') into a file, let's call it 'xxx', and then run it with either

```
        $ sh xxx
```
or
```
        $ sh < xxx
```

This says to run the shell with the file 'xxx' as input. The effect is as if you had typed the contents of 'xxx' on the terminal. (If this is to be a regular thing, you can eliminate the need to type 'sh xxx' by putting it in your '.profile'. See chmod(1) and sh(1).) The shell has quite a few other capabilities as well, some of which we'll get to in the section on programming.

## IV. DOCUMENT PREPARATION

UNIX is used extensively for document preparation. There are three major formatting programs, that is, programs which produce a text with justified right margins, automatic page numbering and titling, automatic hyphenation, etc. The simplest of these formatters is 'nroff', which in fact is simple enough that if you type almost any text into a file and 'nroff' it, you will get plausibly formatted output. You can do better with a little knowledge, but basically it's easy to learn and use.

'Nroff' is designed to produce output on CRT terminals, typewriter-like terminals (Diablos, Spinwriters, etc.), line-printers, and the like. Another formatter, 'troff' (pronounced tee-roff'), instead drives a Graphic Systems phototypesetter, which produces very high quality output photographic paper. There is a third formatter called 'vroff' which puts its output onto a Versatec printer. All three of these packages operate similarly except for a few specialized commands. Therefore, we'll just deal with generalities, and leave the specifics for you to discover for yourself.

However, 'nroff' and 'troff' are relatively hard to learn to use effectively, so several 'packages' of canned formatting requests are available which let you do things like paragraphs, running titles, multicolumn output, and so on, with little effort.

## A. Nroff

The basic idea of 'nroff' and 'troff', is that the text to be formatted contains within it 'formatting commands' that indicate in detail how the formatted text is to look. For example, there might be commands that specify how long lines are, whether to use single or double spacing, and what running titles to use on each page. In general, you don't have to spell out all of the possible formatting details. Most of them have 'default values', which you will get if you say nothing at all. For example, unless you take special steps, you'll get single-spaced margins and 58 text lines per page when you 'nroff' a file. This is the reason that 'nroff' is so simple -- most of the decisions have been made for you. Some things do have to be done, however. If you want a document broken into paragraphs, you will have to tell 'nroff' where to add the extra blank lines. This is done with the '.sp' command, as follows

```
this is the end of one paragraph.
.sp
This begins the next paragraph ...
```

In 'nroff', formatting commands consist of a period at the beginning of a line, followed by two letters, an optional argument, and nothing more. The '.sp' command tells 'nroff' to finish printing any of the previous line that might be still unprinted, then print a blank line before continuing. You can have more than one space if you wish; '.sp 2' asks for 2 spaces, and so on.

If you simply want to ensure that subsequent text appears on a fresh output line, you can use the command '.br' (for 'break') instead of '.sp'.

Most of the other commonly used 'nroff' commands are equally simple. For example, you can center one or more lines with the '.ce' command.

```
.ce
Title of Paper
.sp 2
```

This causes the title to be centered, then followed by two blank lines. As with '.sp', '.ce' can be followed by a number: in that case, that many lines are centered. '.ul' underlines lines, and can also be followed by a number, as follows.

```
.ce 2
.ul 2
An Earth shaking Paper
.sp
John Q. Scientist
```

Will center and underline the two text lines. Notice that the '.sp' between the lines is not part of the line count.

You can get multiple-line spacing instead of the default single-spacing with the '.ls' command

```
.ls 2
```

causes double spacing.

If you're typing things like tables, you will not want the automatic filling-up and justification of output lines that is done by default. You can turn this off with the command '.nf' (no-fill), and then back on again with '.fi' (fill). Thus

```
this section is filled by default.
.nf
here lines will appear just
as you typed them -
no extra spaces, no moving of words.
.fi
Now go back to filling up output lines.
```

You can change the line-length with '.ll', and the left margin (the indent) by '.in'. These are often used together to make offset blocks of text

```
.ll -10
.in +10
          this text will be moved 10 spaces to
          the right and the lines will also be
          shortened by 10 characters from the
          left.  The '+' and '-' mean to
          change the previous value by that
          much.  Now revert:
.ll +10
.in -10
```

Notice that '.ll +10' adds ten characters to the current line length, while '.ll 10' makes the line ten characters long.

The '.ti' (temporary indent) command indents (in either direction) just like '.in', except that it only acts on the following line of text. Thus, to make a new paragraph with a 10-character indent, you would say

```
.sp
.ti +10
          New paragraph ...
```

You can put running titles on both the top and bottom of each page, like this

```
.ds LH left top
.ds CH center top
.ds RH right top
.ds LF left bottom
.ds CF center bottom
.ds RF right bottom
```

The headers and/or footers are divided into three parts. If any part of the title is left out, that part of the title will be left blank. If you use a percent sign anywhere in a header or footer, the current page number will be inserted. So to get centered page numbers with dashes around them, at the top, use

```
.ds CH -%-
```

You can skip to the top of a new page at any time with the '.bp' command; if '.bp' is followed by a number, that will be the new page number.

The foregoing is probably enough about 'nroff' for you to format most everyday documents. Read nroff(1) for more details.

## B. Hints for Preparing Documents

Most documents go through several versions (always more than you expected) before they are finally finished. Accordingly, you should do whatever possible to make the job of changing them easy.

First, when you do the purely mechanical operations of typing, type so subsequent editing will be easy. Start each sentence on a new line. Make lines short, and break lines at natural places, such as after commas and semicolons, rather than randomly. Since most people change documents by rewriting phrases and adding, deleting and rearranging sentences, these precautions simplify any editing you have to do later.

Also, remember not to type a space at the end of the line after a period. The formatting programs will automatically insert two spaces after periods (and other such punctuation) at the end of each line. However, if one or more spaces follow such punctuation, the formatters assume special treatment and generate only one space as a separator.

The second aspect of making changes easy is not to commit yourself to formatting details too early. For example, if you decide that each paragraph is to have a space and an indent of 10 characters, you might type, before each

```
        .sp
        .ti +10
```

But what happens when later you decide that it would have been better to have no space and an indent of only 5 characters? It's tedious indeed to go back and patch this up.

Fortunately, all of the formatters let you delay decisions until the actual moment of running. The secret is to define a new operation (called a 'macro'), for each formatting operation you want to do, like making a new paragraph. You can say, in all three formatters

```
        .de ni
        .ti +15
        ..
```

This defines '.ni' as a new 'nroff' (or 'nroff or 'troff') operation, whose meaning is exactly

```
        .ti +15
```

(The '..' marks the end of the definition.) Whenever '.ni' is encountered in the text, it is as if you had typed the definition in place of it. It means that everytime .ni is typed you will have a temporary indent of 15 spaces on the following line.

The beauty of this scheme is that now, if you change your mind about what a paragraph should look like, you can change the formatted output merely by changing the definition of '.ni' and rerunning the formatter.

As a rule of thumb, for all but the most trivial jobs, you should type a document in terms of a set of macros like '.ni' and then define them appropriately. As long as you have entered the text in a systematic way, it can always be cleaned up and reformatted by a judicious combination of editing and macro definitions. The packages of formatting commands that we mentioned earlier are simply collections of macros designed for particular formatting tasks.

## C. Standard Macro Packages ('msBRL' and 'mm')

There exist standard 'macro' commands for 'nroff', 'troff', and 'vroff' specifically designed for producing papers. Although the macro packages are made up of 'nroff'-like commands, the macros that are defined give you a much easier method for controlling the style and layout of your paper. Also, the commands are defined for all three packages, so that papers written for one formatter can be reproduced by another if desired.

There are a number of styles which you can request: the 'Technical Memorandum', the 'Released Paper', and the 'Mike Lesk paper saving Released Paper' for the 'msBRL' package. We shall concern ourselves here only with the 'Released Paper' format, as it is the most heavily used of the three. This paper is an example of what you can do with the 'msBRL' macro package.

The command file (what you feed to the formatter, text and commands interspersed) for a simple paper would look something like

```
.RP
.TL
Title of the document (one or more lines)
.AU
Author(s) (may also be several lines)
.AI
Author's institution(s) if wanted.
(This command and text need not appear)
.AB (optional)
Abstract to be placed on cover sheet.
.AE (Abstract End) used only if .AB invoked
.SH (Section Heading)
As much heading as you would like ...
.PP (Paragraph begins)
First paragraph.
Type your text here,
as many lines of text as you like,
as much text on each line
as
suits
you.
.PP
Second paragraph.
The same thing over and over again ...
.SH
Another section heading.
.PP
Yet another paragraph
.
.
.
<End of Paper>
```

Once you have entered your text (with the macro requests included) into a file, you can print your paper with 'nroff' by typing

```
$ nroff -msBRL your text file name(s) ...
```

and 'nroff' will do the rest! Similarly, you can substitute 'troff' and 'vroff' where 'nroff' is above. The '-msBRL' informs the formatters to use the 'msBRL' macro package.

For a more detailed discussion of the macro packages, see: 'NROFF/TROFF User's Manual' by Joseph Ossanna, or 'Text Processing' from SECAD listed in the bibliography of this manual.

## D. Printing Travel Orders

There are a number of coordinating programs written by various persons to make writing travel orders a little easier. To begin you must link a few files into your directory, as follows

```
$ ln /bmd/steven/tbdfiles/to.copy
$ ln /bmd/steven/tbdfiles/person.copy
$ ln /bmd/steven/tbdfiles/place.copy
```

The 'ln' command makes a link to the files. You now have a directory listing of the file and it seems as if you have a copy of it. However, you really only have a pointer to that file, so when you execute it, you are really executing the copy in '/bmd/steven/tbdfiles', not your home directory. Therefore, it is important that you do not change these files. Otherwise, no one else will be able to use them properly. If you plan to change things in these macro files, use the 'cp' command in place of 'ln'. To remove them from your directory, you can still use the 'rm' command.

To begin writing a travel order, issue the command

```
$ person.copy
```

Now the program will begin by prompting you for the filename for the person file, then for the things you need to fill out in the travel order that pertain to an individual (i.e., name, address, phone, etc.). You may want to keep this file in your directory for any future travel orders on this particular person.

In order to fill out the rest of the information (i.e., that information specific to just the one travel order), you must give the command

```
$ place.copy
```

This time the program will prompt you for a file name for your travel file, and then the travel information (i.e., purpose of the trip, no. of days travel, date leaving, etc.). When you get to the Mode of Transportation, there are various possibilities, but only the codes are listed. The codes and their meaning translate as follows

| | | |
|---|---|---|
| CR | Commercial | Rail |
| CA | ' | Air |
| CB | ' | Bus |
| CS | ' | Ship |
| GA | Government | Auto |
| GV | ' | Vessel |
| GS | ' | Ship |
| P | Privately Owned Vehicle | |

to actualy write the travel order, type

```
$ to.copy personfile travelfile
```

but, before you press RETURN (You do remember the RETURNS now, don't you?) put the form for the order in the printer and line it up, then turn the printer on. Make sure that the margin is lined up properly, or you may wind up going by bus instead of flying.

## E. Miscellany

In addition to the basic formatters, UNIX provides a host of supporting programs. 'eqn' and 'neqn' let you integrate mathematics into the text of a document, in a language that closely resembles the way you would speak it. 'tbl' formats tables of various types, where you may enter combinations of labels and data. 'spell' detects possible spelling mistakes in a document. 'grep' looks for lines containing a particular text pattern (rather like the editor's context search does, but on a whole series of files). For example,

```
$ grep 'ing$' chap*
```

will find all lines ending in the letters 'ing' in the series of files 'chap*'. (It is almost always a good practice to put quotes around the pattern you're searching for, in case it contains characters that have a special meaning for the shell, as in this case '$' is special.)

'wc' counts the words and lines in a set of files. 'tr' translates characters into other characters; for example, it will convert upper to lower case and vice versa. This translates upper case letters into lower case

```
$ tr '[A-Z]' '[a-z]'
```

'diff' prints a list of the differences between two files, so you can compare two versions of something automatically (which certainly beats proofreading by hand). 'sort' sorts files in a variety of ways; 'cref' makes cross-references; 'index' makes concordances and indexes; 'ptx' makes a permuted index (keyword-in-context listing).

Most of these programs are either independently documented (like 'eqn' and 'neqn'), or are sufficiently simple that the description in the 'UNIX' Programmer's Manual' is adequate explanation.

## V. PROGRAMMING

UNIX is a marvelously pleasant system for writing programs; productivity seems to be an order of magnitude higher than on other interactive systems. There will be no attempt made to teach any of the programming languages available on UNIX, but a few words of advice are in order. First, UNIX is written in C, as is most of the applications code. If you are undertaking anything substantial, C is the reasonable choice. More on that in a moment. But remember that there quite a few programs already written, some of which have substantial power.

The editor can be made to do things that would normally require special programs on other systems. For example, to list the first and last lines of each of a set of files, say a book, you could laboriously type

```
$ e chap1.1
1p
$p
q
$ e chap1.2
```

```
        1p
        $p
        q
        etc...
```

But instead you can do the job once and for all. Type

```
        $ l chap* > temp
```

to get the list of filenames into a file. Then edit this file to make the necessary series of editing commands (using the commands of 'emacs'), and write it into 'script'. Now the command

```
        $ e < script
```

will produce the same output as the laborious hand typing.

The pipe mechanism lets you fabricate quite complicated operations out of spare parts already built. For example, the first draft of the spell program was (roughly)

```
        type ...   (collect the files)
        |tr ...    (put each word on a new line, delete
                    punctuation, etc.)
        |sort      (into dictionary order)
        |uniq      (strip out the duplicates)
        |comm      (list words found in text but not
                    in dictionary)
```

## A. Programming the Shell

An option often overlooked by newcomers is that the shell is itself a programming language, and since UNIX already has a host of building-block programs, you can sometimes avoid writing a special purpose program merely by piecing together some of the building blocks with shell command files.

As an unlikely example, suppose you want to count the number of users on the machine every hour. You could say

```
        $ date
        $ who | wc -l
```

every hour, and write down the numbers, but that is rather primitive. The next step is probably to say

```
        $ (date; who | wc -l) >> users
```

which uses '>>' to append to the end of the file 'users'. (We haven't mentioned '>>' before -- it's another service of the shell.) Now all you have to do is put a loop around this, and ensure that it's done every hour. Remember that all programs are just files, and can therefore be created with any of the editors you know about. Thus, place the following commands into a file with either ed or emacs, and call it 'count'

```
        : loop
        (date; who | wc -l) >> users
        sleep 3600
        goto loop
```

The command ':' is followed by a space and a label, which you can then 'goto'. Notice that it's quite legal to branch backwards. Now if you issue the command

```
        $ sh count &
```

the users will be counted every hour, and you can go on with other things. (You will have to use 'kill' to stop counting.)

If you would like 'every hour' to be a parameter, you can arrange for that too. You can change 'sleep 3600' to 'sleep $1', so the program looks like

```
        : loop
        (date; who | wc -l) >> users
        sleep $1
        goto loop
```

'$1' means the first argument when this procedure is invoked. If you say

```
        $ sh count 60
```

it will count every minute. A shell program can simultaneously access up to nine arguments, '$1' through '$9', but can have more. See shift(I) on how to access the extra ones.

The other aspect of programming is conditional testing. The 'if' command can test conditions and execute commands accordingly. You could arrange that the 'count' procedure count every hour by default, but allow an optional argument to specify a different time. Simply replace the 'sleep $1' line by

```
        if $1x = x then sleep 3600 else sleep $1 fi
```

The construction

```
        if $1x = x
```

tests whether or not '$1', the first argument, was present. If you use an else clause, you need to end the 'if' with a 'fi'.

More complicated conditions can be tested: you can find out the status of an executed command, and you can combine conditions with 'and', 'or', 'not' and parentheses -- see if(I). You should also read shift(I) which describes how to manipulate arguments to shell command files.

## B. Programming in BASIC

For small, 'quick and dirty' programs, or programs that you are only going to use a few times, BASIC is probably the best choice. It is easy to learn, has a reasonably powerful syntax, and gives excellent diagnostics and debugging aids. We have interactive DEC RSTS BASIC-PLUS that has been adapted to run under UNIX.

To get into BASIC, simply type

```
$ basic
```

BASIC will respond with something like

```
Revised 02.11.76
Ready
```

and you are ready to go! You can use BASIC to create your program, if you don't want to use one of the editors. If you want to use one of the editors, you will have to number the lines of text sequentially, and the file name must have the extension '.bas' (e.g., test.bas). After you enter the BASIC program you can load in your created file with 'old filename'.

If you wish to use BASIC to create your text file, enter the BASIC program with the command 'basic', then say 'new filename'. This creates a place for your new file. You can then enter your program by giving the line number and the text for that line. An example would be

```
10 PRINT "Hello there!"
```

Once the file has been created you may modify any lines by typing the line number and the replacement text. The new file is then saved by typing 'save' if this is a new file, or 'replace' if it was an old one. Lines are listed by typing 'list'. To exit from BASIC, just type '^b'. There is more information on BASIC that was written by Steven Segletes. To look at it type 'cat /bmd/steven/tbdfiles/basic.doc'. You could also refer to the 'BASIC' Language Manual', and basic(1).

## C. Programming in C

As we said, C is the language of choice: everything in UNIX is tuned to it. It is also a remarkably easy language to use once you get started. Sections II and III of the user's manual describe the system interfaces, that is, how you do I/O and similar functions.

You can write quite significant C programs with the level of I/O and system interface described in 'Programming in C: A Tutorial', if you use existing programs and pipes to help. For example, rather than learning how to open and close files you can (at least temporarily) write a program that reads from standard input, and use 'cat' to concatenate several files into it. This may not be adequate for the long run, but for the early stages it's just right.

There are a number of supporting programs that go with C. The C debugger, 'cdb', is marginally useful for digging through the dead bodies of C programs. 'adb', the assembly language debugger, is actually more useful most of the time, but you have to know more about the machine and system to use it well. I have found that the program 'lint', when applied to programs gives a thorough listing of possible errors in a program (and usually it's right!) The most effective debugging tool is still careful thought, coupled with judiciously placed print statements.

You can instrument C programs and thus find out where they spend their time and what parts are worth optimizing. Compile the routines with the '-p' option; after the test run use 'prof' to print an execution profile. The command 'time' will give you the gross run-time statistics of a program, but it's not super accurate or reproducible. See cc(I) and prof(I) for more on this.

C programs that don't depend too much on special features of UNIX can be moved to the Honeywell 6070, IBM 370, DEC VAX and many other computer systems with modest effort. Read 'The Portable C Library' by M.E. Lesk for details.

## D. At Your Service

If you want to use Fortran, you might consider 'RATFOR', which gives you the decent control structures and free-form input that characterize C, yet lets you write code that is still portable to other environments. Bear in mind that UNIX Fortran tends to produce programs that are larger and slower than their equivalents in C.

If you want to use assembly language, be aware that there are three assembly languages on UNIX: 'as', the standard UNIX assembler; 'asm', a macro assembler similar to MACRO-11; and 'macro', a macro assembler specifically geared to Fortran. However, for virtually all applications, C should be used instead of assembler. This implementation of C is essentially a very powerful shorthand for assembly language. Most statements compile into a very few instructions.

If your application requires you to translate a language into a set of actions or another language, you are in effect building a compiler, though probably a small one. In that case, you should be using the 'yacc' or 'tmg' compiler-compilers, which help you develop a compiler quickly.

If you are making a few changes to a large or 'sacred' program/database, and two (slightly different) copies would be undesirable, there is a solution! The programs 'mkpat' and 'patch' are the answer. 'Mkpat' takes two files and, after some fiddling, comes up with the editor commands that, if applied to the original file, will reconstitute the modified file. 'patch' accepts the original file and the file output by 'mkpat', and reconstitutes your modified file.

## BIBLIOGRAPHY

1. B. Henriksen, M. Hartwig, and S. Wolff, "Text Processing," The Ballistic Research Laboratory Office Of The Director Automated Office System, SECAD Internal Memorandum.

2. B. W. Kernighan, "A Tutorial Introduction to the UNIX Text Editor," UNIX Programmer's Manual for UNIX System III, Vol. II, Western Electric Company, Inc., Greensboro, 1981.

3. B. W. Kernighan, "UNIX for Beginners - Second Edition," UNIX Programmer's Manual for UNIX System III, Vol. II, Western Electric Company, Inc., Greensboro, 1981.

4. B. W. Kernighan, The C Programming Language, Prentice-Hall, Inc., Englewood Cliffs, 1978.

5. M. E. Lesk, "Typing Documents on the UNIX System: Using the -ms Macros with TROFF and NROFF," Bell Labs Internal Memorandum, Bell Laboratories, Murray Hill, 1978.

6. R. Thomas, and J. Yates, A User Guide to the UNIX System, Osborne/McGraw-Hill, Inc., Englewood Cliffs, 1978.

7. K. Thompson, and D. M. Ritchie, UNIX Programmer's Manual for UNIX System III, Vol. I, Western Electric Company, Inc., Greensboro, 1981.

8. Computer Technology Group, Telemedia, Inc., Class notes, 1982.

## APPENDIX A - EXAMPLES

## I. Login

Login refers to what you have to do to let the computer know that you wish to speak to it. This is a sample login, you must type those things that are in the normal type, the things that the computer types to you are in boldface.

```
BREAK                          (you don't need this if you have a dial-up
                                modem)

RETURN                         (may be repeated)
```

(The RETURN key must be pressed after every command as indicated by the 'RETURN' responses below)

```
enter class  bmd  RETURN
class bmd start   RETURN

JHU/UNIX 6.174    Ballistic Modeling Division   ttyN
11/70 1024kw mem, 102/150 procs, 29 users.  Unattended

User Name:  cider    RETURN     (you would type your user name)
Password:  nancyc    RETURN     (you would type your password)

11/22/82 10:56:06  Nancy Cider (TBD/PMB) <cider@bmd70>
Prev:  11/22 08:17 from ttyI

$
```

PRECEDING PAGE BLANK-NOT FILMED

## II. Password

This is an example of how to change your password. The only things that show up on the screen are "passwd" and the computer prompts, the rest is invisible to protect your new password. Your password must be at least 6 characters, and can be any combination of upper and lower case letters, plus digits.

```
$ passwd
cider:                        (Everything on the next 3 lines that you type
                               won't show on the terminal)
Old password:   nancyc  RETURN     (Type your password as it is)
New password:   TBDstuff  RETURN   (Type what you want as your new password)
Confirm:  TBDstuff RETURN          (Type your new password again)
$
```

## III. Permissions

Before we discuss how to change permissions, maybe we ought to discuss exactly what permissions are and how you are affected by them. If you make a listing of your files by executing the command

```
$ ls -l
```

you will get a result like the following

```
-rw-rw-rw-     1  cider    1345 Dec  1  10:35  emacs.test
-rw-------     1  cider       0 Feb 10  12:56  mailbox
-rw-------     1  cider   12786 Feb 10  12:56  mbox
```

the section that looks like '-rw-rw-rw-' is the permission section. You should notice that the permission section has three parts. The first 'rw-' is the permissions that you have, which are read and write. The second and third parts are for your group and others, respectively. Everyone has read and write permission for the first file, but my group and others have no permissions for my mailbox and mbox files.

There are default permissions used when you create a directory or a file. If you feel that these permissions are either too liberal or too stringent for your use, you can change them. Your directory and individual files can be changed by using the "chmod" command.

You can do

```
$ chmod 777 junkfile
```

and the result will be that everyone can read the file, write a new version of the file, delete the file, and execute the file (if appropriate). In files, 4 is the ability to read the file, 2 is the ability to write a new version of the file or delete it, and 1 is the ability to execute the file (used only for programs).

The same type of thing can be done to your directories. The permissions for your home directory can be changed by typing

```
$ chmod abc .
```

where the abc stands for the sum of the permissions you wish to allow for everyone. In directories, 4 is the ability to do an ls (list the filenames in a directory), 2 is the ability to put something into the directory, and 1 is the ability to take something from the directory, or delete a file from a directory. The abc refers to the separate person's permissions. The a refers to you, the b refers to your group members, and the c refers to everyone else. So, 777 would allow everyone to search your directory, write things into it, and read things from it. Using 766 allows you to list, write, and delete files in the directory, and other people to list the files in your directory and put new files into it, but not take something from it.

If you wish every file you create to have the same permissions, and you don't want to use the ones that are the default (644, or read and write for you, and read everyone else), you can put the command "umask abc" in your .profile that is executed each time you login. The "umask" subtracts the permissions listed for every file you create. The command "umask 026" says to subtract no permissions from yourself, subtract write permission from your group, and subtract read and write permission from all others, every time a file is created.

You may, on the other hand, want to create a "closed" directory for sensitive information as follows

```
$ mkdir CLOSED                (It is suggested that directories be
                                distinguished from files by capitalizing
                               directory names)

$ chmod 700 CLOSED
```

Then to store files in this directory, you move to it with the following command

```
$ cd CLOSED
```

Now you can edit and create files with the editor or emacs as you normally would. To change back to your original directory, just type 'cd' without any argument, and you will be back in your HOME directory.

## IV. Send

This is an example of how to send a message to someone else on the system via the mail facility.

```
$ send
SEND (22 Nov 82)
To:  barb                    (The user name of whomever you want to send mail
                              to)
cc:  RETURN                  (You can give names of users you wish to receive
                              a copy, or RETURN for no copies)
Subject:  Type whatever the message is about!
Type message; end with a CTRL-D...

Give the contents of the message, however many lines you wish, until
you have said everything you wish to say!
^D
Command or ?: f              (To include another file with your text)
File: barb.msg               (Type the name of the file to be included)
 ...included
Command or ?: e              (To edit the file)
234
1,$
Give the contents of the message, however many lines you wish, until
you have said everything you wish to say!
This is the contents of the file called "barb.file" that I created
for sending to barb in my message!  I just have a few lines of ths/that
for example.
$-1
for sending to barb in my message!  I just have a few lines of ths/that
s/ths/this/p
for sending to barb in my message!  I just have a few lines of this/that
w
235
q
Command or ?: s              (For send)
 barb@Brl-Bmd:  address ok
 Message posted.

$
```

## V. Msg

There are two examples here, containing different possibilities and options of the "msg" command. You can refer to the section on mail in part I of my tutorial.

A.

```
$ msg
MSG (22 Nov 82)  Type ? for help.
Reading /other/cider/mailbox  .. 2 messages total.
<- headers all
  1N       1251: 17 Nov 82  Myra Hartwig      Computer Maintenance
  2N        532: 19 Nov 82  Nancy Cider (TB   Your assistance
<- type 1
(Message # 1: 1251 bytes)
Date:     17 Nov 82 9:56:50-EST (Fri)
From:     Myra Hartwig  <myra@BRL-BMD>
To:       keller at Brl-Bmd, klem at Brl-Bmd, etc
cc:       myra at Brl-Bmd, hardware at Brl-Bmd
Subject:  Computer Maintenance
Folks,
        We are rapidly approaching the "drop dead" date for the
current computer maintenance contract, and the start-up of our
new contract.  ETC.
<- delete 2
<- headers all
  1   1251: 17 Nov 82  Myra Hartwig      Computer Maintenance
  2ND  532: 19 Nov 82  Nancy Cider (TB   Your assistance
(The "D" is the delete flag, to tell you that you wanted that message
deleted.  There are other flags that are also used, N for new, A for
answered, P for put into file, etc.)
<- forward 1
  1N       1251: 17 Nov 82  Myra Hartwig      Computer Maintenance
SEND (22 Nov 82)
Subject:  [Myra Hartwig:   Computer Maintenance]
To:  barb                      (The user name of the person to whom the message
                                is forwarded to)
Your comment; end with CTRL-D on a new line ...

I just wanted to give you a copy of this in case you
didn't recieve one.  Is there anything you wish to say in reply?
                               Nancy
^D
... included
command or ?: s
 barb@Brl-Bmd:  address ok
 Message posted.
<- exit and update /other/cider/mailbox into /other/cider/mbox
   [Confirm] yes               (Or RETURN)
Moving undeleted mail to /other/cider/mbox
$
```

B.

```
$ msg mbox                    ("mbox" is where your undeleted mail goes when
                               you exit)
MSG (22 Nov 82)  Type ? for help.
Reading mbox  .... 4 messages total.
<- headers all
  1   565: 9 Nov 82  Mike Muuss       Experience
  2   231:12 Nov 82  Barbara Ringers  Your apprenticeship
  3  1251:17 Nov 82  Myra Hartwig     Computer Maintenance
  4   232:21 Nov 82  Nancy Cider      Sample
<- answer 3
  1 A 1251:17 Nov 82  Myra Hartwig     Computer Maintenance
copies to which original addresses:  cc'd
(You have a choice of "c" for copy the header off the original letter, or
RETURN for no copy.  I'm afraid that if you want different header info.,
you have to RETURN, then do a header edit to change it.)
To:    Myra Hartwig <myra@brl-bmd>
cc:    myra at Brl-Bmd, hardware at Brl-Bmd
Subject:  Re:  Computer Maintenance
Type message; end with CTRL-D...

This is just a little message to show you what's going on!
^D
Command or ?: h              (For header edit)
[Return (skip) - (delete) + (append)]
To: Myra Hartwig <myra@brl-bmd>
To: RETURN
cc: myra at Brl-Bmd, hardware at Brl-Bmd
cc: cider
Subject:   Re:  Computer Maintenance
Subject: RETURN
Command or ?: r              (For review)
From:     Nancy Cider (TBD/PMB)  <cider@BRL-BMD>
To:       Myra Hartwig <myra@brl-bmd>
cc:       cider at Brl-Bmd
Subject:  Re:  Computer Maintenance

This is just a little message to show you what's going on!
Command or ?: s              (For send)
   myra at Brl-Bmd:  address ok
   cider at Brl-Bmd:  address ok
<- quit
$
```

## VI. Emacs

There is a file in /bmd/cider called "emacs.test". Use the following commands on it to test out what emacs can do. You must do everything in the order it is listed here, or odd results will occur.

A. Type "cp /bmd/cider/emacs.test emacs.test" to get a copy in your directory. DO NOT USE "mv" or "ln"! This will remove the file from use by others.

B. Type "emacs emacs.test". This puts you in emacs, editing "emacs.test".

C. Before you begin typing, you need to know something about the characters you will be using. There are 2 types, control and meta-characters. By convention, the control characters are represented by the "^", followed by a character. For example, the "control d" character would look like "^d", and it is typed by holding down the CTRL key, and pressing the "d". Meta-characters are represented by a capital M followed by a character. For example, "meta d" is represented by "M-d", and is typed by pressing the ESC or ESCAPE key, (which produces "M-: " at the bottom of the screen) and then the "d". Another different feature is the "multiplier", which allows you to get multiples of a command. You use the ESCAPE key followed by how many times you wish to perform the command, then the command you wish to perform. This means that if you want to move down 16 lines, you can type "M-16", followed by "^n".

D. The first thing in the file to be done is that of deleting my beginning message. This can be done by typing "M-6", then "^k". The "^k" kills from the current cursor position to the end of the line. The cursor should be positioned at the beginning of the text.

E. The first typo to correct is the word "publcty" in the first line. Two "i"'s must be added in the appropriate positions to make the word "publicity". To move over there, type "M-3", then "M-f", which moves you forward 3 words, then type "M-5", and "^f", which moves you forward 5 characters. The cursor should now be positioned on the "c" and if so, you can type your first "i". Type a "^f" to move you forward, and type your second "i". The rule for inserting text is to position the cursor on the character after the position where the text is to be inserted and just type.

F. The second typo is 6 lines down in the word "recieving". To get there, type "M-6", then "^n". The cursor should now be in the right line, but the wrong place, so you'll need to back up. Type "M-2", "M-b" to back up 2 words, then "M-7" and "^b" to back up so that the cursor is positioned on the first "i" in "recieving". Type "^t" and you should notice that the "i" and "e" have been transposed. That is the function of "^t".

G. The last typo is "meckanisms" another 12 lines down. Type "M-12" and "^n", then use "^b" to back up so that the cursor is on the "k", then type "^d". The "^d" deletes forward one position, meaning that the cursor is the place where the deletion is to occur, then the cursor stays where it is. There is also a "^h" that deletes backward. Position the cursor one character in front of the one you want to delete, then type "^h" and the character in back of the cursor is deleted, while the cursor backs up to take that place. Now that the "k" is deleted, all you need to do to fix the word is type the "h".

H. Since we're through with the typos, why don't we save what we've done in case the computer goes down? The command to do this is "^x", "^s". As you type the "^x", notice that the cursor moves to the bottom of the screen, and is preceded by a "^X:". After the "^s" is typed, you should get a message at the bottom of the screen saying "Wait" then "Written: emacs.test", which says that the buffer we've been working with has been saved.

I. There was supposed to be a new paragraph at the line that begins "The ozone layer ranges ...". To correct this use "M-11" and "^p", then type "^a" to get to the beginning of the line. Type "RETURN" followed by 3 spaces, and it should look just like the other paragraphs.

J. The last thing to be done is use the search to find and correct an oddity that you might find a useful application. Go to the beginning of the file by typing "M-<". Now to search type "^s", and you should see "Search:" at the bottom of your screen. Type a "-" and the cursor should jump to the dash in between "microcomputer-based". Type "RETURN" and the "Search:" string should be "Search: -^J", which is the control symbol for RETURN. The cursor should jump to the dash in "strato-", and the screen should rewrite with line 21 as the first line on the screen.

Some packages like 'nroff' don't allow hyphenation, because 'nroff' hyphenates as it sees fit. So I'm going to show you a way to join the words without retyping them. Type a "^d" to get rid of the dash, as well as ending the search. Type another "^d" to get rid of the *end-of-line character* and join the word. Go to the end of the word with a "M-f", and type "^d" to get rid of the space between words. Type a "RETURN" and you now have 2 lines with "stratospheric" joined and positioned on the upper line.

K. This is the end of the emacs example, so let's exit and save the file. Type "^x", "^c" and a message should appear at the bottom of the screen saying "buffer emacs.test modified since last write to file emacs.test, write?". You can say "y" if you want the file written (this is what you should type), "n" if you don't want the file written, and "^g" if you want to abort the exit and write command and continue to edit. After the file has been written, emacs sometimes doesn't clear the screen and give you back a cursor, but don't panic! Type 2 letters of a command (I usually use "^q" twice) and the screen will come back with a cursor, and the command you've typed.

L. There is more to emacs than just these few commands I've taught you. The best way to learn is to create a file, and just experiment using the list of commands I've included in this handout. And, if you ever get stuck doing something you don't want to do, just type "^g", it usually quits anything you've started. Or, if the file is wrecked beyond repair, you can read the file in again using "^x", "^r" and start everything over.

# VII. Creating Emacs Files

## A. .Profile

These are some of the options you can put in your .profile. The "path" and "mail" lines are pretty standard, but you'll have to change "cider" to your user name. In the "term" line you will have to change the terminal type to that of your terminal, and the "stty" will probably have a different page length. If you don't know your terminal type, give me a call and we can work one out. The "stty" command has many more options that are listed in stty(1). If you don't wish to use the reminder command, you can leave off the last two lines.

To actually enter the file, type "emacs". The emacs screen will come up, and you can begin to enter text. Enter the whole portion, then use any emacs commands you need if you make a mistake. Then to save the file, type "^X ^C", and the words "Main modified since last write to Main, write file?" will appear at the bottom of the screen. You reply with "y" for yes, and the words "Write file?" will appear at the bottom of the screen. You reply with ".profile" which will be the file name.

```
PATH=:/bin:/usr/bin:/usr/sbin:/usr/ofcbin:/usr/3bin
MAIL=/other/cider/mailbox
TERM=adm3a
export PATH MAIL TERM
umask 026
stty page23
grep 'date %m/%d/%y' .dates
reminder
```

## B. .Dates

These dates are used by "reminder" (from your .profile). The first section is the date of the event, the second section is the number of days in advance you wish to be reminded, and the third section is the content of the reminder. The fourth section is who the event is about or from. The last section is the status field, which is usually "n" or "x". An "n" just reports the event, and the "x" deletes the event after it has passed. You can create this file in the same manner as your .profile, but when "Write file?" appears, you reply ".dates".

```
12-25:365:Christmas::n
12-22:4:Checking about a bumper sticker!::nx
1-2:14:Birthday:Mom:n
```

When you logged on for 21 December, the output would look like

```
4 days before Christmas
One day before checking about a bumper sticker!
12 days before Mom's Birthday
```

## C. <u>.Plan</u>

Your .plan is used when someone does a "finger" on you. It can be whatever information you wish to give anyone trying to get a hold of you, or general information you wish people to know about you. All you need to do is create a file called ".plan" and fill it with whatever text you wish given out. This can be created as above, only you must use the name .plan.

## VIII. Creating Files With "ed"

This is a sample file that you can create yourself using "ed".

```
$ ed junkfile
?                             (This means that the file did not previously
                               exist)
a                             (This means to append to the text existing in
                               the file)
Despite all the publcty about damage being done to
the ozone layer, scientists have had little information
with which to judge man's impact on this part of the
earth's atmosphere.  A microcomputer-based satellite
designed and operated by the University of Colorado is
now examining the ozone layer, sending data and
recieving instructions through a minicomputer.
.                             (end the apending)
1                             (move to line 1)
Despite all the publcty about damage being done to
s/lcty/licity/p               (substitute the first occurrence of the string
                               "lcty" with "licity")
Despite all the publicity about damage being done to
+6                            (move down 6 lines)
recieving instructions through a minicomputer.
s/ie/ei/p                     (substitute the first occurrence of "ie" with
                               "ei")
receiving instructions through a minicomputer.
w                             (write out a copy of the file "junkfile")
771                           (number of characters in the file)
q                             (quit the editor)
$
```

## IX. Logout

You have your choice of doing a "^D", or a "logout". The "logout" is a longer version that gives you some accounting information as well as logging you out. It looks like:

```
$ logout
Checking... OK (375/1000 blocks)
Disk reads     303      writes     64
Connected  0:03:18     to date  51:58:05
CPU time   0:00:12.86  to date   2:39:23.65
ttyd free Mon Nov 22 14:52:13 1982
```

The ^D is a quick exit, and just checks to see if you are under your storage quota, then it exits as follows:

```
$ ^D
Checking... OK (375/1000 blocks)
ttyd free Mon Nov 22 14:52:13 1982
```

## X. Nroff

This is the text of the first section of this manual as it looked before I put "nroff" to work on it. I've enclosed some comments,to make things clearer, but if you have any questions, refer to the "Programmer's Manual" or see me for additional documentation manuals.

```
.RP                         (tells nroff you have a released paper style)
.nr LL 7i                   |\   these two commands tell nroff to set the
.ll 7i                      |/   line length at 7 inches
.ds CH -%-                  (Puts page numbers like "-1-" at the top of the
                             page)
.ce                         (Centers a line, or multiple lines if specified)
I.  INTRODUCTION
.PP                         (Start of paper)
In many ways, UNIX represents the state of the art in computer operating
systems.
From the user's point of view, it is easy to learn and use,
and its tool-oriented structure makes it very productive.
.PP
It is hard, however, for the beginner to know where to start,
and how to make the best use of the facilities available.
The purpose of this tutorial is to point out high spots for new users,
so they can get used to the main ideas of UNIX and start making good use of
it quickly.
It is not an attempt to rewrite the UNIX Programmer's Manual;
instead it should suggest in what order to read the manual,
and collect together things that are stated only indirectly there.
Throughout, references to the manual will look like this:  see msg(1);
which means "see the article titled "msg" in section 1 of the
Programmer's Manual".
If you don't have a copy of the Programmer's Manual, there is one on-line.
You give the command 'man msg' to get information to see how 'man' works.
.PP
There is also an on-line program called "learn" you can use to help
you get started.
It is a self-paced course on various topics.
It may be worth your while to spend some time going through this course as an
introduction to the system.
.PP
There are five sections to this manual:
.IP II.                     (Indented paragraph set off by "II.")
.ul Getting Started         (Underline the following words)
:  How to type to UNIX, using simple commands, and how to log in and out.
.IP III.
.ul Day-to-day Use
:  Things you need every day to use UNIX effectively;
generally useful commands and the file system.
```

```
.IP IV.
.ul Document Preparation
:  Preparing manuscripts is one of the most common uses for UNIX.
This section contains an introduction to the simplest text processing
program, and some information about more sophisticated software.
.IP V.
.ul Writing Programs
:  UNIX is an excellent system for program development.
This section talks about some of the tools, but again is not a tutorial in
any of the programming languages that UNIX provides.
.sp 2
.ce
II. GETTING STARTED
.sp
A.
.ul
Typing to UNIX
.PP
Typing on a computer terminal is similar to using a typewriter,
with the exception that some keys have special meanings to the computer.
You should become familiar with these keys because some are essential and
some may cause unexpected things to happen.
Typing characteristics can be changed using the 'stty' command to suit
various kinds of terminals and user preferences.
See stty(1) for more information.
```

# XI. DF Forms

This is a sample of a df form that you can use. It is written for nroff and uses similar commands. A full explanation is contained in the text processing handout that is included. To run the file type "runoff file-name".

NOTE: The output will have the correct spacing for a standard DF form.

```
.CS 1                         (tells the package that you're on comment 1, the
                               package can handle up to 9 comments)
.OS T                         (prints the DRDAR-BLT flag, can be changed for
                               other office symbols, by changing the "T")
.SB                           (subject)
Emergency Justification
.TO                           (to whom)
Ms. Sara Bauer
Procurement Directorate
.FM                           (from whom)
Chief, TBD
.AU                           (author and phone number)
BERingers/6065
.ND 9 December 1982           (new date, or no date if left blank)
.SG                           (signature block)
DONALD F. MENNE
.P1                           (first level of paragraphing)
The Terminal Ballistics Division submitted orders W3Q4AA - 2152-0007,
2152-0009, 2152-0005, 2175-0006, 2152-0008, 2152-0006, 2152-0012, 2175-0007
for 72 Modgraph terminals.
No shipping date has yet been received for the delivery of these terminals.
.P1
Two VAX 11/780 Computer Systems and peripherals have been ordered for TBD and
have a delivery date of January, 1983.
Such systems are useless without accompanying terminals to enable TBD
personnel to access the systems.
.P1
It is thereby requested that emergency measures be taken to acquire the
aforementioned terminals ASAP.
Otherwise equipment involving considerable investment will remain idle.
.GO                           (last command in the file)
```

# DISPOSITION FORM

For use of this form, see AR 340-15; the proponent agency is TAGO.

| | |
|---|---|
| REFERENCE OR OFFICE SYMBOL<br>DRDAR-BLT | SUBJECT<br>Emergency Justification |

TO Procurement Directorate
Ms. Sara Bauer

FROM Chief, TBD

DATE 9 December 1982 CMT 1
BERingers/6065

1. The Terminal Ballistics Division submitted orders W3Q4AA - 2152-0007, 2152-0009, 2152-0005, 2175-0006, 2152-0008, 2152-0006, 2152-0012, 2175-0007 for Modgraph terminals. No shipping date has yet been received for the delivery of these terminals.

2. Two VAX 11/780 Computer Systems and peripherals have been ordered for TBD and have a delivery date of January 1983. Such systems are useless without accomanying terminals to enable TBD personnel to access the systems.

3. It is thereby requested that emergency measures be taken to acquire the aforementioned terminals ASAP. Otherwise equipment involving considerable investment will remain idle.

DONALD F. MENNE

DA FORM AUG 80 2496 PREVIOUS EDITIONS WILL BE USED ☆U.S. GOVERNMENT PRINTING OFFICE: 1982-372-711

## XII. Travel Orders

There are a few programs that have been combined to make writing travel orders easier. First issue the next three commands

```
$ ln /other/steven/tbdfiles/to.copy
$ ln /other/steven/tbdfiles/person.copy
$ ln /other/steven/tbdfiles/place.copy
```

You don't need to alter these files in any way (And indeed you should not, because then the travel order programs will not work for anyone!). To begin writing a travel order type the command

```
$ person.copy
```

You should begin getting the prompts that follow, and you would replace my answers with those that you need for your order.

```
What personfile do you wish to create ?
Your response will become a UNIX file in your directory.
cider
Answer all questions on one line, as they will appear on the
travel orders...(any questions, call Steve Segletes X6073)

Enter person's name (last name, first name init.)
CIDER, NANCY L.
Enter the person's home address, town, state
8150 Pleasant Plains Road, Towson, MD
Enter the person's position, grade
Computer Scientist, GS-07
Enter the person's phone extension
6066
Enter the requesting official, title
C. T. CANDLAND, A/C, PMB, TBD
Enter the approving official, title
D. P. MENNE, C, TBD
Enter Organizational Element
PMB
Enter c for civilian, m for military
c


Due to regulations, the person's Social Security number will have to
be typed manually on the travel orders since it can not be entered
on the computer.

Your personfile cider is now created
Have a nice day.
```

You should now have a file in your directory of the form that follows. This is your personfile.

```
.ds NM "CIDER, NANCY L.
.ds AD "8150 Pleasant Plains Road, Towson, MD
.ds PG "Computer Scientist, GS-07
.ds SS "
.ds PH "6355
.ds RO "C. T. CANDLAND, A/C, PMB, TBD
.ds AO "D. P. MENNE, C, TBD
.ds OE "PMB
.ds OA c
```

To fill out the information that is specific to each travel order (eg., place, dates, cost, etc.) you then type

```
$ place.copy
```

The program will prompt you as the person program did, for the following information

```
Enter file name:
cider.chicago

End each entry with a period on a line by itself



Purpose of Trip:
To attend training on "C" Language Programming.
.

Security Clearance (type N if none):
N
.

No. of days TDY:
seven (7) days
.

Proceed on:
12 Dec 82
.

Itinerary:
Depart APG, MD to Chicago, IL (Telemedia, Inc.) and return.
.

Mode of Transportation:  (CR, CA, CB, CS, GA, GV, GS, or P)
CA
.
```

**Advance Authorized:**
$475.00
.

**Remarks:**
Trip report required.
GSA motor pool/contract vehicle authorized as determined by APG Transportation Offier for use in and around/to and from TDY station. Authorize use of rental car/POV to and from BWI. POV is equipped with required seatbelts.

N. L. CIDER

.

**Work Order Number:**
XXXX-XX-XXX RG2
.

You should now have a second file called cider.chicago, or whatever you've called the file, in your directory. It should be of the form

```
.PR
To attend training on "C" Language Programming.
.SC N
.DA seven (7) days
.PC 12 Dec 82
.IT
Depart APG, MD to Chicago IL (Telemedia, Inc.) and return.
.MT CA
.AV $475.00
.RM
Trip report required.
GSA motor pool/contract vehicle authorized as determined by APG
Transportation Officer for use in and around/to and from TDY station.
Authorize use of rental car/POV to and from BWI. POV is equipped with
required seatbelts.

                                                 ____________________
                                                    N. L. CIDER
.WO
XXXX-XX-XXX RGX
.GO
```

To actually type out the travel order, type

```
$ to.copy cider cider.chicago
```

but, before you press RETURN, put the form for the order in the printer and line it up, then turn the printer on. Make sure the margin is lined up properly, or you may wind up going by bus instead of flying.

**REQUEST AND AUTHORIZATION FOR TDY TRAVEL OF DOD PERSONNEL**
*(Reference: Joint Travel Regulations)*
Travel Authorized as Indicated in Items 2 through 21.

1. DATE OF REQUEST
25 Jan 1983

**REQUEST FOR OFFICIAL TRAVEL**

| | |
|---|---|
| 2. NAME *(Last, First, Middle Initial)*<br>CIDER, NANCY L. SSN:<br>8150 Pleasant Plains Road, Towson, MD | 3. POSITION TITLE AND GRADE OR RATING<br>Computer Scientist, GS-07 |

| | | |
|---|---|---|
| 4. OFFICIAL STATION<br>USABRL, ARRADCOM | 5. ORGANIZATIONAL ELEMENT<br>PMB | 6. PHONE NO.<br>6066 |

| | | |
|---|---|---|
| 7. TYPE OF ORDERS<br>TDY | 8. SECURITY CLEARANCE | 9. PURPOSE OF TDY<br>To attend training on "C" Language Programming |
| 10a APPROX NO. OF DAYS OF TDY *(Including travel time)*<br>seven (7) days | b PR FED O A *(Date)*<br>12 Dec 82 | |

11. ITINERARY [✓ XX] VARIATION AUTHORIZED

Depart APG, MD to Chicago IL (Telemedia, Inc.) and return.

12. MODE OF TRANSPORTATION

| COMMERCIAL | | | | GOVERNMENT | | | PRIVATELY OWNED CONVEYANCE *(Check one)* |
|---|---|---|---|---|---|---|---|
| RAIL | AIR | BUS | SHIP | AIR | VEHICLE | SHIP | RATE PER MILE |
| | X | | | | | | ☐ MORE ADVANTAGEOUS TO GOVERNMENT |

☐ AS DETERMINED BY APPROPRIATE TRANSPORTATION OFFICER *(Overseas Travel only)*

☐ MILEAGE REIMBURSEMENT AND PER DIEM LIMITED TO CONSTRUCTIVE COST OF COMMON CARRIER TRANSPORTATION & RELATED PER DIEM AS DETERMINED IN JTR. TRAVEL TIME LIMITED AS INDICATED IN JTR.

13. [✓ X] PER DIEM AUTHORIZED IN ACCORDANCE WITH JTR.
☐ OTHER RATE OF PER DIEM *(Specify)*

| 14. ESTIMATED COST | | | | 15. ADVANCE AUTHORIZED |
|---|---|---|---|---|
| PER DIEM | TRAVEL | OTHER | TOTAL | |
| $ | $ | $ | $ | $ $475.00 |

16. REMARKS *(Use this space for special requirements, leave, superior or 1st-class accommodations, excess baggage, registration fees, etc.)*

Trip report required. GSA motor pool/contract vehicle authorized as determined by APG Transportation Officer for use in and around/to and from TDY station. Authorize use of rental car/POV to and from BWI. POV is equipped with required seatbelts.

N. L. CIDER

| | |
|---|---|
| 17. REQUESTING OFFICIAL *(Title and signature)*<br>C. T. CANDLAND, A/C, PMB, TBD | 18. APPROVING OFFICIAL *(Title and signature)*<br>D. F. MENNE, C, TBD |

**AUTHORIZATION**

19. ACCOUNTING CITATION
21X4992.06KA 6K S28017 EOE 2119
XXXX-XX-XXX RG2

| | |
|---|---|
| 20. ORDER AUTHORIZING OFFICIAL *(Title and signature)* OR AUTHENTICATION<br>PATRICIA C. ROBERTS, DAC, Chief, PMRD<br>USA BALLISTIC RESEARCH LABORATORY, APG, MD 21005 | 21. DATE ISSUED<br><br>22. TRAVEL ORDER NUMBER |

DD FORM 1 JUN 67 1610

## XIII. Subscripts and Superscripts

Steve Segletes has written a macro package for nroff that takes care of superscripts and subscripts as well as scientific formulas. It will tell a Diablo printer, or something similar, to move the carriage up and down. To run the package you need to execute the following command

```
$ cp /bmd/steven/tbdfiles/macros macros
```

then to actually get the output, type the following

```
$ nroff -ms -T450 macros filename
```

or for a twelve-pitch wheel (elite type)

```
$ nroff -ms -T450-12 macros filename
```

A sample using some of the commands follows.

```
.PP
The Temperature-Humidity Index, or THI, is calculated on the basis
of the following formula:
.sp
.SB 0.4(T d \ +\ T w )\ +\ 15
.sp
In this formula, the symbol
.SB T d \ represents
the temperature of a dry-bulb thermometer, and
.SB T w \ represents
the temperature of a wet-bulb thermometer.
.PP
Since 1 kilocalorie is equal to 4.2 x
.SP 10 3 \ joules,
the value of the specific heat of a substance in SI units is simply 4.2 x
.SP 10 3 \ times
its value in MKS units.
.PP
Thus the specific heat of gold is 4.2 x
.SP 10 3 \ x
0.031 or 1.3 x
.SP 10 2 \ joules
per kilogram-kelvin.
```

The reason for using a backslash followed by a space in most of the examples, is that each argument is separated by spaces. Therefore, if you want a space to show up in the text, you need to quote it with a backslash. If you were to enter the above text and "nroff" it in the manner described above, you would wind up with a paper in the form on the following page.

Temperature-Humidity Index, or THI, is calculated on the basis of the following formula:

$$0.4(T_d + T_w) + 15$$

In this formula, the symbol $T_d$ represents the temperature of a dry-bulb thermometer, and $T_w$ represents the temperature of a wet-bulb thermometer.

Since 1 kilocalorie is equal to $4.2 \times 10^3$ joules, the value of the specific heat of a substance in SI units is simply $4.2 \times 10^3$ times its value in MKS units. Thus the specific heat of gold is $4.2 \times 10^3 \times 0.031$ or $1.3 \times 10^2$ joules per kilogram-kelvin.

## APPENDIX B - A System Summary for Various Commands

## I. FILES

A file is a named collection of information. The collection might be, for instance, an essay, an executable program, or merely raw data.

### A. Manipulating Files

1. chmod. Used to change the mode (permissions) of a file. The mode consists of the 'or' of permissions in three fields. The permission for read is 4, write is 2, execute is 1, and no permission is 0. The three fields are user (owner), group, and other. In the example below the user has read, write and execute permissions for the 'junk' file, his group has read permission, and others have no permissions.

```
$ chmod 740 junk
```

2. rm. Deletes the specified file, in this case called 'junk'.

```
$ rm junk
```

3. del. Deletes the file 'junk', after again prompting for the deletion. Gives you a minute to make sure tha`t you really want that file deleted.

```
$ del junk
```

4. dsw. Prompts for the selective deletion of files in a directory. Answer with 'y' for yes, RETURN for no, and 'x' for exit.

```
$ dsw
junk RETURN
temp y
stuff RETURN
$
```

5. mv. Moves or renames a file. Here, the old file is called 'junk' and the new file is called 'newjunk'.

```
$ mv junk newjunk
```

6. mvall. Moves multiple files into a target directory. Here, the files to be moved are any files beginning with a 't', and the target directory is '/bmd/cider/temp'.

```
$ mvall t* /bmd/cider/temp
```

7. cp. Makes a copy of a file. The old file is 'junk' the copy of the file is 'newjunk'.

```
$ cp junk newjunk
```

8. cpall. Copies all the files specified to target directory keeping their original names. The file to be moved are those whose names begin with 't' and the new directory is '/bmd/cider/temp'.

```
$ cpall t* /bmd/cider/temp
```

9. ls. Gives the names of all files in your directory. Lists in time order when invoked with the -t argument; adds permissions and other information when invoked with -l; includes filenames beginning with a '.' when invoked with -al.

```
$ ls -lt
```

10. lss. Gives a multiple-column listing of your files. Helpful when listing a lot of files.

```
$ lss
```

11. ln. Makes a link from your directory into the directory where the file exists. This is only a directory entry referring to a file. The file is not actually duplicated, but retains its original ownership and permissions. This means that everyone having a link to the file can change it, and will receive the new version if anyone else changes it. However, you can edit or use it the way you would any other file.

```
$ ln /bmd/noone/jur':
```

12. ar. Archives or stores a file as part of a larger file, while allowing it to be retrieved individually. It is mainly used to make library files to be used by the loader. See ar(1) for more information.

## B. Special Files

1. .profile. Is the file that contains special commands that will be executed each time you login to the computer.

2. .dates. If you wish to use the reminder command in your '.profile', then you will need a file called '.dates'. This file is used store those dates that you wish to be reminded about.

3. .plan. Is used in conjunction with the 'finger' command. As 'finger' is executed, it looks for this file to print out as your plan of action, or that information that you wish to give others looking for you.

## C. Working Within Files

1. ed, or e. The UNIX text editor. 'e' is used to input and edit text, data, and programs.

```
$ e junk
```

2. emacs. Is the UNIX screen editor. Edits text, but is used to edit a screen full of text, not a single line.

```
$ emacs junk
```

3. grep. Searches through a file for a given sequence of characters and prints out any line(s) and the file(s) where they occur.

```
$ grep 'the' junk*
```

D. Comparing Files

1. cmp. Indicates the first byte where two files differ; indicates all differing bytes when invoked with the -l argument.

```
$ cmp junk diffjunk
```

2. diff. Prints out the changes necessary to transform one text file into another. Used when you changed a file, then decided to change it back. It gives 'ed' editor commands as output, which can be used on 'junk' to get 'newjunk'.

```
$ diff junk diffjunk
```

3. comm. Indicates which lines are common to two files. It is a three column listing, the first column contains lines from 'junk', the second contains lines from 'diffjunk', and the third is lines contained in both 'junk' and 'diffjunk'.

```
$ comm junk diffjunk
```

4. mkpat. Produces a patch file which can be used to convert 'junk' into 'diffjunk'. It produces a file with the new filename plus '.patx', where 'x' is a letter from 'a' to 'z'. This allows you to make 26 patch files. This 'mkpat' will produce a file called 'diffjunk.pata'.

```
$ mkpat junk diffjunk
```

5. patch. Uses the patch file to convert the first file into the second. The output goes into a file with the old filename plus 'vxn', where 'x' matches the 'x' from the patch file, and 'n' is a number between '0' and '9'. The number is the version number of the reconstitution. For example, the command below would result in a file with the name 'junk.va1'.

```
$ patch junk diffjunk.pata
```

## II. DIRECTORIES

UNIX maintains its file system in an upside-down tree structure of groups of files called directories.

1. mkdir. Makes a directory, using the name you give it. Many people like to give their directories names beginning with a period, because then

they aren't listed using the normal 'ls' command; 'ls -al' must be used. Some people also use all capital letters to denote directories, so they do show up on the normal listing.

```
$ mkdir .new
```

2. cd. Moves you to another directory, in this case called .new.

```
$ cd .new
```

3. pwd. Tells you where in the file system you are. The example below would give a result of '/bmd/cider/.new', if used after 'cd .new'.

```
$ pwd
```

4. mvdir. Moves a directory to another place in the tree, which has the effect of renaming the directory.

```
$ mvdir /bmd/cider/.new /bmd/cider/.temp
```

5. rmdir. Eliminates the directory. This must only be done after all the files have been removed from the directory.

```
$ rmdir /bmd/cider/.temp
```

## III. TEXT OUTPUT

### A. On the Terminal

1. type. Outputs the file to the terminal, stripping certain control characters and terminating on receipt of a null character.

2. cat. Concatenates and outputs the file (or files) but does not strip control characters or do any similar processing.

3. pr. Produces numbered and dated pages, optionally with userspecified columnation, line widths, page lengths, indentations, and headers.

### B. On the Lineprinter

1. lpr. Prints files on the line printer.

2. dlpr. Operates similarly to 'lpr', but does not queue the output.

3. makevg. Print vugraph on Versatec printer with BRL and ARRADCOM logos printed at the top, and the date at the bottom. Use -w for a wide vugraph and -l for a long vugraph.

## IV. Storage Media

### A. Magnetic Tapes

1. tar. Saves and restores files on magnetic tape. You may append, read, list the filenames on the tape, update, or create a new tape.

2. assign. Either lists the assigned devices or assigns one to you.

3. dump. Copies to magnetic tape all files in the file system that have been changed after a certain date.

### B. Disks

1. mount. Is used to tell the system at what point in the UNIX tree to insert the disk's file system. Also used to determine what devices are already mounted, and where in the file system they reside.

2. umount. Is the inverse of 'mount', and removes the disk from the file system tree.

3. du. Summarizes disk usage for the present directory, or a directory you specify.

4. ustat. Displays the amount of free space and some other characteristics of mounted file systems.

## V. PROGRAMMING

### A. Languages

UNIX is equipped with compilers, interpreters, and front end processors for many languages. The following are presently available:

1. Apl. A sophisticated algorithmic programming language.

2. As. PDP 11/70 assembler.

3. Asm. DEC MACRO-11 assembler converted to run in UNIX.

4. Basic. A simplified beginner's programming language.

5. Bc. An arbitrary-precision arithmetic language that is a preprocessor for 'dc' (desk calculator).

6. C. The language that UNIX is written in, very fast, but you need some programming background to tackle this one well.

7. Fortran. A scientific programming language. Both Fortran 4 and 5 are available.

8. Ingres. A data base management system created for UNIX.

9. Lisp. A sophisticated language used mainly in artificial intelligence work, very good for english language processing.

10. M4. A macro preprocessor for RATFOR, C, and other languages.

11. Pascal. A language similar to C, very structured.

12. Ratfor. A dialect of FORTRAN (rational FORTRAN) that provides control flow structures missing in normal FORTRAN.

13. Yacc. (yet another compiler-compiler) converts a context-free grammar into a set of tables to be processed.

## B. Programming Tools

The following are helpful in the debugging and optimizing of programs.

1. adb. An advanced and very powerful debugger, but it is very difficult to learn.

2. cdb. A debugger primarily for programs written in C.

3. db. A universal debugger for examining object files.

4. ld. Links object modules to produce an executable program, resolves external references, and searches system libraries for undefined subroutines.

5. lint. Gets the 'lint' out of C programs. Checks for statements which are impossible to reach, data-type mismatches, subroutine calls whose returned values are never used, inconsistent use of arguments, unused or uninitialized variables, portability difficulties, and performs many other helpful checks.

6. nm. Prints the symbol table of an executable program (assuming it has not been removed).

7. od. Dumps the file specified in one of many modes.

8. prof. An optimizing tool which presents an operating profile of a program previously run. The program must have been compiled with the profiling option.

9. size. Gives the amount of core required to load and run a program; broken down into text, data, and bss categories.

10. strip. Removes the symbol table and relocation bits from an assembled or loaded program.

11. time. Prints out the execution times for a program.

## VI. SYSTEM FEATURES

### A. User and Terminal Status

1. tty. Tells you which terminal you are on.

2. uid. Indicates your user identification number.

3. money. Prints your CPU and connect time usage and quotas, terminal authorizations, and disk storage quota.

4. quota. Tests to see if you are under your disk storage quota.

5. passwd. Allows you to change your login password.

6. stty. Sets up the mode (speed, parity, etc.) which the computer uses to communicate with your terminal.

7. ksrset. Sets up the mode of a terminal port for use with the KSR-47.

8. admset. Sets the mode of a terminal port to that suitable for use with an ADM scope terminal at 9600 baud.

9. logout. Logs you out (if you are under your disk storage quota) and summarizes your CPU and connect time usage for the session.

10. ^D. Logs you out without summarizing your time usages.

### B. Communicating With Other Users

1. who. Provides a list of what accounts are in use. To find out the full name of each user, use the -n option. To find out the location of each user, use the -w option. To find out the full name of a specified user, use 'who is'.

2. write. Allows you to talk interactively with other users who are logged in.

3. mesg. Enables or disables other's ability to write to you.

4. finger. Prints information on person specified, including last login time, mailbox status, full name and home directory. May also print info from the person's .plan file.

5. nchk. Check for news.

6. news. Print new news articles or send news articles.

7. send. Send mail to another user or list of users. Computer will prompt for information.

C. Mail Facility

1. msg. Allows you to leave messages for someone. You can read out of 'mailbox' by default, or out of 'mbox' by saying 'msg mbox'. A list of possible commands follow (just type the first letter of each, plus the number, if required. If you leave out the number, 'msg' will assume you are referring to the current message.).

a. answer #. Calls 'send' with the 'To:' and 'Subject:' already filled in. The user is prompted for 'cc:' and the body of the message.

b. backup. Types previous message.

c. current. Prints current message number, number of messages, and the current mail file.

d. delete #. Sets a flag to indicate message will be deleted when an overwrite or exit command is given.

e. exit (exit and update). Removes deleted messages, moves undeleted messages to mbox, then quits.

f. forward #. Calls 'send' with the draft body of the named messages and the 'Subject:' already filled in.

g. goto #. Specified message becomes the current message.

h. headers #. Prints a one line header for each specified message.

i. jump (jump into lower fork running). Allows user to give a shell command, for example, who or date.

j. list #. Copy messages to file in preparation for printing with an optional page separator between each message. 'Msg' will prompt for file name.

k. move #. Same as put, followed by delete.

l. next. Types the next message in the file.

m. overwrite. Actually removes messages marked for deletion; then re-reads the file.

n. put #. Copies messages specified to a file. 'Msg' prompts for file name.

o. quit. Exit without changing any files or updating 'mailbox' to 'mbox' (will not perform deletions).

p. read. Read new message file. 'Msg' will prompt for file name. Default is mail.

q. send. Call 'send' program. (See the next section for details on send.)

r. type #. Types specified message or messages.

s. undelete #. Removes delete marking from messages, so they will not be removed.

t. :. Current date and time.

u. ?. Displays a list of valid commands

2. Send. This program is used for the actual sending of mail. It also allows you to edit your message, input more text, review the message, etc. A list of commands follows.

a. bcc. Prompts for addresses for blind carbon copies.

b. bye. Exits the 'send' program.

c. delete (delete body). Permits draft body to be deleted after the message is sent.

d. ed body. Invokes the UNIX editor 'ed' and reads in the message body for editing.

e. file (file include). Prompts for the filename to be appended to the end of the message body.

f. header (header edit). Allows user to change the header entries, 'To', 'Subject', etc.

g. input (input more body). Allows the user to add more text to the end of the message.

h. program (program run). Allows the user to give a shell command. Control returns to send when command has been executed.

i. quit. Same as bye.

j. review. Retypes message in its current form.

k. send. Posts the message to the message delivery system.

l. x (emacs body). Invokes the screen editor 'emacs' and reads in the file.

m. z (two-window emacs answer). Prints the message you are to answer along with your reply in a two-window mode.

D. Program Control

1. |. Takes the output of one program and uses it as the input for the next.

2. &. When appended to a command line, detaches the program from the terminal, thus freeing the terminal to do something else.

3. ;. Is used between commands on the same line, forcing them to be executed in sequence from left to right.

4. >. Put output from this program into the file that follows the '>'.

5. >>. Put output from this program into the file; but create the file if it does not exist, or append it to the file if it does exist.

6. <. Take input for this program from the file that follows the '<'.

7. nohup. Runs a process immune to hang-ups, useful for preventing broken pipes and ignoring CTRL-C's and CTRL-B's.

8. tee. Used in conjunction with a pipe; allows output to go to another process (program) as well as the terminal.

9. kill. Kills processes which have been detached from the terminal by use of &.

10. sleep. Puts anything in a wait state for the specified time.

11. wait. Waits until all detached processes have completed and reports on abnormal endings.

12. nice. Runs a process with a low execution priority.

## E. Miscellany

1. date. Prints out the current date and time.

2. factor. Finds the prime factors of a number.

3. dc. A nearly unlimited precision 'desk calculator'.

4. units. Converts measurements from one set of units to another.

5. banner. Creates 'bannered' letters using 6 x 6 matrices of characters.

6. cal. Prints a calendar for the specified year.

# VII. TEXT PROCESSING

The UNIX operating system provides some of the most advanced text processing facilities available. These facilities are sufficient to process input text into a completely formatted document and typeset it for publication, if desired.

## A. The NROFF Family

One of the 'nroff' family's great advantages is that a document can be processed with either 'nroff' or 'troff', without changing the text file.

1. nroff. An extremely sophisticated program which can hyphenate, justify lines, paginate, produce a table of contents, put running headers and footers on pages, and handle footnotes. 'Nroff' is designed to be used with a daisy wheel printer.

2. troff. The companion of 'nroff' which drives a phototypesetter.

3. vroff. Also like 'nroff', but drives a 'Versatec' printer.

4. neqn. A 'pre-processor' used with 'nroff' to process equations and similar mathematical material.

5. eqn. Like 'neqn', but is used instead with 'troff'.

6. tbl. A 'pre-processor' for the handling of tables. Used with either 'nroff' or 'troff'.

B. Macro Packages for Nroff/Troff

1. -ms. The standard macro package; has provisions for footnotes, most standard academic paper formats, etc.

2. -mm. Does all that the standard package does and has the ability to insert tables of contents and other expanded features.

3. macros. There is a macro package written for subcripts, superscripts, and chemical formulas written by Steve Segletes. It is called *. /bmd/steven/tbdfiles/macros and is available for everyone's use.* Documentation is included in the file.

C. Output Filters

1. bold. Is used to produce false or shadow emboldening.

2. col. Performs the overlays implied by reverse line feeds, allows multi-column output on printers which have no reverse line feed capability.

3. dtx. A filter for the DTC terminal.

4. diablx. A filter for the DIABLO 1620 terminal.

D. Other Text Manipulation Programs

1. sort. Sorts a list of lines into alphabetical and/or numerical order.

2. index. Generates an index or concordance.

3. cref. Generates cross references of program source.

4. ptx. Produces a permuted index (keyword-in-context listing).

5. spell. Checks the words of a file against a dictionary and prints out those not found.

6. _typo_. Prints out words likely to have been misspelled.

7. _uniq_. Finds and deletes those lines which are repeated in a file.

8. _diction_. Flags poor diction in specified file by enclosing suspect phrases in '*[words]*'.

9. _suggest_. Suggests a better way of phrasing. Tell 'suggest' the phrase you want to replace and 'suggest' will suggest alternate expressions.

10. _style_. Gives estimate of reading level of document, along with statistics about word use.

## APPENDIX C - Using Emacs

## Just Some of the Commands

```
===========================================================================
^  means hold down the Control Key while typing the character
M- means press the Escape Key, then type the character

___________________________________________________________________________
To CREATE a file using the emacs screen editor:  emacs
To EDIT a file using the emacs screen editor:    emacs your-file-name
To QUIT emacs:                                   ^x^c

___________________________________________________________________________
To ABORT an emacs command:                       ^g
To EXPLAIN an emacs command character:           M-?   "character   to   be
                                                 explained"

===========================================================================
```

To POSITION the cursor

```
                                    ^p
                                     ^
                                     |
                                  up Line

^b      <----backward   Character   forward---->   ^f
M-b     <----backward   Word        forward---->   M-f
^a      <----beginning  Line            end---->   ^e
M-a     <----beginning  Sentence        end---->   M-e
M-v     <----previous   Page        forward---->   ^v
M-<     <----beginning  Buffer          end---->   M->

                                 down Line
                                     |
                                     v
                                    ^n

===========================================================================
```

To SEARCH for a string

```
^r      <----backward               forward---->   ^s
```

To DELETE text

```
*BACK-
SPACE   <----backward  Character    forward---->   ^d
M-DEL   <----backward  Word         forward---->   M-d
                       End-Of-Line  forward---->   ^k
```

*BACKSPACE can be ^h or the BACKSPACE, DELETE, or RUB keys

## To OPEN a line to insert text

```
^o
RETURN  |\
^j      | > These go down one line if one is open; otherwise,they open a line
^m      |/                 and go down to it
```

## To REPLACE text

```
M-r    starts a query replace
       "y" does a replace
       "n" goes to next instance of the string
       "r" replaces the rest of the instances of the string
```

## To MOVE text

1. Position cursor at the beginning of text to be moved.
   Set MARK with M-space.

2. Position cursor at the end of the text to be moved.
   Write text between MARK and cursor into KILL-STACK and delete with "^w".
   Write text between MARK and cursor into KILL-STACK without deleting
   (that is, duplicate the text elsewhere) by "M-w".

3. Position cursor where text is to be inserted.
   Insert text from KILL-STACK into buffer with "^y".

## To SAVE a file

To write a new file:  ^x^w file-name
To save an updated file: ^x^s

## Miscellaneous

| | |
|---|---|
| Capitalize the current character: | ^c |
| Insert a tab: | ^t |
| Redisplay the screen: | ^l |
| Redisplay the screen with the current line at the top: | M-^l |
| Transpose the current character and the one following: | ^t |
| Multiply an argument by 4 | ^u |
| Mail the current buffer | M-^m |
| Execute a shell command | M-! |
| Underline a word | M-_ |

==============================================================================

## This is a complete (hopefully) listing of the EMACS commands

| | |
|---|---|
| ^@: | sets the mark at the cursor position |
| ^A: | moves to the beginning of the line |
| ^B: | moves back one character |
| ^C: | capitalizes the current character |
| ^D: | deletes forward one character |
| ^E: | moves to the end of the line |
| ^F: | moves forward one character |

| | |
|---|---|
| ^G: | quits from any command in progress |
| ^H: | moves back one character |
| ^I: | inserts a tab |
| ^J: | opens a new line and moves to the beginning of it if the next line is non-empty, otherwise down one line |
| ^K: | kills to end of line (with argument, kills multiple lines) |
| ^L: | refreshes the screen |
| ^M: | opens a new line and moves to the beginning of it if the next line is non-empty, otherwise down one line |
| ^N: | moves down one line |
| ^O: | opens up a new line |
| ^P: | moves up one line |
| ^Q: | quotes the next character |
| ^R: | starts a reverse search |
| ^S: | starts a search |
| ^T: | transposes the next two characters |
| ^U: | multiplies the argument by 4 |
| ^V: | moves to the next page |
| ^W: | kills the current region (between cursor and mark) |
| ^X: | is a prefix for more single character commands, type character or '*' for all |
| ^Y: | restores last killed text (leaves cursor and mark around it) |
| ^Z: | exits one level |
| ^[: | makes the next character a meta character |
| ^]: | makes a local variable of a macro invocation the argument to the next command |
| ^^: | causes the last returned result to become the argument |
| ^?: | deletes backward one character |
| M-^B: | moves back one word |
| M-^F: | moves forward one word |
| M-^L: | re-displays with current line at top of page |
| M-^M: | mails the current buffer |
| M-^Q: | returns the next input character (in a macro) |
| M-^R: | regular expression query replace |
| M-^S: | regular expression search |
| M-^Z: | exits one level |
| M-^]: | assigns the result of the next command to a macro local variable |
| M- : | sets the mark at the cursor position |
| M-!: | gets and executes a shell command |
| M-": | auto Fills the whole buffer |
| M-$: | executes a command, saving the output in buffer .exec |
| M-/: | starts a comment |
| M-:: | maps a character to a command |
| M-<: | moves to top of file |
| M->: | moves to bottom of file |
| M-?: | explains the next character |
| M-\: | converts its argument to a character and inserts it |
| M-_: | underlines the next word |
| M-a: | moves to beginning of sentence |
| M-b: | moves back one word |
| M-c: | capitalizes the next word |
| M-d: | deletes the next word |
| M-e: | moves to End of Sentence |
| M-f: | moves forward one word |

| | |
|---|---|
| M-g: | moves to a specific line (its argument) |
| M-m: | displays active modes |
| M-p: | puts a wall chart of explanations in the buffer |
| M-q: | quotes the next character and adds the 0200 bit |
| M-r: | starts query replace |
| M-s: | gives EMACS statistics |
| M-t: | prompts for terminal type |
| M-v: | moves back one page |
| M-w: | puts the current region in the kill buffer without killing it |
| M-x: | calls a macro by name |
| M-y: | replaces the last restore() with the next text in the kill stack. |
| M-z: | kills emacs with a core dump (for debugging) |
| M-{: | enters a command sequence (in a macro) |
| M-}: | exits a command sequence (in a macro) |
| M-~: | marks a buffer as being unmodified (up to date) |

Control-X commands:

| | |
|---|---|
| ^X^B: | changes Buffers (Change to * lists active buffers) |
| ^X^C: | exits gracefully (after asking whether or not to save the buffer) |
| ^X^D: | changes the working directory |
| ^X^E: | calls emacs recursively taking input from the terminal |
| ^X^F: | edits a file in its own buffer (if file has been read into a buffer, moves to it) |
| ^X^I: | re-directs input from a file |
| ^X^K: | kills a buffer |
| ^X^L: | loads a file full of macro definitions |
| ^X^M: | sets mode from argument (prompts for mode name) and string if necessary |
| ^X^N: | changes the buffer or file name |
| ^X^O: | switches between windows |
| ^X^Q: | returns the character under the cursor (in a macro) |
| ^X^R: | reads a new file |
| ^X^S: | saves the buffer in the current file (if modified) |
| ^X^T: | prompts for a buffer name and inserts the text between the cursor and the mark into the named buffer. |
| ^X^U: | updates the display and delays for a specified time |
| ^X^V: | puts the current version on the kill stack. |
| ^X^W: | writes a new or old file |
| ^X^X: | exchanges the mark and the cursor |
| ^X^^: | enters a "while" loop (in a macro) |
| ^X!: | begins a case statement (in a macro) |
| ^X%: | exchanges the top of the kill stack with another item |
| ^X&: | compares two strings |
| ^X+: | causes the next entry to the kill stack to append to the previous entry |
| ^X-: | pops the kill stack |
| ^X1: | exits two window mode |
| ^X2: | enters two window mode |
| ^X<: | pushes a string from the tty or macro text into the kill stack |
| ^X=: | gives statistics about the buffer |
| ^X>: | duplicates an item on the kill stack |
| ^XB: | puts the buffer name into the kill stack |
| ^XF: | puts the file name into the stack |

| | |
|---|---|
| ^X^: | causes the current window to grow one by line |
| ^Xb: | changes Buffers (Change to * lists active buffers) |
| ^Xd: | defines macros from the current buffer |
| ^Xo: | switches between windows |
| ^X\|: | begins a conditional execution sequence (in a macro) |
| ^X~: | performs arithmetic or logical operations (in a macro) |

## APPENDIX D - UNIX READING LIST*

*Available from Nancy Cider

## I. General

"UNIX Programmer's Manual" (Ken Thompson, Dennis Ritchie, and a cast of thousands). Lists commands, system routines and interfaces, file formats, and some of the maintenance procedures. You will probably use this a lot to make sure of the formats for various commands. It is rough going to learn how to use it at first, but the effort put forth will really be rewarding.

"The UNIX Time-sharing System" (Ken Thompson, Dennis Ritchie). An overview of the system, for people interested in operating systems. Worth reading by anyone who programs. Contains a remarkable number of one-sentence observations on how to do things right.

"A Tutorial Introduction to the UNIX Text Editor" (Brian Kernighan). A walkthrough introduction to the line editor "ed". It includes commands for appending, changing, deleting, moving, and inserting lines of text; reading and writing files; searching; global commands; and special characters.

"An Introduction to the UNIX Shell" (S. R. Bourne). Works from simple examples up to higher level constructs of the shell. It starts out with definitions and procedes quickly to constructing significant programs and flow-structures using the shell. Probably most helpful if you have had some programming.

## II. Document Preparation

"Text Processing" (BRL compilation, by Bruce Henriksen, Myra Hartwig, and others). Helpful for all those persons wishing to get started with some of the more complicated text processing packages. Includes an introduction to "nroff" and "troff", and talks about df's, letters, the memorandum for record, and some other helpful stuff.

"Typing Documents on UNIX" (Mike Lesk). A macro package to isolate the novice from the vagaries of the formatting programs. This one works with both "nroff" and "troff".

"NROFF/TROFF User's Manual" (Joseph F. Ossanna). Difficult to read, but good for reference.

"A TROFF Tutorial" (Brian Kernighan). Good for possibilities in TROFF. Especially useful when making viewgraphs.

"Tbl -- A Program to Format Tables" (M. E. Lesk). Used for making tables in an 'nroffed' report.

"A System for Typesetting Mathematics" (Brian Kernighan and Lorinda L. Cherry). This and the following document detail how to put equations into your 'nroff' and 'troff' papers.

"Typesetting Mathematics -- User's Guide (Second Edition)", (Brian Kernighan and L. L. Cherry).

## III. Programming

UNIX Programming -- Second Edition" (Brian Kernighan and Dennis Ritchie). Introduction to programming on the UNIX system, especially concerns interfacing to the operating system and I/O.

"Programming in C: A Tutorial" (Brian Kernighan). The easiest way to start learning C, but it's no help at all with the interface to the system beyond the simplest I/O. Should be read in conjunction with:

"The C Programming Language -- Reference Manual" (Dennis Ritchie). A good general reference, but a bit heavy going for the beginner, especially one who has never used a language like C.

"UNIX Assembler Reference Manual," Dennis Ritchie.

"The UNIX I/O System," D. M. Ritchie.

"The M4 Macro Processor," B. W. Kernighan and D. M. Ritchie.

"A Portable Fortran 77 Compiler," S. I. Feldman and P. J. Weinberger.

"Ratfor -- A Preprocessor for a Rational Fortran," B. W. Kernighan.

## IV. Other

"Computer Detection of Typographical Errors," R. H. Morris and L. L. Cherry.

"YACC (Yet Another Compiler-Compiler)," S. C. Johnson.

"BC -- An Arbitrary Precision Desk-Calculator Language," L. L. Cherry and Robert Morris.

"DC -- An Interactive Desk Calculator," Robert Morris and L. L. Cherry.

"Lex -- A Lexical Analyzer Generator," M. E. Lesk and E. Schmidt.

See me about anything you may have heard about, but don't see listed here. I may have a copy, or may at least be able to tell you where to go to get it to avoid a lot of hassle.

## APPENDIX E - TERMS

Hardware is the physical units that make up a computer system. This includes the computer itself, disk drives, printers, plotters, and other output devices.

Software is the sets of instructions that tell the physical units how to perform their functions. That is, how to run your programs, how to properly print out the results, and it includes the programs that you are telling the computer to run.

The Operating System is the programs that controls the system. It handles the scheduling of jobs (or programs), and the allocation of resources (including the printers, disk drives, etc.). On our computer, the operating system is UNIX.

A Modem (or modulator/demodulator), also known as a data set, converts the machine signals from the terminal to communication signals that can be sent over telephone lines to the computer.

Terminals are devices that permit data entry or exit to/from a computer system. What you type on a terminal enters the system and is processed, and the results show up again, either on paper or on the screen.

Ascii (or American Standard Code for Information Interchange) is the code for the characters that go from your terminal to the computer. If you have an Ascii terminal and you type an 'a', it will be understood by the computer as an 'a'. If you were using a different standard on your terminal and you typed an 'a', the computer might think you were typing an 'A'.

A Bit is the basic unit of data storage. It is a binary 0 or 1, and the name comes from binary digit.

A Byte is the basic unit of data that is processed. It is eight bits long, and is the length of one character.

Baud is a unit of signaling speed that is measured in terms of units/sec., and the units are usually either bits or characters.

A Buffer is a temporary storage device used to compensate for data flow rate differences. When a receiving device cannot handle as many characters at a time as the device sending those characters it must compensate in some way. It uses a buffer to gather up the characters as they come in, then when the buffer is full, it processes one buffer at a time. A transmitting device works in the same fashion. It collects characters until its buffer is full, then it sends them all at once.

Duplex is a term used in communications. Half duplex is a channel that can either transmit or receive at any given time. Full duplex is a channel that can receive and transmit information at the same time.

A Multiplexer is a piece of communications equipment which enables the signals from a number of individual circuits to be combined and transmitted over a common transmission path. Many terminals can be attached to a multiplexer and the multiplexer hooked up to a telephone line, and then each terminal has its own share of the line in turn.

A Network is a geographically separated set of points (computers and terminals) which are interconnected by communications channels. The advantage to this is that any given terminal may then speak to any given computer that is part of the network.

Arpanet and Milnet are the two networks established by the Advanced Research Projects Agency of the Department of Defense.

Bug is used to denote an unwanted and/or unintended property of a program or system.

Crash is used when there is a sudden, drastic failure. Usually said of the system and disk drives.

Down is used when something is not working, this is usually occurring by plan, e.g., for repair work.

A Hacker is someone who programs to learn and as a challenge, usually working rather quickly. A hacker can also be an expert at a particular type of program or operating system, e.g., a UNIX hacker, or a Fortran hacker.

TTY is a common abreviation for teletype, used also as a shorthand for terminal type.

## DISTRIBUTION LIST

| No. of Copies | Organization |
|---|---|
| 12 | Administrator<br>Defense Technical Info Center<br>ATTN: DTIC-DDA<br>Cameron Station<br>Alexandria, VA 22314 |
| 1 | Commander<br>US Army Materiel Development and Readiness Command<br>ATTN: DRCDMD-ST<br>5001 Eisenhower Avenue<br>Alexandria, VA 22333 |
| 1 | Commander<br>US Army Armament Research and Development Command<br>ATTN: DRDAR-TDC<br>Dover, NJ 07801 |
| 2 | Commander<br>US Army Armament Research and Development Command<br>ATTN: DRDAR-TSS<br>Dover, NJ 07801 |
| 1 | Commander<br>US Army Armament Materiel Readiness Command<br>ATTN: DRSAR-LEP-L<br>Rock Island, IL 61299 |
| 1 | Director<br>US Army Armament Research and Development Command<br>Benet Weapons Laboratory<br>ATTN: DRDAR-LCB-TL<br>Watervliet, NY 12189 |
| 1 | Commander<br>US Army Aviation Research and Development Command<br>ATTN: DRDAV-E<br>4300 Goodfellow Blvd<br>St. Louis, MO 63120 |
| 1 | Commander<br>US Army Communications Rsch and Development Command<br>ATTN: DRSEL-ATDD<br>Fort Monmouth, NJ 07703 |
| 1 | Commander<br>US Army Electronics Research and Development Command<br>Technical Support Activity<br>ATTN: DELSD-L<br>Fort Monmouth, NJ 07703 |
| 1 | Commander<br>US Army Missile Command<br>ATTN: DRSMI-R<br>Redstone Arsenal, AL 35898 |
| 1 | Commander<br>US Army Missile Command<br>ATTN: DRSMI-YDL<br>Redstone, Arsenal, AL 35898 |
| 1 | Commander<br>US Army Tank Automotive Command<br>ATTN: DRSTA-TSL<br>Warren, MI 48090 |
| 1 | Director<br>US Army TRADOC Systems Analysis Activity<br>ATTN: ATAA-SL<br>White Sands Missile Range<br>NM 88002 |
| 2 | Commandant<br>US Army Infantry School<br>ATTN: ATSH-CD-CSO-OR<br>Fort Benning, GA 31905 |
| 6 | Massachusetts Institute of Technology, Artificial Intelligence Lab, 545 Technology Square, Rm 912<br>ATTN: Martin D. Connor<br>Cambridge, MA 02138 |

DISTRIBUTION LIST

| No. of Copies | Organization |
|---|---|
| 1 | Director<br>US Army Air Mobility Research and Development Laboratory<br>Ames Research Center<br>Moffett Field, CA 94035 |
| 1 | AFWL/SUL<br>Kirtland AFB, NM 87117 |

| No. of Copies | Organization |
|---|---|
| | **Aberdeen Proving Ground** |
| | Dir, USAMSAA<br>ATTN: DRXSY-D<br>DRXSY-MP, H. Cohen<br>Cdr, USATECOM<br>ATTN: DRSTE-TO-F<br>Director, USACSL<br>ATTN: DRSMC-CLB-PA<br>DRSMC-CLN<br>DRSMC-CLJ-L |
| | Product Assurance Directorate<br>Chemical Test Branch<br>ATTN: Darlene Bader<br>Building E5100<br>Edgewood Arsenal, MD 21040 (6) |

## USER EVALUATION OF REPORT

Please take a few minutes to answer the questions below; tear out this sheet, fold as indicated, staple or tape closed, and place in the mail. Your comments will provide us with information for improving future reports.

1. BRL Report Number ______________________________

2. Does this report satisfy a need? (Comment on purpose, related project, or other area of interest for which report will be used.)

______________________________

______________________________

______________________________

3. How, specifically, is the report being used? (Information source, design data or procedure, management procedure, source of ideas, etc.) ______________________________

______________________________

______________________________

4. Has the information in this report led to any quantitative savings as far as man-hours/contract dollars saved, operating costs avoided, efficiencies achieved, etc.? If so, please elaborate.

______________________________

______________________________

5. General Comments (Indicate what you think should be changed to make this report and future reports of this type more responsive to your needs, more usable, improve readability, etc.) __________

______________________________

______________________________

______________________________

6. If you would like to be contacted by the personnel who prepared this report to raise specific questions or discuss the topic, please fill in the following information.

Name: ______________________________

Telephone Number: ______________________________

Organization Address: ______________________________

______________________________

______________________________

— — — — — — — — — FOLD HERE — — — — — — — — — — — —

Director
US Army Ballistic Research Laboratory
ATTN: DRSMC-BLA-S (A)
Aberdeen Proving Ground, MD 21005

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

OFFICIAL BUSINESS
PENALTY FOR PRIVATE USE, $300

**BUSINESS REPLY MAIL**
FIRST CLASS PERMIT NO 12062 WASHINGTON, DC

POSTAGE WILL BE PAID BY DEPARTMENT OF THE ARMY

Director
US Army Ballistic Research Laboratory
ATTN: DRSMC-BLA-S (A)
Aberdeen Proving Ground, MD 21005

— — — — — — — — — FOLD HERE — — — — — — — — — — —

ND

DATE
LMED

-84

DTIC