1030

Bell Laboratories Cover Sheet for Technical Memorandum

Title- Documentation a la UNIX

Date- June 15, 1974

TM- 74-9152-2

Other Keywords- Text Processing

Authors	Location	Extension	Charging Case- 39373-09
J. R. Mashey	RR 4A831	6089	Filing Case- 39373-99

ABSTRACT

The purpose of this memo is to provide a brief tutorial and overview on the use of UNIX to produce and maintain documentation. In the first few pages are outlined the four main areas of information which must be learned: communication with UNIX, the UNIX file system, text editing, and text formatting. In conjunction with the included annotated bibliography, these pages should give the prospective UNIX user a quick grasp of the available facilities and documentation describing them. The remainder of the memo is intended to help the beginning user of UNIX documentation tools to get off the ground quickly. Detailed hints are given, expanding upon each of the four topics noted. Hopefully, this memo should serve as a quick introduction to the flexibility and convenience of UNIX-style documentation procedures.

Pages Text 23	Other 3	Total 26	
No. Figures 2	No. Tables 0	No. Refs. 13	

Subject: Documentation a la UNIX

Date: June 15, 1974

Case: 39373-090

From: J. R. Mashey

TM-74-9152-2

MEMORANDUM FOR FILE

This memo provides a brief overview and tutorial regarding the use of UNIX to produce documentation. The first few pages should give a prospective user of UNIX a grasp of its capabilities in this area. A beginning user should find the remainder helpful in getting off to a quick start, without being burdened by unnecessary details. The four topics emphasized are: communicating with UNIX, use of the UNIX file system, editing documents, and formatting documents.

CONTENTS

Documentation a la UNIX

PURPOSE

This document is intended for the prospective UNIX user who wishes to utilize UNIX mainly for the purposes of producing, storing, and maintaining documentation. It is not intended to provide detailed information on the available facilities, as they are quite extensive. Rather, it is to be used to help the new user off to a quick start, summarize the capabilities of available software, and provide pointers to more extensive documentation. USING UNIX FOR DOCUMENTATION is a quick overview of the subject, which is then expanded in the more extensive HINTS section.

USING UNIX FOR DOCUMENTATION

To produce documentation efficiently and conveniently, the user should be familiar with four areas of information:

I. COMMUNICATION: the user must know how to login to a UNIX system and communicate with it. This requires that the user:
(a) obtain a valid user name from the UNIX system administrator.
(b) obtain the telephone number of the desired UNIX system.
(c) login to the system in a manner appropriate to the specific type of terminal being used.
(d) become familiar with UNIX typing conventions, noting especially that two characters have particular significance: @ and # .

WHAT TO READ: Reference (1), section "HOW TO GET STARTED". Also, it may be helpful to glance at part sh(I), which describes the terminal command interpreter.
NOTE: anything of the form name(ROMAN NUMBER) refers to the named part of the numbered section of reference (1).

II. FILE SYSTEM: it is helpful for the user to know a few things about the UNIX file system.
(a) Files are sequences of characters, whose structure and interpretation is completely controlled by the user. Ordinary files may generally be created, manipulated, and destroyed by the user with great ease. Paradoxically, this ease of use is sometimes a stumbling block: people often expect complexity where there is no need for any. In general, UNIX provides a file system which allows the user to ignore irritating details of allocation and manipulation of physical storage for files. The user need not know any of the following:
what type of device is being used to store a file.
which disk volume is used.
the number of tracks, cylinders, etc. to be used.
the record format, blocksize, etc, of the files.
A user may simply refer to files by name, without being bothered by such rigmarole as the above.

(b) Certain files called directories provide a mapping from the names of the files to the files themselves. Although the user may not directly modify directories, very simple commands exist for manipulating them. It is easy for a user to create files and to group them by whatever directory structure is found convenient.
(c) UNIX imposes a structure upon the relationships of directories. (see Figure 1.) This structure is that of a rooted tree. Beginning with the unique root directory, any directory may point to zero or more other directories, and it is then said to be the parent directory of those other directories. Note that a directory may have many children, but only one parent. By contrast, a single ordinary file may have several parent directories (see the ln(I) command for creating multiple links from directories to files).
(d) Files may generally be accessed in several different ways. A file is uniquely specified by giving a path name for it. This consists of a sequence of directory names, separated by slashes, followed by the desired file name, such as:

/usr/mash/unixdocintro

which may be interpreted as: the root directory (/) is the parent of directory usr, which is in turn the parent of directory mash, which points to the file unixdocintro. Such a mechanism resembles the catalogs used by OS/360 and similar systems. However, forcing users to reference their files by such long names would be a general harassment. For convenience, UNIX provides the user with a current directory, which is automatically assigned upon login, but may be changed by the user as desired. This allows the user to forget about using long names for files. For instance, suppose that a user wants to print the file mentioned above. If his current directory were /usr/mash, all he would have to do is type:

pr unixdocintro

i.e., UNIX would note the the requested file name does not begin with a slash, and would look for the file in the current directory instead. In fact, path names may be used also: suppose that the user's current directory were /usr. Then the same file could be printed by typing:

pr mash/unixdocintro

thus illustrating the fact that the possible names for a any given file depend on the user's current directory. Because the user may change the current directory, it becomes very convenient to "move around" in the file system, to whatever directories permit the most convenient specification of names.

WHAT TO READ: Reference (2), parts 3.1, 3.2, 3.5, 3.7 describe ordinary files, directories, protection, and the file system hierarchy. Part 5 of this reference describes the standard command interpreter (the "shell"), and is thus useful for knowledge of file manipulation.

III. EDITING: the bulk of most users' terminal connect time will probably be spent entering and editing text via the UNIX editor. This program permits convenient creation, inspection, and alteration of files consisting of sequences of lines. Lines correspond roughly to punch cards in use, but are of variable size, and thus lend themselves to convenient representation of programs and text. More specialized form letter editing is done with fed(I) and form(I).

WHAT TO READ: Reference (1), section I, part ed(I) describes the editor, albeit somewhat tersely. Reference (3) gives a more tutorial introduction to it, with many examples and exercises.

IV. FORMATTING: when entering text into a UNIX file, normally control lines and other characters are also inserted in addition to the actual text. Various formatting programs can then be used to read the raw text plus control information, printing attractively-formatted output. Among the services provided are justification to even margins, hyphenation, underlining, automatic page numbering, headers, footers, and various others. Note that the document you are reading was produced using one of these programs. Output of such programs can be directed to a file to be saved, printed immediately on a typewriter terminal, displayed on a CRT terminal, or printed on a phototypesetter device (to be available at RRC in the near future). The latter offers multiple type fonts (italics, boldface, etc), multiple type sizes, plus many exotic special characters.
It is well worth the time to become familiar with a basic subset of the commands accepted by these programs.

WHAT TO READ: Reference (1), parts roff(I), nroff(I), and troff(I) briefly describe the use of the three major programs, which are generally very similar. Reference (5) describes one of these in detail (NROFF): for various reasons, it is the recommended one to use. Reference (4) is a tutorially-oriented description of a similar program.

INFORMATION SOURCES

The following documents may be of interest: contact local UNIX system administrators regarding distribution of items which are not Technical Memos. They can also direct you to a much more complete bibliography maintained by the UNIX Support Group.

(1) K. Thompson & D.M. Ritchie, UNIX PROGRAMMER'S MANUAL (4th ed.): this is the primary source of information on the use of UNIX. Only some sections of it are useful to one wishing to prepare documentation, rather than writing UNIX programs. These sections are discussed later.

(2) D.M. Ritchie, MM 71-1273-4 The UNIX Time-Sharing System: Although slightly out of date by now, this memo provides a good overview of UNIX, especially with regard to the file system. Although not required, it is useful for background information. (18 pages).

(3) B.W. Kernighan, A Tutorial Introduction to the ED Text Editor: This tutorial provides many examples of the use of ED, and is especially recommended for the user not familiar with this type of editor. (21 pages).

(4) MHCC-005 Roff Text Formatter: this describes a text formatter like those mentioned above. Although this version of Roff is not identical to the UNIX formatters, this tutorial might be a useful introduction. (29 pages).
NOTE: MHCC publications are obtained from the Computer Information and Documents Group, MH 2C-548, x4373.

(5) J.F. Ossanna, MM 73-1271-2 NROFF User's Manual: this is a detailed description of the formatter most likely to be used. (21 pages).

(6) B.W. Kernighan, TM 73-1273-10 TROFF Made Trivial: this gives a brief description of the facilities available by using a phototypesetter. All of the memo (including the cover page with bell symbol) was produced by TROFF. It contains many unusual special characters, multiple type fonts, and other items not available on normal printers. (21 pages).

(7) R. Morris, L.L. Cherry, TM-73-1271-4, Computer Detection of Typographical Errors. This describes the typo(I) program mentioned elsewhere. It is easy to use typo without reading this, but some users may find it interesting reading. (18 pages).

(8) D.M. Ritchie, TM-74-1273-1, C Reference Manual: this defines the C language, which is the language used to implement much of UNIX and the programs which run under it. C is definitely the first language to consider if it becomes necessary to move from using programs to writing them. (24 pages).

(9) J.F. Ossanna, TM 74-1271-4, TROFF User's Manual: a detailed guide to phototypesetting documents with TROFF. (31 pages).

(10) G. C. Vogel, NROFF Macros for Producing Documents on UNIX, PROGRAMMER'S NOTES, March 29, 1974: a brief description of a package of NROFF macros which includes section titling and automatic numbering, paragraphs, tables, and automatic production of table of contents. (4 pages).

(11) Brian Kernighan, Mike Lesk, Typing Documents on UNIX: description of a macro package usable for both NROFF and TROFF. It provides macros for section numbering, indented paragraphs, boldface and italics, and various additional items. (2 pages).

(12) J. R. Mashey, More NROFF Macros, PROGRAMMER'S NOTES: description of a macro package similar to that of (10), but aimed at certain BISP applications requiring a different format. (6 pages).

(13) B. W. Kernighan, L.L. Cherry, Typesetting Mathematics-User's Guide, TM-74-1273-3, TM-74-1271-3: this memo describes a preprocessor for NROFF/TROFF, which makes the formatting of mathematical formulae much easier to do. Output can be produced either on the phototypesetter using TROFF, or using NROFF on a terminal with half-line capability (Teletype 37 or GSI, for example). (10 pages).

HINTS

This section expands USING UNIX FOR DOCUMENTATION by supplying hints to get the user on the system quickly.

I. COMMUNICATION

(a) TABS: the typewriter tabs should normally be set to those expected by UNIX, e.g., 8 spaces per tab. Be warned that printing text containing tab characters on a terminal whose tabs are not properly set will produce output you will not like. It is recommended that the user get into the habit of issuing the following command immediately after any login:

```
cat /usr/pub/tabs
```

which sets the tabs as expected (8 spaces/tabs). The user can later learn to create more specific tab files.

(b) COMMUNICATING WITH OTHER USERS: the purposes listed can be obtained by the commands shown:

FIND OUT WHO ELSE IS LOGGED INTO THE SYTEM:

```
who
```

SEND MAIL TO ANOTHER USER:

```
mail letter person
```

letter is a file you want sent, and person is a user name. When they next login, they will get a message: "YOU HAVE MAIL".

READ LETTERS IN YOUR MAILBOX:

```
cat mailbox
OR
mail
```

For more details and options, see mail(I).

CONVERSE WITH ANOTHER USER CURRENTLY LOGGED IN:

See the write(I) command: you must know how to respond when someone else writes to your terminal. (Systems administrators may write to you at various times).

PREVENT BAD OUTPUT CAUSED BY OTHER USERS WRITING TO YOU:

```
mesg n
```

When you're printing something important; this inhibits writing to your terminal. To permit such writing, type:

```
mesg y
```

II. FILE SYSTEM

(a) DIRECTONIES: as noted before, UNIX assigns each user a current directory upon login. If the user logs in as username, then the current default directory is:

```
/usr/username
```

Each user has such a directory created for them when they are added to the system.

The following commands should be known for directory manipulation: chdir(I): change current directory, ls(I): list contents of directory, mkdir(I): make a directory, and rmdir(I): remove a directory. Common actions are:

LIST CONTENTS OF CURRENT DIRECTORY:

```
ls
OR
ls .
```

Both forms are equivalent: the period is another name for the current directory.

LIST CONTENTS OF PARENT DIRECTORY (OF CURRENT DIRECTORY):

```
ls ..
```

.. is a special name which always refers to the parent of the current directory. It lets the user access it without having to specify a complete path name.

Both of the previous are of course special cases of the general command ls directory-name. See ls(I) for various other optional services.

CREATE A NEW SUBDIRECTORY (IN THE CURRENT DIRECTORY):

```
mkdir new-subdirectory-name
```

CHANGE CURRENT DIRECTORY:

```
chdir directory-name
```

This allows the user to "move" in the file system to the most convenient place.

REMOVE (DESTROY) A DIRECTORY:

```
rmdir directory-name
```

CREATE A LINK FROM A DIRECTORY TO AN EXISTING FILE IN ANOTHER:

```
ln existing-filename
        OR
```

ln existing-filename myfilename

As mentioned previously, although any directory may have only one parent directory, but an ordinary file may have several. Among other things, this permits any user to reference any file by a simple name, rather than a complete pathname. No file will actually be eliminated until all links to it are removed. All directories are considered equal in this matter; it does not matter which directory hao the file first. The first ln command above causes an entry to appear in the current directory which uses the same name as the existing file (without preceding directory names). The second causes "myfilename" to be used instead. Thus, many users may each have links to a single file, but each may access it by the name they choose. For example, suppose there is a file named /usr/mash/textmacro2, and you issue the following command:

ln /usr/mash/textmacro2 mymacro

There is then a link from your current directory to the file named, so that any references to mymacro access this file.

Suppose there exist the following files. The list itself illustrates the facts that upper and lower case letters are distinguishable, that periods may be used in file names, and that up to 14 characters may appear in a name. (see Figure 1.)

```
/usr/jones/file1
/usr/jones/file2
/usr/jones/direc/file1
/usr/jones/direc/file2
/usr/jones/DIREC/file1
/usr/jones/DIREC/a.longfilename
/usr/smith/ffle1
/usr/smith/file2
/etc/x/y/z
/tmp
```

When user Jones logs in, his current directory is /usr/jones. If he then types:

```
ls
```

the list printed would be:

```
DIREC
direc
file1
file2
```

If he types:

```
ls DIREC
```

he would see:

```
a.longfilename
file1
```

If he would like to print file /usr/jones/file1, it is necessary only to type:

```
pr file1
```

while printing file /usr/jones/DIREC/file1 would require:

```
pr DIREC/file1
```

However, if Jones wanted to manipulate a number of files in DIREC, he could move there via:

```
chdir DIREC
```

after which he could reference the /usr/jones/DIREC/file1 file using

```
pr file1
```

If Smith logged in and then wished to know what other users had directories, this could be done by typing:

```
ls /usr
```

or equivalently (in this case):

```
ls ..
```

If Smith wished to examine Jones' files, one (of several) ways to do so is to switch current directories as follows:

```
chdir ../jones
```

Although these examples have presented files as being accessible to anyone, it is possible to protect them from undesired reading or writing: see chmod(I).

TO SUMMARIZE THE WAYS NAMES ARE USED, WE HAVE THE FOLLOWING POSSIBILITIES:

Format	Meaning
name1	name1 is a file or directory whose parent is the current directory.
name1/name2/..../namen	the current directory is a parent of name1, which is a parent of name2, etc.
/name1/name2/..../namen	this is a complete pathname to file namen: name1 is a child of the root directory /, name2 is a child of name1, etc.
..	parent of the current directory
.	another name for the current directory

(b) ORDINARY FILE MANIPULATION: files are most commonly created by use of the UNIX editor (see part III of this memo). The following sequence is typical:

```
ed                  to enter the editor
.....sequence of commands to create text
w newfilename       create new file
```

This sequence would create the named file in the user's current directory. The following are other common operations:

REMOVE ONE OR MORE FILES:

```
rm filename1 filename2    etc
```

Another way to remove files is to use the dsw(I) command. It is effectively an ls command which lists your files, one by one, and waits for you to reply to each. If you reply y to a filename, it is deleted, otherwise it is not affected.

PRINT ONE OR MORE FILES:

```
cat filename1  filename2      etc
        OR
pr filename1 filename2    etc
```

The first way is the fastest and simplest. The second also adds the date, filename, and page number as a heading for each page. The pr(I) command also has other options.

PRINT PIECES OF A FILE:

Looking around in a file is normally done using the editor.

MAKE A COPY OF A FILE:

```
cp existingfile  newfile
```

MOVE (RENAME) A FILE:

```
mv oldfile  newfile
```

SPLIT A BIG FILE INTO SMALLER PIECES:

```
split filename
```

the file is split into 1000-line chunks (see split(I) for details). Big files may become slow to edit and manipulate. Also, the editor can be used for this purpose.

(c) EXTENSIVE FILE MANIPULATION: many other utilities exist for inspecting and manipulating files. These include sort(I) for sorting files, comm(I) which prints lines common to two files, and uniq(I), which effectively removes repeated lines in a file. Merge(I) can be used to merge from two to eight files together. Tr(I) performs translation of a file on a character basis, and can be used to squeeze out repeated strings of characters. Changes from one version of a file to another can be found using proof(I) (or mad(V) on some systems). Command grep(I) allows a file to be searched, and those lines printed which are matched by a given pattern.

(d) HANDLING OF MULTIPLE FILES AND COMMAND FILES: UNIX offers convenient handling of multiple files and involved sequences of commands. First, many commands are not restricted to single files as arguments, but can process a sequence of them (see cat or pr for example). Second, the usual terminal command interpreter (the "shell", described under sh(I)) is an unusually flexible one. For example, it will let you access a group of files without requiring you to type all their names explicitly. Instead of typing filename(s) as arguments to a command, you type a pattern, which may used to match the names of actual files. The list of filenames matched is effectively substituted for the pattern. For example, suppose your current directory has files X1, x2, x3, x4, x5, xx6, y1, and y2. Then the command:

```
pr x?
```

is interpreted as:

```
pr x1 x2 x3 x4 x5
```

while the command:

```
pr ?1
```

is the same as:

```
pr x1 y1
```

and the command

```
pr x2 x3 x4
```

could be more easily typed as:

```
pr x[2-4]
```

Typing pr *[4-6] would print files x4, x5, and xx6, and the command ls * would list the names of all files which are descendents of your current directory, whether immediate or remote.

Third, some commands normally read from your terminal and write to it. The shell can arrange that they read from a file, write a file, or append to the end of a file instead (see the use of <, >, and >> under sh(I)).

Finally, the shell program is itself an ordinary command, and can itself be invoked directly (or by program). It is easy to create a file of UNIX commands, then pass it to the shell in such a way that it executes them as though you typed them directly. This implies that you can instruct long command sequences (including even conditional branching) and save them for later use.

III. EDITING

(a) UNIX EDITOR: users will perform most of their editing via the ed(I) command. It is described in the UNIX manual, but rather tersely, so we offer a few quick comments here. A much more extensive tutorial introduction may be found in reference (6). This section should be read with the UNIX section ed(I) handy, and preferably while sitting at a terminal: the fastest way to learn the editor's use is to try it.

(1) Many editors are oriented towards the use of line numbers: each line is numbered, usually permanently, and the user addresses the lines using these numbers. ed is a context editor, like the QED editor on which it is based. It provides no permanent line numbers: although each line has a definite number at any point in time, this number changes as lines are inserted or deleted in front of it. However, this is no real loss, as ed permits the user to search for lines having specified character strings in them, which is much more convenient anyway. In fact, although any line can be addressed by number, about the only one used is 1, when inserting something in front of a file.

(2) The editor has two modes: editing and file manipulation are done while in command mode, which recognizes certain character sequences as requests. While in input mode, anything except a line consisting of a single period (immediately followed by a newline) is accepted as input text, and added to the file. Figure 2. illustrates the various modes, commands, and resulting actions in using the editor.

(3) If UNIX crashes, or if you lose your terminal connection, ed does not automatically save your working file anywhere. It is thus wise to issue w commands occasionally, thus saving the edited text in a permanent file. Also, the editor does not save your buffer when you quit: only explicit w commands cause writing of files.

(4) For initial proofreading of a document, try using the program typo(I). Typo prints a list of all (except very common) words in your text, in order by "peculiarity index", which indicates likelihood of error. A quick inspection of the words near the beginning of the list usually will catch most typos quickly, without requiring the labor of printing the document and scanning it. A typical sequence of commands would be to write the current buffer to a file and then use typo:

```
w filename
!typo filename
```

If this program intrigues you, see reference (7).

(5) COMMONLY-REQUIRED ACTIONS: the following lists actions which are often needed, and examples of command sequences to perform them:

PRINT ENTIRE BUFFER:

```
1,$p
```

The $ refers to the current last line of the buffer.

PRINT EVERY LINE CONTAINING SOMETHING:

```
g/SOMETHING/p
```

CHANGE EVERY OCCURRENCE OF SOMETHING TO SOMETHINGELSE:

```
g/SOMETHING/s//SOMETHINGELSE/g
        Ok
g/SOMETHING/s//&ELSE/g
        Ok
1,$s/SOMETHING/&ELSE/g
```

The g command first marks each line containing SOMETHING, then for each line, it substitutes for every occurrence of the last thing matched (represented by //, referring back to SOMETHING), the string SOMETHINGELSE. The second command is equivalent, but uses the & to denote the string just matched. The third is another equivalent form.

INSERT SOMETHING AT THE BEGINNING OF THE CURRENT LINE:

```
s/^/SOMETHING/
```

INSERT SOMETHING AT THE END OF THE CURRENT LINE:

```
s/$/SOMETHING/
```

DELETE PART OF A LINE:

```
s/PART OF LINE TO BE DELETED//
```

i.e., substitute a null (empty) character string for whatever should be deleted. For example, use s/th// to fix up the following line:
Thisth is in error.

FIXING ERRORS:

Under normal UNIX conventions, the typing of an @ character erases the entire current line, while a # erases the last character. Although typing might often continue on the same line after an @, it may not be convenient. For example, you may be typing a table or a program in which tab characters and columns are significant. If after typing an @, you just hit RETURN, you will get an empty line (in effect a blank one) inserted there, which you may not want. In this case, a

convenient way out is to add something unique to the lines rather than cancelling it, then later delete all such lines globally: add zz, then type a newline. After this has been done a number of times, type g/zz/d to wipe out all lines with zz in them.

(6) SPECIAL CHARACTERS: if in using a regular expression (see ed writeup) to address a line, you get a ? in response, and you're sure the addressed line exists, you may be using one of the reserved special characters in it, i.e., one of ^$.*[] . If you are actually looking for one of these, remember to precede it with a \ . Of course, always be careful with # and @.

(b) FORM LETTER EDITING: the commands fed(I) and form(I) are provided to set up and produce form letters. They are especially useful for automating "fill in the blanks" activities.

The form command copies a prototype (skeleton) letter to the output file, making various insertions and substitutions as requested. The prototype letter is created using the fed command. When form processes the prototype, certain kinds of character strings request various sorts of substitutions. Commonly-used forms include:

[0] form substitutes the current date.

[n] the nth argument to the form command is substituted.

[name] form first looks in a list of named character strings (constructable via fed), and substitutes the string attached to the given name, if there is one. If not, it prompts the terminal by printing [name]: . The user can then enter the desired string and have it inserted.

See the UNIX manual writeups on these commands for the additional functions provided and details of operation.

IV. FORMATTING

It is preferrable to use nroff(I), rather than the somewhat simpler roff(I), as NROFF is more compatible with troff(I), thus allowing for later use of a phototypesetter. Note that this section should be read in conjunction with reference (5). The features are quite powerful; unfortunately, they may tend to intimidate the casual user. It is actually quite easy to produce good output knowing only a small subset of the commands.

(a) GENERAL APPROACH: when entering text to be formatted later, the first rule to remember is to begin each sentence on a new line, rather than typing them as they may appear in the finished document. Convenient editing is greatly facilitated by this habit, especially when large insertions or deletions are required. Second, it is usually unnecessary (and undesirable) to ever enter text containing strings of multiple blanks. Various formatting commands can be introduced to produce most desired effects automatically, with less use of storage, and increased flexibility.

It is actually quite easy for the beginner to create documents from scratch. First, a file of NROFF macros should be obtained. Such macros are called by the user to produce headers, footers, paragraphs, section numbering, table of contents, footnotes, etc. A very simple file of this kind is discussed later. Some users may wish to create their own specific set of macros. However, a number of sets of "canned" macros exist, offering various different kinds of services and choices of formats. The beginner should probably start with one of these, then customize if necessary. See references (10), (11), (12). A typical sequence of commands for creating a file and formatting it might be:

```
ed
$a
.
w myfile
!nroff macrofile myfile
```

This sequence assumes that macrofile is the name of the chosen file of NROFF macros. The macrofile can automatically be included if desired, using the NROFF command .so (see INCLUDE A STANDARD FILE later in this section).

(b) SAMPLE FILE OF NROFF MACROS: listed below is a very simple file of NROFF macros, which was used to prepare this memo. In order to be explainable in a small space, it is much more restricted than other available macro packages.

```
.de HD
'sp 2
.tl ''\\*(HT'
'sp 2
.ns
..
.de FO
'sp 2
'tl ''\\*(FT'
'bp
..
.wh 0 HD
.wh -6 FO
.de PA
.sp 1
.ti +3
.ne 3
..
.de HG
.sp 1
.in 4
.ti -4
.ne 4
..
```

The set of lines above was used to help produce this document. They provide for headers, footers, paragraphs, and hanging indents (i.e., the format of INFORMATION SOURCES). Briefly, the sequence from ".de HD" to ".." defines the HD macro, to be used as a header. The two ".tl" lines produce titles, which are filled in by user-supplied parameters, using string macros. these string macros are named FT and HT, and are invoked by their appearance following appropriate character escape sequences. They are filled in by user definitions. For example, the following two lines are at the beginning of this file:

```
.ds HT Documentation a la UNIX''unixdocintro
.ds FT '- % -'
```

This combination of macros and definitions produces the same effects as if the ".tl" lines had originally been coded:

```
.tl 'Documentation a la UNIX''unixdocintro'
.tl ''- % -''
```

Note that appearances of the % character inside title lines are magically changed into current page numbers upon output. Also note that NROFF does recognize .ds commands, even though reference (5) does not mention them: see either (6) or (9) for an explanation. Further examples of different headers may be found in reference (5).

The lines from ".de FO" to ".." define a footer macro, which does little but produce a centered page number with appropriate spacing, and then eject a page (the 'bp).

The two ".wh" commands tell NROFF when the HD and FO macros should be called: HD at the beginning (line 0) of each page, and FO 6 lines before the end of each one. Thus, they are invoked automatically, by what are called line traps.

The remaining macros are intended to be invoked at arbitrary points, by typing their names alone on lines, e.g.

.PA

is used to begin a paragraph. It first produces a blank line (.sp 1), then causes the next input line to be indented 3 spaces. The .ne 3 command prevents the next line from going on the current page unless there is room for at least three more. This helps prevent "widows" (lines separated from related ones). The .HG macro is used to produce a hanging indent: it indents all lines (except the next line) 4 spaces. (See INFORMATION SOURCES for this style.)

Using such macros for spacing is very convenient, leads to uniform-appearing documents, and permits ease of adjustment.

Finally, it may be somewhat puzzling that some of the commands began with an apostrophe, while most began with a period. See CONTROLLING FILLING AND ADJUSTMENT in the next section for a short explanation of this.

(c) SIMPLE HOW-TO'S: listed below are some commonly-required operations and NROFF commands for obtaining them.

EJECT TO NEW PAGE

.bp

PREVENT SEPARATION OF HEADINGS FROM THEIR PARAGRAPHS:

.ne n just before heading (see .HG and .PA macros above).

CONTROLLING FILLING AND ADJUSTMENT:

Unless otherwise instructed, NROFF collects words into lines which are adjusted (and hyphenated as needed) to produce uniform right and left margins. If this is not desired (as in tables, program examples, etc), it can be turned off by:

.nf

and turned on again later by:

.fi

Note that this was used to prevent the listing of the macros above from looking like this:

```
.de HD 'sp 2 .tl '\\*(HT' 'sp 2 .ns ..  .de FO 'sp 2 'tl
'\\*(FT' 'bp ..  .wh 0 HD .wh -6 FO .de PA .sp 1 .ti +3 .nc 3
..  .de HG .sp 1 .in 4 .ti -4 .ne 4 ..
```

Suppose that while remaining in fill mode, you want to cause a line to begin at the left margin, rather than possibly being used to fill in the leftover space in the previous one. To do this, a break is required: i.e, cause a line to be output without complete filling. There are several ways to cause a break. First, you may insert the following command between the two lines:

```
.br
```

Second, any line beginning with a blank (or tab character converted to a blank) causes a break. Of course, this does what you normally want: any line explicitly indented will not be used for filling. Finally, some (but not all) commands normally cause a break, at least when begun with a period. (See the NROFF command summary to find which commands cause breaks and which do not). Any command can be made not to cause a break by beginning it with an apostrophe rather than a period. For example, the HD and FO macros use this ability to avoid causing unwanted output of a partial line.

CONTROLLING HYPHENATION:

If hyphenation becomes bothersome for some reason, it is inhibited by:

```
.nh
```

and restored by:

```
.hy
```

More explicit control can be obtained: see reference (5), section VIII.

CENTERING:

The next n lines can be centered by:

```
.ce n
```

and the following causes a general centering mode until turned off:

```
.ad c
```

with return to normal obtained by:

```
.ad n
```

(in this last case, n is not any number, but the character 'n').

DOUBLE SPACE FOLLOWING TEXT:

```
.ls 2
```

RETURN TO SINGLE SPACING:

```
.ls
```

INSERT n BLANK LINES:

```
.sp n
```

A single blank line is equivalent to .sp 1 .

SET UP BLOCK OF n CONTIGUOUS BLANK LINES (FOR FIGURE):

```
.lv n
```

SET LINE LENGTH (WIDTH) TO n COLUMNS:

```
.ll n
```

INDENT NEXT LINE TO THE RIGHT (+), OR LEFT(-) OF CURRENT PLACE:

```
.ti ±n
```

Such indentation is done relative to the current permanent indent, which is set by:

```
.in ±n
```

(relative to previous indent (if + or -), or absolute, if no + or - given). Examine the .HG and .PA macros for typical uses. As another example, below are shown the original and formatted versions of some text:

```
.in 25
.ll 40
.ti+3
This is the beginning of a paragraph which is centered
and filled.
.sp1
.ti -3
This is an example of the hanging indent style.
```

This is the beginning of a paragraph which is centered and filled.

This is an example of the hanging indent style.

SET UP TABLES (USING TABS):

Although you may enter tabs (HT) into a file to be printed by NROFF, NROFF does not normally issue tab characters to the terminal. Instead, it has a table of internal tabs (pseudotabs), which it uses to convert tabs into strings of blanks. Nroff does not know what tabs are set on your terminal, nor does it care, although its initial set of pseudotabs is identical to the normal UNIX ones (gotten by cat /usr/pub/tabs). In any case, if you want to enter tables of information, the following sequence is appropriate:

```
.ev 2
.nf
.ta list of tab numbers
lines of table, with tabs between items
.br
.ev
```

The .ev commands allow the previous set of tabs to be maintained, undisturbed by the .ta . Also, the .ne command may be handy to keep tables from being broken across pages.

UNDERLINE WORDS (OR PHRASES):

A word can always be underlined by typing it, backspacing to the first character, and then underlining it. The .ul command offers a better way (in terms of storage use and compatibility with line filling and hyphenation):

```
.ul
word or phrase to be underlined.
```

This is obviously handy for titles or headers, but can also be used inside sentences, since NROFF assembles filled text without reference to input lines. I.e., this:

```
This sentence has
.ul
an important phrase
in the middle of it.
```

comes out of NROFF as:

This sentence has an important phrase in the middle of it.

SET UP TABLE OF CONTENTS ENTRIES CONVENIENTLY:

The "leader character" (ASCII SOH), can be used to to produce entries for tables of contents like:

```
TOPIC..... 10
SECOND TOPIC..... 20
```

This was produced by the following sequence, with an SOH character (nonprintable) typed between each C and the follow-

ing blank:

```
.ev 2
.ta 63
.nf
TOPIC 10
SECOND TOPIC 20
.ev
```

The .ev commands are used to switch to a different environment, to avoid fouling up the current pseudotab settings.

CHAIN FILES TOGETHER FOR NROFFING:

End each (except last) file with:

```
.nx nextfilename
```

INCLUDE A STANDARD FILE:

```
.so filename
```

causes reading of the named file, then continues with the current file. This is another way to include a common file of macros.

DIFFERENT TITLES OR NUMBERING ON EVEN AND ODD PAGES:

Investigate header/footer macros using .if o and .if e commands.

AUTOMATICALLY NUMBER SECTIONS OR PARAGRAPHS:

Look at reference (5) section VI on number registers.

The list above is just a small subset of the available operations, but should suffice for many documents.

OTHER NOTES AND WARNINGS

Note that all the user-defined macros used capital letters only. Lower case letters may be used, but the writer should be very careful when doing so, since a predefined NROFF command may thus be accidentally erased.
NROFF allows only limited numbers of macro names, string names, etc, but never issues diagnostics when the limits are exceeded. Be warned that peculiar-seeming behavior may result from such action.
Finally, if you want a file which is compatbile with both NROFF and TROFF, it is advisable to use only user-defined macros in the main body of text, rather than invoking NROFF commands directly.

(c) MISCELLANEOUS TEXT PREPARATION AIDS: a permuted index may be generated using ptx(VI): it was used for the UNIX manual index. Program ov(I) is used in conjunction with NROFF to produce multiple-column output.

STARTER PACKAGE

Together with a copy of reference (5) and perhaps (3), the following set of UNIX manual sections make up a reasonable starter package for the beginning UNIX user.

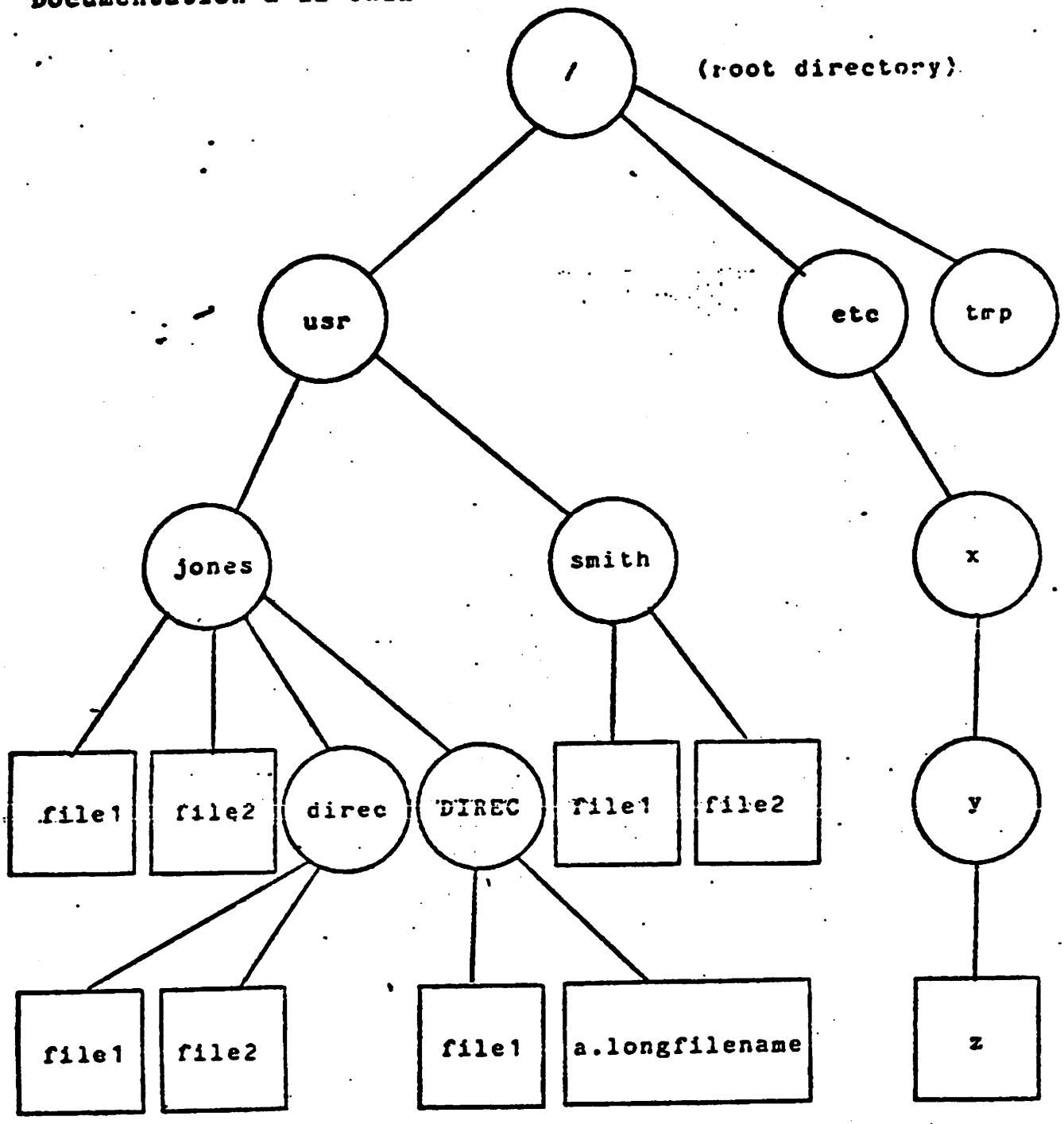
Cover sheet, PREFACE, INTRODUCTION, HOW TO GET STARTED
Index
cat(I)
chdir(I)
chmod(I)
comm(I)
cp(I)
ed(I)
fed(I)
form(I)
login(I)
ls(I)
ln(I)
mail(I)
mv(I)
nl(I)
pr(I)
rf(I)
sd(I)
sh(I)
rm(I)
rmdir(I)
sort(I)
sh(I)
tr(I)
troff(I)
typo(I)
uniq(I)
who(I)
write(I)
hyphen(VI)
ptx(VI)

Acknowledgments: many people have contributed helpful suggestions and comments. The following are especially thanked for their careful readings of previous iterations of this memo: R. Canaday, B. Dickman, R. Haight, E. Ivie, J. Maranzano, and M. Rochkind.

John R. Nashey

RR-9152-JRM-nroff

John R. Nashey



1. Names enclosed in rectangles represent ordinary files.
2. Names enclosed in circles represent directories.

Figure 1. Sample UNIX File Structure

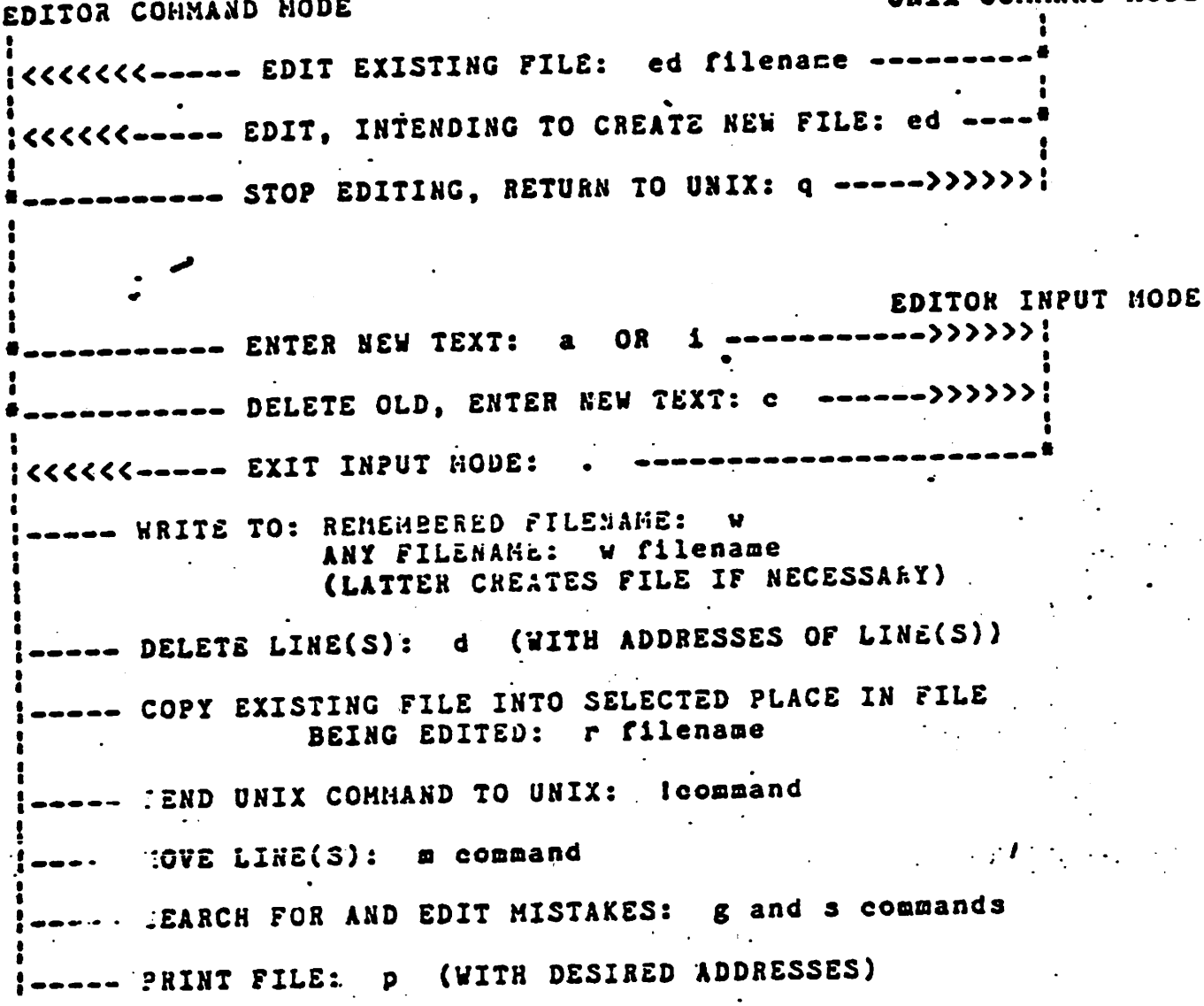


Fig. 2 Ed & UNIX Modes and Actions