UniFLEX®
Utilities
Package I

technical systems
consultants, inc.

UniFLEX®
Utilities
Package I

MANUAL REVISION HISTORY

Revision	Date	Change
A	7/82	Original Release
B	9/85	Revised for release of Version 2.03

DISCLAIMER

The supplied software is intended for use only as described in this manual. Use of undocumented features or parameters may cause unpredictable results for which Technical Systems Consultants, Inc. cannot assume responsibility. Although every effort has been made to make the supplied software and its documentation as accurate and functional as possible, Technical Systems Consultants, Inc. will not assume responsibility for any damages incurred or generated by such material. Technical Systems Consultants, Inc. reserves the right to make changes in such material at any time without notice.

Summary Listing of UniFLEX® Utilities Package I

add-info	:	add information to the info field of a binary file
addusr	:	add a new user onto the system
assert	:	evaluate assertions for shell script programs
bcompare	:	compare binary files and report differences
broadcast	:	send message(s) to all users on the system
checksum	:	get the checksum of a file(s)
compare	:	compare two text files and find differences
continue	:	permits conditional interruption of shell scripts
copy-dir	:	copy directory structures or individual files
cview	:	prints a file in readable form
delusr	:	delete a user from the system
dump	:	gives a hexadcimal and ASCII listing of a file
filetype	:	reports on the type of a file
find	:	finds lines in a text file containing a specified string
flex-copy	:	allows user to copy a UniFLEX file to FLEX diskette
head	:	prints characters from the beginning of a text file
indent	:	indents standard input before writing to standard output
limit	:	limits line length of standard output from standard input
loadsize	:	calculates amount of memory needed to load a binary file
lockterm	:	ignores all input from a terminal for 15 minutes
ls	:	display directory contents or information about a file
news	:	used to get news from the news directory
nice	:	lowers the priority of the command that follows it
nobs	:	removes backspace and following character
noff	:	removes form feeds
pcomm	:	prints all comment lines in assembler source file(s)
plabels	:	prints label lines from assembler source file(s)
remove	:	removes files and directories from system
s1	:	dumps a binary file in Motorola S1/S9 record format
search	:	searches for file(s) based on file name and/or attributes
set_printer	:	set options on the qume or nec printers
sleep	:	puts terminal to sleep for a given number of seconds
split	:	creates a new file from a portion of an existing file
status	:	lists status about programs running on the system
tail	:	prints characters from the end of a text file
tee	:	copies standard input to standard output and file(s)
time	:	times execution of a UniFLEX command
touch	:	sets last update time of file to current time and date
translate	:	maps characters from standard input to standard output
update_all	:	performs a desired function on a series of files
usage	:	lists file usage on a disk by user
validate	:	validates a backup made by "copy-dir"
verify	:	checks specified files or devices for read errors
words	:	finds number of words and lines in file or standard input

INTRODUCTION

The UniFLEX® Utility Package I contains several additional utility commands for use under the UniFLEX Operating System. These utilities range from such functions as word counting to file comparisons to a system status report. The manual contains a complete description of each command and the disk contains UniFLEX "help" files for each command.

This manual may be taken apart and the pages added to the "UniFLEX Utility Commands" section of the "UniFLEX® Operating System" manual. The utilities are listed alphabetically and the pages are one-sided for easy insertion into the UniFLEX manual.

The standard procedure for copying the utilities is to use the "insert" command, which automatically copies all files to the system disk. If a user does not want all these utilities on the system disk, the recommended procedure is to still use the "insert" command, and then selectively delete those files not needed.

add-info

The add-info utility adds information to the info field of a binary file.

SYNTAX

```
add-info file ...
```

DESCRIPTION

This utility allows the user to add information to the info field of a binary file. The add-info utility reads standard input until a standard end-of-file (control D) is encountered. Each line separated by a carriage return is stored in the info field as an individual line.

If more than one file is in the argument list then the information to go into the first file must be terminated by an end-of-file before the information for the next file is entered.

This utility cannot be used to add information to a file which already has an information field. If a file already has an info field then a message will be printed to standard output indicating that fact.

A few example calling lines of add-info follow:

```
add-info file1
add-info file2 file3
```

In the first example, standard input is read until an end-of-file is encountered and then the information is stored in the info field of 'file1'. The second example also reads standard input. When an end-of-file is encountered it will store the first set of information into the info field of 'file2' and continue to read standard input. When another end-of-file is encountered it will store the information into the info field of 'file3'.

SEE ALSO

info, asmb

addusr

Add a new user onto the system.

SYNTAX

addusr <"username">

DESCRIPTION

This utility is used to add a new user onto the system. The specified user name must be unique, unused and between 1 and 8 lower case characters. Only the system manager may use this utility.

The new user name is added to the bottom of the "/etc/log/password" file and a new user id number is assigned. The new user's directory "/usr/username" is created with the permissions 'rwxr-x' and a '.mail' file is automatically inserted.

The system manager or the new user should use the password utility to ensure protection of the new user's personal files.

SEE ALSO

perms, password, mail, delusr

assert

Evaluate assertions for shell script programs.

SYNTAX

```
assert [options]
```

DESCRIPTION

The assert program evaluates the truth value of the given options. For example, if a program should only be run by the system manager, i.e. a back up program, "assert u=system", would prevent any other user from executing that program. Assert is used to interrupt or abort shell scripts.

Options Available:

k=<string>	- Display string and wait for input.
m	- Check for multi-user mode.
s	- Check for single user mode.
u<number>	- Check for current user number.
u=<name>	- Check for current user name.

If the k option is used, the entire option must be enclosed in single or double quotes (see example below).

EXAMPLE

```
assert u=system "k=Insert root structure disk in FD1"
```

In the above example, assert will make sure the user is "system" and echo to the screen "Insert root structure disk in FD1", then wait for the user to type RETURN. If the user is not "system", assert will cause the program to abort and return to shell.

SEE ALSO

continue

broadcast

The broadcast utility will send a message to all users logged on the system.

SYNTAX

```
broadcast [file ...]
```

DESCRIPTION

This utility allows the user to directly communicate with all users logged on the system who have not locked out messages. Only the system manager may broadcast messages to users who have locked out messages. This command will send the contents of the file(s) to each user. If multiple files are specified, they will be listed one following the other, as if they were one file. Broadcast without a file name will take its input from standard input until and end of file is encountered. If a file name argument is encountered which consists of a single plus sign ('+'), the standard input will also be read. The following are some examples of the broadcast command:

```
broadcast message1
broadcast message1 message2
broadcast message1 message2 + message3
```

The first example will send 'message1' to all users on the system who do not have messages disabled. The second example will send message1 and message2. The last example will send message1 and message2, will then read standard input until an end of file is encountered and then will send message3.

SEE ALSO

message, send

bcompare

Compare binary files and report differences to standard output.

SYNTAX

bcompare <file1> <file2>

DESCRIPTION

The bcompare utility is used to compare two binary files. Bcompare reports the block numbers and bytes that differ between the two files.

Differences are output in the form:

<file offset> --> <left byte> : <right byte>

where "file offset" is the position in the file where the difference occured, "left byte" is the contents at this position in file1 and "right byte" is the contents at this position in file2.

Note that the file offset is the offset from the beginning as the file is stored -- it is not the actual offset from the start of data since some UniFLEX type files have header information before the data begins.

While this is used mainly for comparing binary files, it actually can be used to compare any type of file since the comparison is done byte by byte. Therefore, this utility could be used to compare text files. However, for that purpose, it is recommended that the "compare" utility be used.

If a series of bytes is deleted, and bytes which follow are changed, the rest of the entire file will be reported as changed.

Example:

```
++ bcompare file.b file.b.old
```

SEE ALSO

compare

compare

The compare command is used to compare two text files to see how they are different.

SYNTAX

```
compare file1 file2 [+window size]
```

DESCRIPTION

This utility compares two text files and indicates how they are different. The information provided is sufficient in most cases to change 'file1' into 'file2'. The optional 'window size' indicates how many consecutive lines must match in order to consider the file positions synchronized. The default is 3 and this is a decimal number option.

The compare utility will report on sets of lines which have been deleted, inserted or changed. If a set of lines have been deleted and the line which follows was changed, all lines will be reported as changed.

Compare can handle any size files, but will only handle up to 300 lines per change. A few example calling lines for compare follow:

```
compare old-file new-file +5
compare tree tree.bak
```

The first example will compare 'old-file' to 'new-file'. At least five consecutive lines must match in order to synchronize the file positions. The second example compares 'tree' file to its backup file, 'tree.bak'. In this case only the default of three consecutive lines must match for file positions to be synchronized.

checksum

The checksum utility writes the checksum of a file to standard output.

SYNTAX

```
checksum [file ...]
```

DESCRIPTION

This utility will write the checksum of a file to standard output. Checksum is very useful when checking the accuracy of a transmitted or copied file.

If no file is given as an argument, checksum will get its input from standard input. It will terminate reading standard input when given the standard end-of-file (control D) from the user's terminal.

If more than one file is listed in the arguments then the utility will print the checksum of each file on a seperate line following the order of input (left to right).

The checksum is calculated as a simple adding up of the integer values of the characters in the file.

Some examples of checksum follow:

```
checksum
checksum file1
checksum file1 file2
```

The first example will read standard input until an end-of-file is reached and will then report the checksum. In the second example, "file1" will have its checksum reported to standard output. The third example will print the checksum of "file1" on one line, and on the next line will print the checksum of "file2".

The continue command is useful for displaying diagnostic messages to the terminal and interrupting long shell scripts.

SYNTAX

```
continue [+hex] [argument]
```

DESCRIPTION

This utility will echo (display) the command arguments on standard output and input a one character response from standard input. If the response is a 'Y' or 'y', or end-of-file, continue will set a termination code of zero and allow the remainder of the shell script to be executed. A 'N' or 'n' response will set a termination code of 255, abnormally terminate the shell script, and return the termination code to the shell. If any other response is made the command will simply display the arguments again.

There are several options available under this command. If the argument starts with a plus sign ('+'), it is not echoed. If the value following the plus sign is a hexadecimal number (maximum of hex 7f), the equivalent byte will be output. This is useful for producing special control character sequences.

Continue will display all the arguments, and then will accept the one character from standard input. A few example calling lines of continue follow:

```
continue +7 Continue processing?
continue "Insert a new disk.  Type 'y' when ready"
continue +d +a Ready to start?
```

The first example will output a bell character (7) and the message, and then accept a one character response from the terminal. The message in the second example contains a character which the shell interprets a special character (namely '), thus making it necessary to enclose the entire message in double quotes ("). The last example will output a carriage return (d), and a line feed (a) before issuing the message and processing the response.

SEE ALSO

echo, shell

copy-dir

The copy-dir command copies directory structures or individual files to the destination directory.

SYNTAX

copy-dir [file or directory ...] dest-directory [+dbncotBplLD]

DESCRIPTION

This utility allows the user to copy directory structures or files to another directory. The destination-directory must be completely specified. If no source file or directory is specified then the current working directory is copied to the destination-directory. Which file(s) and/or directory(s) are copied can be altered by using any of the following options:

- +d Copy entire directory structure for all named directories.
- +b Do not copy a file unless it already exists in destination.
- +n Copy a file if it is newer than the one at the destination. If no file exists, the copy will be performed.
- +c Do not copy file if it already exists at the destination. This option may not be used in conjunction with +n.
- +o Retain original file ownership. This option may only be used by the System Manager.
- +t Do not create top level directories at the destination. Lower level directories may be created, if other options allow. In other words, if there are three top level directories in the source and only one of those directories exists in the destination, then only the one that already exists will be copied when the +t option is in effect. This is useful when the user wants to copy files to a floppy disk and each top level directory has a lot of lower level directories and files.
- +B Do not copy any files which end in ".bak".
- +p Prompt the user to see if he really wants each file copied.
- +l List file names as they are copied.
- +L Don't unlink the destination file. Useful when updating linked files.
- +D Implicitly specify the high level directory names. When this option is given, source files which are directory names will have that name appended to the destination directory name before the process starts. Example:

  ++ copy-dir bin /usr2 +D

  copies "bin" to "/usr2/bin".
- +P Preserve the modification time of the source file. This allows the user to give the destination file the same modification time as the source file.

With all the options a user has a lot of flexibility in what to actually copy with copy-dir. A few examples of copy-dir follow:

```
copy-dir /usr/bin
copy-dir file1 file2 /usr/bin +n
copy-dir /usr/bin +B
copy-dir /usr/gen /usr/bin +l
copy-dir /usr /usr2 +d
copy-dir /usr/all /usr2/usr/all +td
copy-dir gen /usr2 +D
```

The first example copies all the files in the current working directory to "/usr/bin". The second example will copy "file1" and "file2" to directory "/usr/bin" only if the two files don't already exist in "/usr/bin" or if they are newer than the files that do exist there. The third example will copy all files in the current working directory except those ending in ".bak" to the directory "/usr/bin". The next example will copy all the files from directory "/usr/gen" to directory "/usr/bin" and list the file names as they are being copied. The following example will copy all files and all directories including all lower level directories and their files from "/usr" to "/usr2". For the next example we assume that the only directory in "/usr/all" that also exists in "/usr2/usr/all" is "dira". The files of "/usr/all/dira" will be copied to "/usr2/usr/all/dira". All lower level directories of "dira" will also be copied because of the +d option. The final example will copy directory "gen" to "/usr2/gen". One thing about this option is that you cannot use it to create directories with very complicated path names. If instead of "gen" the user wanted to copy "gen/all/files" to "/usr2", then the directory path "/usr2/gen/all/files" would have already had to exist. If it did not then the copying would not be done. The other example works even if "/usr2/gen" did not exist because copy-dir can be used to create simple path names.

SEE ALSO

copy

cview

The cview utility prints a file in readable form.

SYNTAX

```
cview file
```

DESCRIPTION

This utility removes the parity bit from all characters in the file. If the resulting ASCII character is printable, it is written to standard output.

Control characters (00 to 1f) are printed as "^x" where x is the appropriate printable character (corresponding ASCII character). For example:

carriage return, 0D, is printed as ^M.

A maximum of 72 characters (including the "^" for control characters) are printed per line.

A few example cview calling lines follow:

```
cview file1
cview file.b >readable
```

In the first example file "file1" is printed to standard output. The second example has "file.b" (a binary file) being stored in "readable" in a readable format (the ">" means send standard output to the file following).

delusr

Delete a user from the system.

SYNTAX

delusr <"username">

DESCRIPTION

This utility is used to remove a new user from the system. The specified user name muse be between 1 and 8 lower case characters and currently in use on the system. Only the system manager may use this utility.

The specified user name will be removed from the "/etc/log/password" file and the user's directory(s) and file(s) will be DESTROYED.

WARNING

Use extreme caution when using this utility. It can and will recursively descend the user's directory tree and delete all files within it.

SEE ALSO

addusr

dump

The dump utility gives a hexadecimal and ASCII listing of a file.

SYNTAX

```
dump file [+i]
```

DESCRIPTION

This utility will print the hexadecimal and ASCII listing of a file side-by-side to standard output. Nonprintable characters are replaced by a period ('.') in the ASCII listing.

If the '+i' option is used then dump will enter an interactive mode and prompt for the address or addresses to display. The addresses are relative to the first byte of the file, and must be entered in hexadecimal. If a single address is specified, 16 bytes will be displayed beginning at that address. It is also possible to display a range of addresses by specifying a starting address followed by a hyphen followed by an ending address. For example, the following would display 256 bytes starting at byte 0 of the file: "0000-00FF".

To end the interactive mode just type a return and the UniFLEX prompt will appear.

Following are some examples of dump:

```
dump file1
dump file2 +i
```

The first example will cause a side-by-side hexadecimal and ASCII listing of "file1" to standard output. In the second example, dump will request an address and when it receives one will print the hexadecimal and ASCII lising of that line to standard output.

filetype

The filetype utility reports to standard output the type of a file.

SYNTAX

```
filetype file ...
```

DESCRIPTION

This utility allows the user to determine the type of a file. If more than one file is input as arguments, then the messages about the files will be listed on seperate lines with each message corresponding to the files in the order of input (left to right).

The filetype utility will attempt to recognize a file, but will not always be successful, and may in fact give the wrong type. The following list of files are the types of files that filetype will attempt to recognize:

```
text processor text
Pascal text
C text
BASIC text
BASIC precompiler text
assembler text
original BASIC compiled
current BASIC compiled
standard Pascal binary
system Pascal binary
sort/merge parameter
relocatable binary
shared binary text
segmented, no text binary
common block binary
standard binary
```

A few examples of filetype follow:

```
filetype file1
filetype file1 file2
```

In the first example the type of "file1" is determined and reported to standard output. The second example also determines the types of "file1" and "file2". "File1" will be reported on the first line and "file2" will be the second message.

find

Search for text pattern in a file or standard input.

SYNTAX

```
find [+options] "pattern" file ...
```

DESCRIPTION

The find utility matches the specified pattern. Each matching line found and its line number is copied to standard output.

By default, lower case letters match only lower case letters and upper case letters match only upper case letters.

Options Available:

+u All lower case letters in the pattern will match both upper and lower case letters in the file being searched.

+c The output of matching lines will be suppressed and a count of the matching lines found will be reported.

Pattern:

A group of characters have been assigned special meanings, these characters are matching characters. The backslash (\) character preceding a matching character "turns off" the special meaning of that character. Care should be taken when using matching characters in the pattern as they are also meaningful to the shell. Therefore, the pattern must be enclosed in single or double quotes (for example, "pattern" or 'pattern').

Matching characters

\ - The backslash character, when used before a matching character, removes the special meaning from the character.

? - The question mark matches any character except a newline.

< - The less than character matches at the beginning of lines. It is special only if it is used as the first character in the pattern string.

> - The greater than character matches at the end of lines. It is special only if it is used as the last character in the pattern string.

& - The and character matches only if the subpattern before it and the subpattern following it are both in the line being searched.

(str | str) - The vertical bar matches if either subpattern is in the line being searched.

Character Classes

A character class is a text pattern that matches any single character from the bracketed list of characters. Character classes consist of ranges such as "A-Z", "b-f" or "0-8", and may also contain lists of characters or combinations of both.

The negation (!) of a character class will match any character not contained in the character class.

The syntax for character class is as follows:

[list or range of characters]

while the syntax for the negation of character class is:

[! list or range of characters]

flex-copy

The flex-copy command allows the user to copy a UniFLEX file to FLEX diskette.

SYNTAX

```
flex-copy source-file dest-file [drive-spec] [+bi]
```

DESCRIPTION

This utility copies the source-file to the dest-file on a FLEX diskette. Destination file specifications may contain filename extensions and if the file already exists on the diskette, it will automatically be deleted. The dest-file name will automatically be mapped to upper case. The FLEX disk may be single or double sided but may not be double density.

There are several options which can be specified for this command. They are as follows:

+b	Copy an absolute binary program in a format which can be executed by FLEX. This is very useful for developing programs on UniFLEX and then running them on FLEX. If you use this option, the source file must be an absolute binary file.
+i	Copy the file with no modifications (make an image copy). This is useful for copying absolute data files.

If the file is a text file there is no need to specify any option.

Normally, the copy is done to drive 1, but drive 0 can also be used by using a command similar to:

```
++ flex-copy source dest 0
```

This is a very powerful utility. A few examples follow showing how flex-copy is used:

```
flex-copy letter letter
flex-copy file.b file.b +b
flex-copy text text 0 +i
```

The first example will copy file 'letter' to 'letter' on the FLEX diskette. The second example will copy the file 'file.b' to file 'file.b' in an absolute binary format that FLEX can execute. The final example will copy file 'text' to file 'text' on drive 0 in an image copy format.

head

The head utility will print a given number of characters from the beginning of a text file.

SYNTAX

```
head file [n]
```

DESCRIPTION

This utility will print the first "n" characters of a text file. If "n" is not specified, the default of 250 characters is used. If "n" characters from the beginning of the file happens to fall in the middle of a line, the last line printed will be succeeded by "..." to indicate that only a portion of the line is being printed. All other lines will be printed as they appear in the file.

If "n" is greater than the number of characters in the file, the entire file is printed.

Special characters, including carriage returns, should be included in the count "n" otherwise not as much of the file may be printed out as is desired. A few examples of head follow:

```
head test1
head file1 150
```

The first example will print out the first 250 characters of the file 'test1'. The second example will print out the first 150 characters of the file 'file1'.

SEE ALSO

tail

indent

The indent utility will indent every line of standard input a given number of blanks before writing to standard output.

SYNTAX

```
indent n
```

DESCRIPTION

Indent must be given a value of "n", where "n" is an integer greater than zero. Only integer numbers are allowed in "n".

A common use of this utility would be to indent output that is going to be routed to a hard-copy printer. This utility is a very useful filter for that purpose.

A few calling lines of indent follow:

```
indent 5
page report ~ indent 5 ~ spr
```

The first example will indent 5 blanks before writing standard input to standard output. The second example will have the page utility work on the file 'report', then filter that through indent and indent the entire file 5 spaces, and finally pipe the output to the spooler 'spr' for hard-copy printout.

limit

The limit utility will read standard input but limit the line length before writing the line to standard output.

SYNTAX

```
limit [n] [+t]
```

DESCRIPTION

This utility will read standard input and limit the line length to "n" characters. If "n" is not specified the default line length is 72 characters. The one option for this utility follows:

+t Truncates any characters exceeding "n" rather than wrapping the excess characters to a new line.

A few examples of limit follow:

```
limit
limit +t
limit 50 +t
limit +t <infile
```

The first example will limit the line length to 72 characters. For lines longer than 72 characters, the 73rd character will start a new line. The second example will also limit the line length to 72 characters. However, lines longer than 72 characters will be truncated. The third example sets the line length to 50, and truncation will occur on lines longer than 50 characters. The last example will use as input "infile" and will truncate all lines longer than 72 and print the file to standard output

loadsize

The loadsize utility calculates the amount of memory required to load a binary file.

SYNTAX

```
loadsize [file ...]
```

DESCRIPTION

In addition to the loadsize, the size of the segments in a shared text file are also given. If a "segmentation size" is reported, it is the number of bytes which needs to be reserved after the text segment in order that the data segment starts on a 4K boundary. For absolute files, only the load size is printed to standard output.

The sizes printed will not reflect additional memory which may be requested during the execution of the program. A few example calling lines for loadsize follow:

```
loadsize test1
loadsize test1 test2
```

The first example will print out the loadsize of file 'test1'. For the second example, if file 'test2' is a shared text file, the segmentation size of file 'test2' will be printed to standard output in addition to the loadsizes of both files.

lockterm

The lockterm utility will lock a terminal (ignore all input) for 15 minutes.

SYNTAX

lockterm

DESCRIPTION

Lockterm will prompt the user for a password. This password must be reproduced in order to unlock the terminal before the specified time limit is expended. All characters typed on the keyboard will be ignored except a Control C. When Control C is typed, lockterm will prompt the user for the password. If the passwords do not match, lockterm will continue to lock the terminal for the remainder of the time limit. If the passwords match, lockterm will unlock the terminal and terminate.

A reason for using this might be if you are in a multi-user environment and you will be away from a terminal for a moment and you wish to prevent someone else from taking over your terminal.

ls

The ls command is used to list the contents of directories, or to give a long directory listing of a single file.

SYNTAX

ls [file or directory ...] [+abdflrstS]

DESCRIPTION

The ls command will list the content(s) of the specified file(s) and/or directory(s). If no file or directory is named as a command argument, the contents of the current working directory will be displayed by file name only, alphabetically sorted, and several names per display line.

If a file name is specified, the name can use matching characters as in shell. In this way a series of files with the same root name can be displayed. If a file name is given, a long listing will always be displayed.

There are several options available to change the standard display for file(s) or directory(s). The options are as follows:

+a	List all files. Normally files which begin with '.' are not listed.
+b	List file sizes in bytes. This implies +l.
+d	If a listed file is a directory, also list its contents. This allows the entire directory structure to be listed.
+f	List the FDN number for each file. This implies +l.
+l	Print detailed information about each file.
+r	Reverse the sense of any sorts.
+s	List files in one name per line format. This is useful for creating a file which has the names of files in it.
+t	Sort files by last modification time. The most recently modified files are listed first.
+S	Print summary information after all files have been listed.

Ls will normally not display file names which start with a period ('.'). The +a option will allow those files to be displayed as well. The +l option provides great detail about each file name in the directory. The files are listed one per line with the file name appearing first on the line, unless the +f option is used in which case the FDN number of the file appears first. Following the name is the file size in blocks, or bytes if the +b option is specified.

The next field specifies the file type as defined by the following:

b	block special file (device)
c	character special file (device)
d	directory
blank	regular type file

The majority of files will be regular files and therefore have this field blank.

The next field contains the permissions associated with the file. The permission field has six columns, the first three represent the user's permissions (owner), the second three represent all others' permissions. The columns are in the order 'rwx' which are described below:

r	read permission granted.
w	write permission granted.
x	execute permission granted.
-	the permission for this field is denied.

The next field is the link count and specifies how many directory entries are linked to this file. Following that is the file owner. An owner's name is the same as his (her) login name. The final information provided is the time and date of the last modification made to the file. An example of a long ls listing might be:

```
5145 ls          3558   rw-rw-  1    system  15:53 Apr 28,1982
5230 macros       105   rw-rw-  1    system  14:47 Apr 28,1982
```

The FDN number for file 'ls' is 5145 (+f option) and the number of bytes in the file is 3558 (+b option). For file 'macro' the FDN number is 5230 and there are 105 bytes in the file. Both files are regular files, allow the owner and others read and write permission, and have a link count of 1. The files' owner is 'system'. 'ls' was last modified was 15:53 on 28 April 1982.

.... continued on next page ....

A few example calling lines for ls follow:

```
ls
ls +lt
ls +l +t
ls +d
ls +S
ls / /usr +l
ls file* +B
```

The first line will display all file names in the current working directory (with the exception of those starting with '.'). The next two examples will give a long listing of the working directory with the files sorted by modification time. These two commands are functionally equivalent. The fourth example will display all files in the current directory, and if any of those files are directories themselves they will also have their contents displayed. The fifth example will display all files in the current directory and then print out summary information about them. The next line will display the contents of directory '/' (the root) and the directory '/usr'. Both listings will be in the long format. The last example will give a long listing of all files which start with 'file' except those files which end in '.bak'. Note that the options may be anywhere on the command line.

SEE ALSO

dir

The news command is used to get news from the news directory.

SYNTAX

```
news [file ...]
```

DESCRIPTION

If called with no arguments, a directory of "/usr/news" is printed and a prompt issued. A carriage return typed in response to the prompt terminates the news session.

If called with arguments, they are assumed to be the names of files in the news directory, and their contents are printed. If there is no file corresponding to the argument file, then a message to that effect is printed to standard output.

News files are created and edited by the system manager or any other user who wants to make some information available to all of the other users on the system. To create and/or edit a news file just change directories to "/usr/news".

New information should be added to news files at the beginning of the file so that the user does not have to wait for the old news to be printed.

A few examples of how the news utility is used follow:

```
news
news mine
news mine yours
```

The first example will display the directory of "/usr/news" and prompt the user about which file should have its contents displayed on standard output. The second example will assume that the file 'mine' is in the directory of "/usr/news" and will display its contents. The third example will assume that both files 'mine' and 'yours' is in directory "/usr/news" and will display the contents of both files.

nice

The nice utility lowers the priority of the command that follows it.

SYNTAX

```
nice "UniFLEX command"
```

DESCRIPTION

This utility is used to lower the priority of the UniFLEX command that follows it. A good use of the nice command would be when the user has to run a very long compilation and there is no urgency. By using the nice command the priority of the task would be lowered. That would allow jobs to run which need to be executed immediately, but the task would still be compiling when nothing else was running.

Some examples of nice follow:

```
nice ls
nice pascal test1.p
```

The first example will lower the priority of the command "ls". The second example will lower the priority of the running of the pascal job.

nobs

Remove backspace and following character from an output stream.

SYNTAX

```
... ^ nobs
```

DESCRIPTION

The nobs utility removes the backspace and the following character from an output stream possibly containing backspace characters. Characters removed are "thrown away". This filter is for use on the Centronix or other printers that do not support backspaces.

This utility reads standard input and writes to standard output and is most likely to be used as a pipe which gets its input from some UniFLEX command and sends its output to a spooler or printer.

noff

Remove form feeds from an output stream.

SYNTAX

```
... ^ noff
```

DESCRIPTION

The noff utility takes an output stream possibly containing form feeds and removes them. The filter is used for printers that do not support form feeds. The form feeds removed are "thrown away".

This utility reads standard input and writes to standard output. It is useful as a pipe which gets input from a UniFLEX command and sends the output to a spooler or printer through another pipe.

The pcomm utility prints to standard output all comment lines in an assembler source file.

SYNTAX

```
pcomm file ...
```

DESCRIPTION

This utility will search an assembler file and print out all the comment lines (those lines which have an asterisk in column one) to standard output.

The following are some examples to illustrate the pcomm utility:

```
pcomm test1
pcomm test1 test2
```

The first example will print out all of the comments from assembler source file 'test1'. The second example will print out all of the comments of both files 'test1' and 'test2'.

SEE ALSO

plabels

The plabels will print to standard output all lines in an assembler source file which contain a label.

SYNTAX

```
plabel file ...
```

DESCRIPTION

This utility is used to print all lines from an assembler source file which contain a label in column one to standard output. The line is preceded by its line number. Lines with local labels (labels beginning with a numeric digit) in column one will be ignored.

The following examples will illustrate the plabels utility:

```
plabels test1
plabels test1 test2
```

The first example will print to standard output all of the lines which have labels in them from file 'test1'. The second example will do the same but for both files 'test1' and 'test2'.

SEE ALSO

pcomm

remove

Remove files and directories from the system.

SYNTAX

remove <file or directory ...> [+options]

DESCRIPTION

The remove utility removes the specified files or directories from the file system. If a file has no write permissions, it will not be deleted unless the "+w" option is specified. Directories will only be deleted when the "+d" option is specified and the directories are empty, or if the "+k" option is specified.

Options Available:

+p - Prompt about each file individually.
+l - List the files as they are deleted.
+w - Ask about files that have write permission.
+d - Delete directory if empty.
+k - Delete directory and all files in that directory.

EXAMPLE

++ remove fool +lw

In the example, remove will delete all the files contained in the directory "fool". Remove will list each file deleted ("+l") and will ask about files that don't have write permission ("+w"). Remove will delete files that don't have write permission only if the prompt is answered with "y".

SEE ALSO

kill

The s1 utility will dump a binary file in Motorola S1/S9 record format.

SYNTAX

```
s1 file ...
```

DESCRIPTION

This utility is useful for down loading binary programs to other machines or transferring binary files across a serial line.

If more than one file is included both files will be listed to standard output together.

The following is an ASCII hex record of the standard Motorola format:

```
S1BBAAAADDDDDD.....DDDDCC
```

where

```
  S1 is a record start marker
  BB is a byte count which includes all bytes in the record past
     the byte count itself
AAAA is the load address of the first data byte
  DD is the actual data
  CC is a checksum equivalent to the one's compliment of the modulo
     256 sum of all bytes preceding the checksum (except S1)
```

A few examples of s1 calling lines follow:

```
s1 file1
s1 file2 file3
```

The first example will print the s1 dump to standard output of "file1". The second example will do the same for files "file2" and "file3" and they will be printed out one right after the other without a break.

search

Search for file(s) based on a file name and/or a series of attributes.

SYNTAX

search [filename] [attributes]

DESCRIPTION

The search utility is used to search through a directory or a series of directories for a file or set of files. The utility starts searching in the user's current directory (or a specified directory given by the p option) and searches through that directory and all directories within that directory. The utility continues until all subdirectories have been searched. As output, the search utility will print out in alphabetical order the name of any file(s) which match the given filename or specified attributes. If the file is a directory, then it will again have its name printed out when the utility searches through it to try and match more files. The output can be changed by several of the options.

The search is based either on a filename, a set of attributes, or both. However, either a filename or an option must be given or an error will occur. If matching characters are used in the filename, then the filename must be enclosed in double or single quotes (for example - "*.bak").

The options can be preceded by either a '+' which indicates an "and" condition or by a '-' which indicates an "or" condition. If no filename is given the first option must be preceded by a '+'. Different options must be separated by blanks and each new option must be preceded by either a '+' or '-'. The general form of an option follows:

(+ or -)letter(operator)operand

where letter is any valid option, operator is any valid operator as described below, and the operand is dependent on the option. The parenthesis in the above expression are included for readability and must not be used in the utility command line.

The valid options are n, p, f, t, u, s, d, l and x and they are described on the following page. The valid operators are the equals ('=') operator, the not equals ('#') operator, the greater than ('>') operator and the less than ('<') operator.

For several of the options (n, p, f and x) it does not matter whether a '+' or a '-' is used, subject to the constraint that the first option must be a '+' if no filename is used. For these four options a '+' and a '-' are treated identically.

A description of the possible options is as follows:

n — No listing is produced. This option does not require an operator or operand.

f — Print the full path name of the file. This option also does not require an operator or operand.

p — The pathname that is given is used instead of the current directory. The only operator allowed is the equal operator. Other operators will assume an equal operator.

t — Only files which match the given file type are processed. Valid types are f (regular), d (directory), b (block) or c (character) files.

u — Only those files belonging to the user specified will be processed. If the user name is not a valid user name (the user name cannot be found in "/etc/log/password") then it will cause an error. The maximum number of user names that may be used during a search is four for each condition (four for the "and" condition and four for the "or" condition).

s — The files with the specified size in blocks will be processed. If this option is specified, block and character files will be ignored.

d — The files with the specified time in days (since the last modification) will be processed.

l — The files having the specified link count will be processed.

x — This will cause the utility to execute the command(s) following the equals operator. The equals operator is the only valid operator. Other operators will produce an error. Only one x option is allowed, and the entire option must be enclosed in double or single quotes ("+x=echo &" for example). If the user wishes to execute more than one command, then it may be done by using the standard multiple statements per line separater (';') within the operand of the x option. An '&' may be used in the command line in place of the normal file specification. It will be replaced by any filename(s) produced by the search.

For both the t and u options only the equal and not equal operators are valid. Other operators will assume an equal operator.

If both a greater than operator and a less than operator are used for the same option (s, d or l options only), then the lower limit must be less than the upper limit or an error will result. The s, d and l options also will only take one equals or one not equals operator. If more than one is included then an error will occur.

A few example calling lines for search follow:

```
search filename +p=/usr +t=f +f
search +n +s>20 +l#2 "+x=ls & +l"
search +d>2 +d<10 -s>30
search "*.bak" +p=/ +u=system "+x=kill &"
```

The first example will search for the file 'filename' in the directory /usr and all of /usr's subdirectories. If it finds a file called 'filename' and the file is a regular file (t option), then the full path name of the file will be printed (f option). The second example starts in the user's current directory and looks for all files where the size of the file is greater than 20 blocks and the file's link count is not two. If it finds such a file then it will print the long listing of the file by executing 'ls'. In the third example all files will be printed that are greater than two days old and less than ten days old, or all files that are greater than 30 blocks in size. The last example starts searching in the root for all files that end in '.bak' and that are owned by 'system'. Any files that are found will be killed.

set_printer

Set options on the QUME or NEC printers.

SYNTAX

```
set_printer <printer> [+ <options>]
```

DESCRIPTION

The set_printer utility enables the user to set options on the Qume or NEC daisy wheel printers. Note that set_printer cannot change options while the printer is in use.

The options consist of an option letter, followed by a decimal value. Some values have optional fractional parts as shown by the ".nn" in the option table.

Upon the execution of set_printer, all values are reported.

Options Available

```
d=nn        -- set depth (number of lines per inch)
f=nn.nn     -- set form length value in inches
               (zero indicates infinite form length)
m=nn.nn     -- set left margin value in inches
p=nn        -- set pitch (number of characters per inch)
w=nn.nn     -- set form width value in inches
               (zero disables form width checking)
```

This utility will only work with a Southwest Technical Products interfacing board for the qume or nec printers. Set_printer won't work if the device is connected to a standard serial board.

EXAMPLE

```
++ set_printer
Depth        - 6 lpi (lines per inch)
Pitch        - 12 cpi (characters per inch)
Left Margin  - 1 "
Form Width   - 14 "
Form Length  - 11 "
```

If no options are specified, set_printer returns the present values of the parameters.

sleep

The sleep utility puts the user's terminal to sleep for a given number of seconds.

SYNTAX

```
sleep [n]
```

DESCRIPTION

This utility is used by giving a value for "n". The terminal will not accept any more input for "n" seconds. This means that anything the user types in during the sleep time will be ignored until "n" seconds have past. At the end of "n" seconds a UniFLEX prompt ('++') will be displayed and the user may again use the terminal. If the user has typed something in during the sleep period of time then the shell will attempt to execute those commands.

If no "n" is given then a UniFLEX prompt will appear immediately.

A few example calling lines of sleep follow:

```
sleep 10
sleep 45
```

The first example will put the terminal to sleep for 10 seconds and the second example will put it to sleep for 45 seconds.

split

The split utility is used to create a new file from a portion of an existing file.

SYNTAX

```
split source-file [destination-file line-number-range ...]
```

DESCRIPTION

The line-number-range indicates which lines from the source file are to be extracted and copied to the destination file. A line-number-range is a pair of line numbers separated by a hyphen (starting line-ending line). If the "starting line" is not specified, the beginning of the file is assumed. If the "ending line" is not specified, the end of the file is assumed.

```
split big-file small-file 1-10
split prog-library program 158-792
split letter body -75 salutation 76-
```

The first example will copy the first 10 lines from big-file into small-file. The second example will copy lines 158 through 792 inclusive from prog-library into program. The last example will copy the first 75 lines of letter into body, and from line 76 to the end of the file into salutation.

status

The status command will list status about programs running on the system.

SYNTAX

```
status [+alxw]
```

DESCRIPTION

This utility determines what tasks are running on the system, but is not always accurate due to the dynamic nature of the system. Following are some options giving the user control over which tasks are listed:

+a	List all tasks, not just those belonging to the user.
+l	Produce a long, detailed list for each task listed.
+x	List every task in the system. Normally, only "interesting" tasks (anything but shell, "System" or "init") are listed.
+w	Wait after listing (for about 30 seconds) and then produce another listing. This action continues about 100 times.

Normally the status command simply prints out the Task-id, Mode, tty, Prio (priority), Time and Command. Those fields are:

Task-id
: This is the number given to the task by the operating system.

Mode
: The "c" status means the task is in core (memory) and "s" means it is swapped out to the disk.

tty
: This indicates which terminal the task originated from. If there is an "xx" in the field it means no terminal.

Prio
: If the priority is a number, then the number indicates the priority of the task. Higher numbers indicate higher priorities. Other possible priorities are:

slp	the task is sleeping (not executing)
wait	the task is waiting for some other task to complete
out	task is waiting for output to terminal to complete
in	the task is waiting for input from the terminal
pipe	task is waiting for pipe data (usually input)
buf	the task is waiting for a system buffer
disk	task is waiting for some disk activity
file	the task is waiting for some file activity (rare)
upd	task is updating FDN
sys	this is the highest possible priority available
swap	the task is being swapped out to the disk

Time
This is the total user and system time. The time for "System" is the amount of unused CPU time.

Command
The command field indicates the command which originated the task. This only will print out the first 35 characters of the command with any additional characters truncated. If the +l option is used, then the field is only 18 characters with the remaining characters truncated. The "System" seen in this field is the operating system and "etc/init" runs the login program.

When the +l option is used several other fields are included:

Status
The possible values of status are as follows:

run	the task is running
sleep	the task is waiting for something to happen
term	the task has terminated

User
The user who originated the task.

Parent
If the task was a fork, this field gives the task-id of the parent task.

Size
This indicates the amount of memory that the task is using. This is always reported in 4K amounts.

Res
This indicates the amount of time a task has been resident in core or has been swapped out to the disk. Each unit indicates 4 seconds. The maximum this will ever display will be 255. When a task changes status this number will go to zero.

The following are a few status call lines:

```
status
status +a
status +axl
```

The first example simply gives the short listing of the user's tasks in the system when status is run. The second example lists all of the jobs in the system, including shell, system and init tasks. The third example does the same thing as the second example but gives the long, detailed list for each task listed.

tail

The tail utility will print a given amount of characters from the end of a text file.

SYNTAX

```
tail file [n]
```

DESCRIPTION

This utility will print the last "n" characters of a text file. If "n" is not specified then the default of 250 characters will be used. If "n" characters from the end of the file happens to fall in the middle of a line, the line will be preceded by "..." to indicate that only a portion of the line is being printed. All other lines are printed as they appear in the file.

If "n" is greater than the number of characters in the file then the entire file will be printed.

Special characters, including carriage returns, should be included in the count "n" or else not as many characters from the end of the file will be printed as desired.

Here are some examples which demonstrate the tail utility:

```
tail test1
tail test2 150
```

The first example will print the last 250 characters from the file 'test1', provided that 'test1' contains more than 250 characters. The second example will print the last 150 characters from the file 'test2'.

SEE ALSO

head

tee

The tee command is used to read standard input and write to standard output and to the file(s) specified.

SYNTAX

```
tee [file ...]
```

DESCRIPTION

The tee command with no file(s) specified reads standard input and writes standard output. With file(s) specified, data read from standard input is duplicated to each of the file(s) in addition to standard output. A few examples will demonstrate the use of tee.

```
tee
tee test1
tee test1 test2
```

The first example will read standard input and then write the data to standard output. The second example will input from standard input and output to both standard output and test1. The last example will copy standard input into test1 and test2 as well as output the data to standard output.

time

The time utility prints the amount of time it takes to execute a UniFLEX command to standard output.

SYNTAX

```
time "UniFLEX command"
```

DESCRIPTION

This utility will determine the actual time spent processing the given UniFLEX command, the user CPU time and the system CPU time, and will then print those times to standard output after any output generated by the UniFLEX command has been printed.

A few example time calling lines follow:

```
time ls
time asmb asmb.test +sl
```

The first example will return the information about how long it took to execute the command "ls". The second example will return the same type of information (although the times reported will generally be longer) about how long it took to execute the command "asmb".

touch

The touch utility is used to set the last update time of a file to the current date and time.

SYNTAX

```
touch file ...
```

DESCRIPTION

This utility sets the last modification field of a file to the current time and date. For example, touch is useful if it is necessary to have an assembly language source file's modification date more current than its binary file so it can be reassembled using "update_all".

A few example calling lines of touch follow:

```
touch file1
touch file2 file3
```

The first example sets the last update of "file1" to the current time and date. The second example performs the same function, but for both files "file2" and "file3".

translate

The translate utility is used to map (translate) characters from standard input to standard output.

SYNTAX

```
translate map-file
```

DESCRIPTION

The mapping function of translate is specified by means of a map-file. Each mapping specification line of the map file specifies the domain (source) character and the range (target) string and has the following form:

```
domain-char delimiter range-char-string delimiter cr
```

where domain-char is the character from the input string which is to be replaced by range-char-string. Delimiter is any single character that does not appear in range-char-string. Each mapping specification must be terminated with a carriage return (cr). As many mapping specifications may be included in a map-file as are necessary.

The following is an example of how the translate utility might be used:

```
translate mapfile
```

where mapfile contains:

```
B/b/<cr>
0&zero&<cr>
<cr>.<cr><lf>.<cr>
```

If the input stream is:

```
Because we need an example, this is it.<cr>
But, only a 0 won't echo as the input character.<cr>
```

will result in the following being written to standard output:

```
because we need an example, this is it.<cr><lf>
but, only a zero won't echo as the input character.<cr><lf>
```

update_all

Process a set of files, performing the specified operation on each file if it is newer than the file it is compared to.

SYNTAX

```
update_all [<make_file_name>] [+q]
update_all <make_file_name> [<arg_list>] [+q]
```

DESCRIPTION

The "update_all" command reads the specified "makefile", which must conform to a special format, and conditionally performs the command or commands in that file. By default, "update_all" sends informative messages to standard output telling the user what it is doing. The command is most often used to recompile programs whose sources have been updated.

Arguments

<make_file_name>	The name of the file to read for instructions. This file must be in a special format (see Format of the "makefile"). If no other arguments are present, the default is the file "makefile" in the working directory. If other arguments are present, the user must specify the name of the "makefile".
<arg_list>	A list of strings to substitute for any string designators that appear in the "makefile" (see Format of the "makefile). If this argument is used, the user must specify the name of the "makefile".

Format of the "makefile"

The "makefile" is composed of modules, each of which is terminated with a percent sign, '%', in column 1. A module itself is composed of up to two parts. The first part specifies the process that "update_all" is to perform. The format for this first part is as follows:

```
[<item-one>::[$]<item_two>;]<command_sequence>
```

where <item_one> and <item_two> are the names of files; "::" is the "is newer than" operator; the dollar sign, '$', changes the interpretation of the "is newer than" operator if <item_one> exists but <item_two> does not; and the semicolon, ';', separates the names of the files from the

(continued)

command sequence.

The command sequence is composed of one or more UniFLEX commands. The "update_all" command replaces any sequence of more than one space character with a single space. Multiple commands are separated by additional semicolons. If the commands do not fit on one line, the user must begin and end the sequence with an exclamation point, '!', which serves to delimit the entire command sequence. If the first portion of the module uses more than one line, the second exclamation point marks the boundary between the first and second portions of the module. The command sequence is executed if <item_one> is newer than <item_two>.

The user may substitute an ampersand, '&', for any character or sequence of characters in <item_one>, <item_two>, or the command sequence. In such a case the "update_all" command substitutes for all ampersands the strings specified in the second portion of the module. If the second portion of the file is absent, no command sequence is performed. This portion consists of one or more lines, each of which contains a single string to substitute for the ampersands. The "update_all" command replaces each occurrence of an ampersand with the string on the first line of the second portion of the module and performs the command sequence if <item_one> is newer than <item_two>. It then replaces all ampersands with the string from the second line, continuing in this fashion until it reaches the end of the second portion of the module (marked by a percent sign in column 1).

The user may substitute a string designator (a pound sign, '#', followed by a digit from '1' through '9' inclusive) for any character or sequence of characters in <item_one>, <item_two>, or the command sequence. In such a case, the "update_all" command substitutes for each pound sign and its digit the corresponding element of <arg_list>. If the number represented by a digit is greater than the number of elements in <arg_list>, the sequence of the pound sign and digit remains intact.

If the file represented by <item_one> exists but the file represented by <item_two> does not, and if the "is newer than" operator is not followed by a dollar sign, "update_all" considers <item_one> newer than <item_two>. Under the same circumstances, if the "is newer than" operator is followed by a dollar sign, "update_all" does not consider <item_one> newer than <item_two>. In any case, if the file represented by <item_one> does not exist, or if neither the file represented by <item_one> nor <item_two> exists, <item_one> is not considered newer than <item_two>.

For instance, consider the following command:

```
update_all makefile s 1 y
```

and the accompanying "makefile":

(continued)

```
&::&.b;asmb & +#1#2#3
file_1
file_2
  .
  .
  .
file_n
%
```

This "update_all" command makes the following translation of the "makefile":

> If "file_1" is newer than "file_1.b", execute the command "asmb file_1 +sly".
>
> If "file_2" is newer than "file_2.b", execute the command "asmb file_2 +sly".
>
> It continues in this fashion until "file_n" is processed. The percent sign in column 1 marks the end of the module, and because it is the only module in the file, the "update_all" command terminates.

More than one set of commands can be processed with a single "makefile" if the user includes more than one module in the file.

Note that the use of the pound signs allows the same makefile to be used for another version of the "asmb" command. However, if the user does not specify three arguments on the command line, "update_all" cannot perform all the substitutions and the operating system cannot recognize the resulting form of the "asmb" command because it contains one or more string designators.

Options Available

> q Do not send informative messages to standard output.

NOTES

- The "chd" command has no effect in a "makefile".
- In order to remove the special meaning from either of the characters '#' or '&', the user must precede it with a backslash character, '\'. Similarly, to remove this special connotation from a backslash, the user must use two backslashes in a row.

(continued)

ERROR MESSAGES

*** Can't access Makefile "<file_name>" - aborted!
The operating system returned an error when "update_all" tried to open <file_name> for reading. Most probably, the file specification is incorrect, the file does not exist, or the user does not have read permission for the file.

*** Error: Command too complicated.
<command_sequence>
After substitution for the ampersands has taken place, the command sequence is too long (the limit is 1,024 characters).

*** Error: Pattern too complicated.
<command_sequence>
The pattern for the command sequence (before substitution for ampersands takes place) is too long (the limit is 1,024 characters).

Makefile syntax error - aborted
The "update_all" command was unable to interpret the "makefile".

Syntax: update_all [<make_file_name>] [+q]
update_all <make_file_name> [<arg_list>] [+q]
The "update_all" command requires exactly one argument. This message indicates that the argument count is wrong.

Unknown option: <char>
The option specified by <char> is not a valid option to the "update_all" command.

SEE ALSO

touch

The usage utility lists file usage on a disk by user.

SYNTAX

```
usage [+n] [device-name]
```

DESCRIPTION

This utility counts the number of files, directories and blocks used by each user on a disk, and then lists that information to standard output. Also, some summary information is generated and listed as well.

There is only one option to use with this utility:

+n sort the results by name rather than by usage. Normally the list is sorted by usage in blocks in ascending order.

If no device is given, the default device is "/dev/fd0". Some example calling lines of usage follow:

```
usage
usage /dev/fd1
usage +n /dev/fd0
```

The first example lists the file usage on "/dev/fd0" by user. The second example lists the file usage on "/dev/fd1" by user. The final example sorts the results and lists by name the files on "/dev/fd0". All three examples will print some summary information.

validate

The validate command is used to validate a backup made by "copy-dir".

SYNTAX

validate <source> ... <dest> [+options]

DESCRIPTION

This utility allows the user to ensure that a copy made by "copy-dir" was done properly. The program performs a file-wise comparison of the files from the source with the files on the backup. Any files which to do not match or are not present in the backup are reported.

The options available for this utility are a subset of the options available for "copy-dir". They are:

+d	Perform a depth-first directory search.
+D	Imply top-most directories.
+B	Ignore files which end in ".bak".
+l	List file names as they are being validated.
+t	Ignore directories at the top-most level which exist in the source directory but not in the destination directory.

If options are used making a backup with "copy-dir", the same options should be used with validate to ensure that the backup is good.

A few examples of validate follow:

```
validate bin /usr/bin +B
validate /usr /usr2 +d
validate /usr/all /usr2/usr/all +td
validate gen /usr2 +D
validate /usr/gen /usr/bin +l
```

The first example ensures that all files in 'bin' except those ending in ".bak" are also in '/usr/bin'. The second example validates that all files in '/usr' are also in '/usr2'. In the next example validate checks '/usr/all' against '/usr2/usr/all', ignoring any directories in '/usr/all' that do not exist in '/usr2/usr/all'. The fourth example validates 'gen' against '/usr2/gen'. In the final example, validate checks '/usr/gen' against '/usr/bin' and lists the name of each file as it is validated. This is the only example that will print anything unless a file is found that does not match or is not present in the backup.

SEE ALSO

copy-dir

verify

The verify command checks specified files or devices for read errors.

SYNTAX

verify <file or device ...> [+options]

DESCRIPTION

This utility reads every file or device specified and checks for read errors (and seek errors on devices). If an error occurs, a message will be printed telling at what block the read (or seek) error occured, and will continue reading. No message is printed if an error is not found.

Two options are available to change verify's procedure or output:

+l List the file or device names as they are being verified. If this option is specified, the number of bytes in the file or the number of blocks of the device will also be outputed.

+d If the file is a directory, a tree search of that directory will be performed, and all files in that directory and its subdirectories will also be read and verified. This option has no effect if a device is being verified.

If the '+d' option is used and a read error occurs while verifying a directory file, the files in the directory will not be verified.

This utility will stop reading if a seek or read error occurs while obtaining device information from a device. If any other file or device is specified, verify will resume verification of them.

Verify can only handle block, regular and directory type files. If a character file is encountered a message will be printed saying the file can not be handled and verify will resume verification of any other files or devices.

If while verifying a set of files, a file is encountered that is a block type file, for example, device "/dev/fd1", and there is a disk located in that device, then the device will be open and verified. This small side effect, while not a major problem, could cause a lot of unnecessary disk activity. One small caution should be exercised when using verify. If the user is verifying all of the files on a hard disk, do not also verify the directory '/dev'. If '/dev' is also verified, the check will be done twice when the hard disk is verified by reading the device '/dev/hd0'.

A few calling lines for verify follow:

```
verify file +l
verify dir +d
verify /dev/fd1 +l
```

The first example will read 'file' and if no read errors are found, will print 'file' and the number of bytes in it. In the second example, 'dir' will be read, and then all of the files contained in 'dir' will be read if there was no error in 'dir' itself. There will be no output unless a read error is found. The final example will read '/dev/fd1' and will print the number of blocks of the device if no error is found.

SEE ALSO

check, devcheck

words

The words utility determines the number of words and number of lines in a file and prints those values to standard output.

SYNTAX

```
words [file ...]
```

DESCRIPTION

This utility will determine the number of words and the number of lines from a file and will print those values to standard output. If more than one file is included then each set of values will be printed on two different lines following the order of input (left to right) of the file names.

A word is any series of characters (including a single character) except the blank (" ") which is used to separate words. A line is any series of words followed by a carriage return.

If no files are given then the utility reads from standard input until the standard end-of-file (control D) is encountered. A few example words calling lines follow:

```
words
words file1
words file1 file2
```

The first example will read standard input until an end-of-file is reached and will then print out the number of lines and number of words. The second example will print out the same information, but for "file1". In the the third example "file1" will have the information printed out, then following that will be the information from "file2".