UniFLEX™ Utilities Package II

technical systems consultants, inc.

Documentation Changes for Release 1.02 of UniFLEX® Utilities Package II
Technical Systems Consultants, Inc.
August 1, 1985

This package of documentation contains the changes necessary to bring the manual for your UniFLEX Utilities Package II up-to-date. Replace the existing documentation for the "nshell" command with the attached document. Add the documentation for the "addpath", "hangup", "nice", "prompt", "setpath", and "time" commands to your manual.

UniFLEX™ Utilities Package II

MANUAL REVISION HISTORY

Revision	Date	Change
A	3/84	Original Release

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.

Introduction

The UniFLEX™ Utility Package II contains close to thirty utilities for use under the UniFLEX Operating System. These utilities range in function from dynamically displaying resource activity of the operating system to executing commands at a specified date or time to an enhanced shell program. The manual contains a complete description of each command, and the disk contains "help" files for most commands (for all those not residing in the directory "/etc").

The user may want to take this manual apart and insert the pages into the "UniFLEX™ Utility Commands" section of the operating system manual. The utilities appear in alphabetic order. The documentation for each utility begins on a right-hand page.

The standard procedure for copying the utilities to the system disk is to use the "insert" command, which is supplied on the master disk. If a user does not want all the utilities on the system disk, the recommended procedure is to use the "insert" command, then to delete the files that are not needed.

Syntax Conventions

The following conventions are used in syntax statements throughout this manual.

1. Items that are not enclosed in angle brackets, '<' and '>', or square brackets, '[' and ']', are "keywords" and should be typed as shown.

2. Angle brackets, '<' and '>', enclose descriptions which the user must replace with a specific argument. Expressions enclosed only in angle brackets are essential parts of the command line. For example, in the command

   newuser <user_name>

   the name of a user must be specified in the place indicated by <user_name>.

3. Square brackets, '[' and ']', indicate optional items. These items may be omitted if their effect is not desired.

4. The underscore character, '_', is used to link separate words that describe one term, such as "user" and "name".

5. Characters other than spaces that are not enclosed in angle brackets or square brackets must appear in the command line as they appear in the syntax statement.

6. If the word "list" appears as part of a term in a syntax statement, that term consists of one or more of the elements described by the rest of the term, separated by spaces. For example, the term

   <user_name_list>

   represents a list of user names.

7. Some utilities support optional features, known as options, which alter the effect of the command. Options consist of either a single character or a single character, followed by an equals sign, '=', followed by an argument. An "option string" is a plus sign followed by one or more options. An option string may contain any number of single-character options but only one option which takes an argument. An option requiring an argument must be the last option in an option string. Thus, the command line must contain a separate option string for each option requiring an argument. It may or may not contain a separate option string for each single-character option.

(continued)

The following are valid option strings:

```
+abcdefg
+abc=<arg>
```

The following are not valid option strings:

```
+abc=<arg>de
+a=<arg>b=<arg>
```

Unless specifically stated in the documentation about a particular command, option strings may appear anywhere on the command line after the command name.

Many common terms appear (often as abbreviations) in more than one syntax statement. The manual does not explain these terms each time they appear. However, the following table describes each one.

Table 1. Common Terms Used in Syntax Statements.

Term	Meaning
char	Character
dev	Device
dir	Directory
str	String
file_name	A valid file name

Command Summaries for Utilities Package II

at	Queue commands and jobs for execution at a specified time.
atexecute	Activate the queue handler used by the "at" command.
atinfo	List information about commands queued for later execution.
backup	An archival backup and restore program.
basename	Extract the base file-name from a path name.
basicerror	Translate BASIC error numbers into English messages.
boottime	Report the date and time of the last system boot.
calendar	Display a calendar for any specified month or year.
crt_termcap	Create a file of terminal capabilities.
dircompare	Compare the entry names in two directories.
dirname	Extract the directory name from a path name.
diskinfo	Display detailed information about disk format, size, and contents.
dsd	Dynamically display resource activity of the active operating system.
filedevice	Report the name of the device on which a specified file resides.
flex-rel	Convert a UniFLEX relocatable file to FLEX format.
keep	Retain selected files in a directory.
libinfo	Display entry point and module information contained in a library.
more	List text files with full user control.
newuser	Temporarily login as a new user.
nshell	An enhanced shell program.
objcmp	Compare two object files and report differences.
pack	Compress text files.
pwfcheck	Validate the contents of the password file.
relinfo	Display relocation, external, and symbol information of an object file.
strings	Extract strings from an object or binary file.
swapstats	Report statistics for system swap device and memory usage.
trail	List a file as it grows.
unique	List text files, omitting sequentially duplicated lines.
unpack	Restore text files compressed by "pack".

```
at <when> [+bdmrwx]
/etc/atexecute [+krs]
atinfo [+hp]
backup <dev_name> [<file_name_list>] [+abdlnptABCLRT]
basename <path_name> [<suffix>]
basicerror <number_list>
boottime
calendar [<month>] [<year>]
/etc/crt_termcap <ttycap_file> <ttyassoc_file> <termcap_file>
dircompare <dir_1> <dir_2>
dirname <path_name>
diskinfo <device_name> [<device_name_list>]
dsd [<sleep_time>]
filedevice <file_name> [<file_name_list>]
flex-rel <UniFLEX_file_name> <FLEX_file_name>
keep <file_name> [<file_name_list>] [+pq]
libinfo <library_name> [<library_name_list>] [+emM]
more [<file_name_list>]
newuser [<user_name>]
nshell [<+abclnvx>] [<argument_list>]
objcmp <file_name_1> [<file_name_2>] [+cq]
pack [<infile_name>] <outfile_name>
pwfcheck [<file_name>] [+nw]
relinfo <file_name> [<file_name_list>] [+ehrs]
strings [<file_name_list>]
swapstats
trail <file_name> [+flns]
unique [<file_name_list>] [+d]
unpack <infile_name> [<outfile_name>]
```

addpath

Add a name to the list of directories the nshell program searches when looking for an executable file.

SYNTAX

addpath <dir_name_list>

DESCRIPTION

The "addpath" command, which is part of the nshell program, adds a name to the end of the list of directories the nshell program searches when looking for an executable file. This list, which is known as the search path, is searched sequentially. By default, the search path consists of the following directories: the user's working directory, "<home_dir>/bin", "/bin", and "/usr/bin". (The home directory is the user's login directory, as specified in the password file.) If the user is the system manager, the search path also includes the file "/etc", which is searched after "<home_dir>/bin" and before "/bin".

Arguments

<dir_name_list>	A list of the names to add to the search path. The names are added to the end of the list in the order the user specifies them on the command line.

EXAMPLES

1. addpath /usr/games
2. addpath .. bin

The first example adds the name "/usr/games" to the end of the list of directories to search.

The second example adds the parent directory of the user's working directory and the directory "bin" in the user's working directory to the end of the list of directories to search.

NOTES

- The "addpath" command is only effective while the nshell program under which it is invoked is running. The list of directories searched by the login nshell can be permanently altered by placing the appropriate command in the file ".startup" in the user's home directory. The nshell program automatically executes this file each

(continued)

time the user logs in.

SEE ALSO

setpath
nshell

at

Submit commands for execution at a later date and time.

SYNTAX

at <when> [+bdmrwx]

DESCRIPTION

The "at" command submits commands read from standard input for execution at a later date and time, which are specified by the <when> argument. If standard input is a terminal, the prompt "at>> " is issued to standard error, requesting another line. To stop entering lines, the user must type the end-of-file character (control-D) as the first character of a line.

Arguments

<when> The date and time to execute the commands.

Format for Arguments

<when> [<time>]
[<time>] <date>
[<time>] <day>
now

The <time> parameter may be of the form <hh>:[<mm>] where <hh> is the hour number (0 through 23 inclusive), and <mm> is the minute number (0 through 59 inclusive, 0 by default). A twenty-four hour clock is assumed unless the user appends "am" or "pm" (or "AM" or "PM") to the <time> parameter (in which case the hour number <hh> must be between 1 and 12 inclusive). The <time> parameter may also be a keyword which describes the time of day (see Keywords). If the <time> parameter is omitted, 00:00 (midnight) is assumed.

The <date> parameter may be of the form [<mm>/]<dd> or <dd>[.<mm>] or [<mm>-]<dd> where <mm> is a number representing the month of the year (1 through 12 inclusive) and <dd> is a number representing the day of the month (1 through 31 inclusive, see NOTES). If the month <mm> is omitted, it defaults to the next month if both the day of the month <dd> and time <time> have passed in the current month or to the current month if they have not. The <date> parameter may also be of the form <month> <dd> or <dd> <month> where <month> is a keyword describing a month of the year (see Keywords), and <dd> is a number between 1 and 31 inclusive.

(continued)

The <day> parameter is a keyword describing a day of the week (see Keywords). The keyword "now" requests execution as soon as possible.

Keywords

A keyword is recognized by any sequence of adjacent characters, beginning with the first character, that is unique to that keyword. For example, "su", "sun", "sund", "sunda", and "sunday" are all recognized as the keyword "sunday". However, "s" is not, since two other keywords, "september" and "saturday", also start with that sequence.

Keywords known to "at" are

Months	Weekdays	Time of Day	Miscellaneous
january	sunday	midnight	now
february	monday	noon	
march	tuesday		
april	wednesday		
may	thursday		
june	friday		
july	saturday		
august			
september			
october			
november			
december			

Options Available

b	When the specified time arrives and execution of the list of commands begins, do not wait for its completion before starting to execute commands queued by other "at" calls.
d	If the list of commands has expired (i.e., the "atexecute" command discovers that over an hour has passed since the requested execution time), tell "atexecute" to execute the commands at the first opportunity instead of deleting them.
m	Mail messages to the user telling at what times execution of the list of commands began and ended.
r	Resubmit the commands on successful completion with the same <time> parameter.
w	Execute on working days only.
x=<cmd>	Execute the command supplied as an argument (<cmd>) instead of obtaining a list of commands from standard input (see NOTES).

(continued)

EXAMPLES

1. at 5:00pm wednesday <wed_cmnds
2. at
3. at 8:am "+rwmx=shell startday"

The first example submits commands for execution at 5:00 P.M. on a Wednesday. The commands are read from the file "wed_cmnds".

The second example submits commands for execution at midnight. These commands are read from standard input. If standard input is a terminal, the prompt "at>> " requests the next command in the list being submitted. The user must type an end-of-file character as the first character on the line in order to end the list.

The third example submits the command "shell startday" for execution at 8:00 A.M. on working days only and requests that the command be resubmitted upon successful completion (the quotation marks are necessary because of the space character embedded in the command). This example sends timestamps to the user through the system mail when the command begins and when the command successfully ends.

NOTES

- The time parameter <when> is the next occurrence of that time. For example, the command "at 14:00 1" should be read as "at the next 14:00 hours on the first of a month" instead of as "on the next first of the month at 14:00 hours." Notice the difference in meaning if it is currently noon on the first of the month.

- If the user types a keyboard interrupt (control-C), the commands being submitted by "at" are discarded, and control returns to the calling procedure (usually the shell program).

- If the argument to the 'x' option contains a space or any other character which has special meaning to the shell program (such as the matching characters, the pipe symbol, or the symbols for I/O redirection), the user must enclose the option string which contains the 'x' option in quotation marks (see the third example). command.

- If the user represents a day of the month with a number that is greater than the largest day of that month, "at" interprets it as the last day of the month. For example, "2/31" always refers to the last day of February.

- If a list of submitted commands expires (the requested execution time passes by more than an hour and the 'd' option has not been requested) and the 'r' option has been requested, the list of commands is not executed, but is resubmitted with the same <time> parameter as though it had been executed.

(continued)

ERROR MESSAGES

at error: Unknown option: <char>
The option <char> is not known to "at" (see SYNTAX and Options Available).

at error: Unrecognizable string: <str>
The characters <str> could not be deciphered by "at" (see Format of Arguments).

at error: Ambiguous string: <str>
The characters <str> are not unique to one keyword (see Keywords).

at error: Invalid construction of date and time.
The date and time specified contain conflicting information (see Format of Arguments).

at error: Month number is out of range.
The month number specified is less than 1 or greater than 12 (see Format of Arguments).

at error: Day of month is out of range.
The day of the month specified is less than 1 or greater than 31 (see Format of Arguments).

at error: Hour number is out of range.
The hour number specified is greater than 23 (see Format of Arguments).

at error: Minute number is out of range.
The minute number specified is greater than 59 (see Format of Arguments).

at error: "AM" or "PM" with twelve hour clock only.
Either "AM" or "PM" was used with an hour number which was not between 1 and 12 inclusive (see Format of Arguments).

at warning: 'r' option ignored with 'now' keyword.
The repeat option 'r' is ignored if the <time> parameter is "now". The commands are not resubmitted upon successful completion.

at warning: 'w' option ignored with 'now' keyword.
The option that requests execution on working days only, 'w', is ignored if the time specification is "now".

at warning: restart /etc/atexecute
The commands have been queued, but they cannot be successfully executed unless the "atexecute" command is run before the execution time arrives.

(continued)

SEE ALSO

atexecute
atinfo
nshell
shell

atexecute

Prepare the "at" subsystem for execution and initiate the "at daemon" that handles the lists of commands submitted by the "at" command.

SYNTAX

```
/etc/atexecute [+krs]
```

DESCRIPTION

The "atexecute" command initiates the handling of lists of commands that have been submitted through the "at" command. It first prepares the directory containing the lists submitted ("/usr/spooler/at"), then spawns the daemon, or continuous background task, that executes these lists.

The command fails if the system is in single-user mode. All expired lists of commands (those whose execution time has passed by more than one hour) that were submitted through "at" without the 'd' option are removed.

Options Available

- k Keep all lists of commands, including those which have expired.
- r Remove all lists of commands, regardless of their execution times.
- s Permit execution in single-user mode.

The "at daemon"

The "at daemon" (from now on referred to as the daemon) handles the execution of lists of commands submitted through the "at" command. It normally sleeps until an event occurs to indicate that there is something for it to do. These events are "alarm" interrupts, "hangup" interrupts, and interrupts from the "at" command.

The "alarm" and "at" interrupts indicate to "atexecute" that it must search the list of files containing lists of commands (called submitted files) to see if one is ready for execution. Any submitted file whose requested execution time has passed is ready for execution. If no file is ready for execution, the daemon computes the time until something will be ready to execute and sleeps for that amount of time. If at least one file is ready for execution, the daemon selects a file for execution.

(continued)

If more than one file is ready to run, the daemon selects the files in order of ascending execution time. If two or more files have the same requested execution time, the files submitted with the 'b' option are selected before those submitted without it. Otherwise, the order of selection is indeterminable.

Submitted files have names of the form "##########A?" where each pound sign, '#', is a digit (0-9), 'A' is the letter 'A', and '?' is any upper- or lowercase letter. Encoded in that file name are the requested execution time and the execution flags. (The "atinfo" command decodes that information.)

After a file has been selected for execution, the first number in its name is changed to its corresponding letter of the alphabet (e.g., 0 becomes 'A', 1 becomes 'B', etc.). A shell program is invoked to process the selected command list. The daemon waits for the shell program to complete before selecting another file unless the file was submitted with the 'b' option. When the shell ends, "atexecute" removes the file and searches to see if any remaining files are ready for execution.

If the daemon receives a "hangup" interrupt, it terminates gracefully as soon as possible. It terminates immediately if it is sleeping, but if it is currently executing a file, it waits until that file completes before terminating. In all cases, the daemon immediately breaks the communication link between "at" and itself.

Format of the "holidays" File

The "at" daemon handles the "at" option 'w' (execute on working days only). The daemon looks for a file called "/usr/spooler/at/holidays". If it does not find the file, the daemon assumes that all days of the week are working days and that there are no holidays. If it finds the file "holidays", it expects it to contain a list of days of the week which are not working days and a list of dates which are holidays.

The first line of the file "holidays" is a list of keywords, separated by spaces, naming the days of the week which are not working days. A maximum of five nonworking weekdays is accepted. (See "at" for information on keywords.)

On the second and subsequent lines, the dates of the year that are holidays (nonworking days) are listed, one per line. The format for the date is the same as that for the <date> parameter described for the "at" command. A maximum of thirty holiday dates is accepted.

(continued)

NOTES

- The ´k´ and ´r´ options are mutually exclusive and may not be specified together.

- If a file submitted through "at" with the ´r´ option expires, it is resubmitted before it is removed.

- Communications between "at" and the daemon are made through a file called "/tmp/atxctrpid", which contains the task number of the daemon and other information. If the daemon discovers that this file has been deleted or corrupted, it terminates.

- The system manager can avoid having to execute the "atexecute" command every time the system is booted by putting the following command in the file "/etc/startup":

```
/etc/atexecute +ks
```

ERROR MESSAGES

atexecute error: ´r´ incompatible with ´k´
The ´r´ option (remove all submitted files) is incompatible with the ´k´ option (keep all expired files).

atexecute error: ´k´ incompatible with ´r´
The ´k´ option (keep all expired files) is incompatible with the ´r´ option (remove all submitted files).

atexecute error: System is in single-user mode.
The "atexecute" command does not allow itself to run if the system is in single-user mode unless the user specifies the ´s´ option.

atexecute error: /tmp/atxctrpid already exists
The file that allows communication between the "at" command and the "at" daemon already exists. Probably, the "atexecute" command is not necessary as a daemon is already running on the system. However, if "at" commands are warning that they are unable to awaken the daemon, the system manager should remove this file and try the "atexecute" command again.

atexecute warning: no holidays
The file "/usr/spooler/at/holidays" could not be found or could not be deciphered. No holidays (nonworking days) are recognized.

SEE ALSO

at
atinfo

atinfo

Display the current status of the "at" subsystem.

SYNTAX

```
atinfo [+hp]
```

DESCRIPTION

The "atinfo" command examines the "at" subsystem and writes its current status, consisting of the status of the "at" daemon and a table of submitted lists of commands, to standard output. If the user specifies the ´h´ option, the file containing the information on nonworking days and holidays is also displayed.

The ´p´ option allows a user to examine and remove files that the same user previously submitted. The system manager may examine or remove any file. In response to the prompt "? ", the user may type an ´l´ to examine (list) the list of commands, an ´r´ to remove the list of commands, or an ´n´ to go to the next list of commands. A carriage return must follow the character. Typing only a carriage return is the same as typing an ´n´ followed by a carriage return. The ´p´ option causes "atinfo" to write all information to standard error. Otherwise, it writes all information to standard output.

Options Available

h	List the file containing information about holidays.
p	Prompt to list and remove each list of commands submitted.

ERROR MESSAGES

syntax: atinfo +hp
The "atinfo" command contains either an argument which is not an option string or an unknown option.

´<char>´ unknown. Use ´l´, ´n´, or ´r´
The "at" command cannot recognize the character typed in response to the question-mark prompt. The recognizable responses are ´l´, ´n´, and ´r´.

atinfo warning: Cannot be examined
The file has either been selected for execution or removed. Therefore, it may not be examined.

(continued)

atinfo warning: Cannot be removed
The file has either been selected for execution or removed. Therefore, it may not be removed.

MESSAGES

"at" daemon is available, process id is <number>
The "at" subsystem is currently active and is available for the execution of lists of commands. The process ID of the "at" daemon is <number>.

"at" daemon is not available
The "at" subsystem is not currently active. Lists of commands submitted by the "at" command cannot be executed until the subsystem is activated by the "atexecute" command.

SEE ALSO

at
atexecute

backup

Backup or restore files.

SYNTAX

backup <dev_name> [<file_name_list>] [+abdlnptABCLR]

DESCRIPTION

The "backup" command is used to create and maintain archival backups of files or directories on the system. Although the program is named "backup", it can operate in four distinct modes, selected by options: create mode, append mode, catalog mode, and restore mode. In create mode "backup" copies the specified files or directories to the backup device. It destroys any data that are already on the backup device. In append mode, "backup" adds the specified files or directories to the backup device beyond all existing files. Thus, it is possible to append to a backup device a file with path and file names identical to those of an existing backup file. In catalog mode "backup" lists the contents of the backup device in much the same format as that used by the "dir" and "ls" commands. In restore mode it retrieves files or directories from a backup device.

The "backup" command stores files and directories on, and retrieves them from, block devices only. In most cases the backup device is some sort of disk, probably a floppy diskette. The "backup" command uses a unique file structure on backup devices, which is completely different from the standard UniFLEX file structure. Therefore, backup devices must not be mounted onto the UniFLEX file system using the "mount" command. The only way to read devices written by "backup" is to use "backup" in restore mode. The only other UniFLEX command which the user should use on a backup device is "devcheck".

If the backup device is a disk, it should generally be formatted before the backup operation begins. Although the UniFLEX file structure created by the format command is destroyed by "backup", the raw track-formatting is essential. During the backup process, the user is given the opportunity to request that "backup" format disks before writing to them.

Backups may extend over more than one volume of the backup medium. There are no restrictions on the sizes of files copied. If necessary, "backup" breaks files into segments and stores each segment on a different volume.

(continued)

Arguments

<dev_name>	Name of the backup device.
<file_name_list>	List of the names of files and directories to process. Default is the working directory.

If the user specifies a directory name as an argument in restore, create, or append mode, the program processes only the files within that directory. If the user also specifies the 'd' option, the program restores all files within the given directory and its subdirectories.

Options Available

a=<days>	Copy only those files which are no older than the specified number of days. A value of 0 specifies files created since midnight on the current day; a value of 1 specifies files created since midnight of the previous day, and so forth.
b	Print sizes of files in bytes.
d	Backup or restore entire directory structures.
l	List file names as they are copied or restored.
n	Only restore a file if the copy on the backup device is newer than the copy at the destination. If the destination file does not exist, the program restores the file (unless prohibited by another option, such as the 'B' option).
p	Prompt the user with each file name to determine whether or not the specified procedure (backup or restore) should be performed on that particular file.
t[=<file_name>]	Backup only those files which have been created or modified since the date in the specified file. When the backup is finished, update the date in the file (see NOTES). If the user does not specify a file, the default is ".backup.time".
A	Append to a previous backup.
B	Do not backup or restore files which end in ".bak".
C	Print a catalog of the files on an existing backup. If the user specifies the 'C' option, "backup" ignores all the names in <file_name_list>.
L	Do not unlink files before restoring.
R	Restore files from an archive.

All modes except catalog mode are quiet. The 'l' option allows the user to see what the program is actually doing.

The 'n' option is only used in restore mode.

The 't' option can be used only in create and append modes. If the user specifies the 't' option, but the "backup time" file specified as its argument does not yet exist, "backup" copies all the files and

(continued)

directories listed on the command line. Thus, a user may obtain a full backup (either without the 't' option or with a nonexistent "backup time" file) or a partial backup, which includes only those files created since the last backup.

EXAMPLES

1. backup /dev/fd0 +1
2. backup /dev/fd0 +C
3. backup /dev/fd0 +1R
4. backup /dev/fd0 +1d file1 file2 dir1 dir2
5. backup /dev/fd0 +1Rn file1 dir2
6. backup /dev/at0 +1t
7. backup /dev/fd0 +1At=backup_time
8. backup /dev/fd0 +1d file1 file2 dir1 dir2 +a=5

The first example backs up all files in the working directory to the device "/dev/fd0". The file names are listed as they are copied to the device.

The second example lists the contents of the backup on "/dev/fd0". If this command is executed just after the command in the first example, a detailed listing of the files copied is printed. The format of this listing is very similar to that of the commands "dir" and "ls".

The third example restores all of the files, excluding subdirectories and their contents, from the backup on "/dev/fd0". If this command is executed just after the command in the first example, the files backed up in that example are restored.

The fourth example copies (in order) the files "file1" and "file2", then all files and directories contained in the directories "dir1" and "dir2".

The fifth example restores the file "file1" from the backup. It then restores the files contained in "dir2" on the backup, creating the directory "dir2" if necessary. This example does not restore any subdirectories in "dir2" or any files or directories contained in subdirectories in "dir2".

The sixth example creates the same backup as the first example, but only copies the files created or modified after the time contained in the file ".backup.time". If this file does not exist, all the files are copied.

The seventh example adds a set of files to a previously created backup. In particular, it adds exactly the files which were created or modified since the creation of the file "backup_time".

(continued)

The eighth example performs the same function as the fourth example except that it copies only those files which are five days old or less.

NOTES

- When using append mode, the user must place the final volume of the backup medium in the backup device. The program then appends files to that volume, requesting new volumes as necessary.

- In restore mode, file names or directory names on the command line are used to select the files or directories to be restored. The program searches the entire backup for each argument specified. If multiple files satisfy the restoration criteria, the program restores them all, destroying the older version as the new one is restored. Thus, the user must provide all backup volumes (in order) for each argument to ensure proper restoration.

- When files are restored, they are generally restored to the same directory location as the user specified when they were backed up. As files are backed up, "backup" makes an indication of the path name for each file. When files are restored, the program uses the path name to place the file in its proper directory location. If the path name is relative (i.e., does not begin with '/'), the path name of the restored directory is also relative. Thus, files backed up with a relative path name may be restored to a directory location different from the one in which they were created. An example should make this clear. If the working directory is backed up, either by specifying no source files or by using the directory name '.', the files are backed up with a relative path of '.'. When these files are restored, they are placed in the directory '.', which might not be the same directory they originally came from. This feature allows the manipulation of entire file systems in a general fashion. To specify a unique directory location for a file, the user should specify its entire path name, starting with '/'.

- It is possible to restore backed up data onto the device currently being used as the root device or system disk. Two possible problems arise, however. First of all, if the UniFLEX operating system is restored from a backup, the result is not bootable. In such a case, the UniFLEX file must be copied from the original master diskette and installed in order to allow booting. The second problem occurs if the shell program or the device "tty00" is restored over the current shell or "tty00". This operation leaves unreferenced files in the file system. Unreferenced files must be corrected with the "diskrepair" command. In general, it is always a good idea to run "diskrepair" on the root device after restoring backed-up data to it.

(continued)

MESSAGES

Several of the following messages prompt the user for a positive or negative response. The program interprets any response that does not begin with an upper- or lowercase 'n' as a positive response.

Restore backup from "<file_name>"
Catalog of backup on "<file_name>"
Update backup on "<file_name>"
Backup to "<file_name>"
These messages are printed when "backup" begins. They notify the user of the function about to be performed.

Volume name?
Each set of backup volumes has a name. The user should enter the name in response to this prompt. The name may contain as many as forty characters.

Insert next volume - Hit C/R to continue:
This prompt is issued when the program needs a new backup volume. The user should type a carriage return only when the next volume has been placed in the device. When creating new backups or when appending to an old one, the user may enter the character 'f', followed by a carriage return. If the program is in append mode, it automatically switches to create mode and starts a new backup. The 'f' indicates that the volume has been inserted in the drive, but that it must be formatted before continuing. In this case the program prompts the user for the specific information necessary to format the volume. Subsequent format operations use the same information; thus, all volumes which were not previously formatted must have the same characteristics (e.g., single-sided, double-density).

Format program name?
This prompt is issued in response to a "format" request for the next volume. The user should respond with the name of the appropriate formatting program for the given device.

Device model name?
The user should respond to this prompt with the model name which corresponds to the volume being formatted. Refer to the documentation for the format program for the available models.

Do you wish to abort "append" function and create a new backup?
This message is printed at the initiation of the "append" operating mode if an invalid header (indicating a bad backup format) is detected. The user has the option of aborting from "append" mode and switching to "create" mode.

(continued)

Volume <number> of "<vol_name>"
Whenever a new volume is inserted and properly validated, the program prints this message, which indicates the name of the backup volume and its sequence number.

This is Volume #<number_1> -- Expected Volume #<number_2> --
Continue?
The program expects the user to insert volumes in sequential order. If a volume appears out of order, "backup" prints this message. If the user types anything except an 'n' or an 'N' as the first character of the response to the message, "backup" ignores the fact that the volumes are out of order and continues with the backup. Otherwise, it prompts the user for another volume. If the program is in restore mode, it is important to insert volumes sequentially because "backup" cannot correctly restore files that are broken across volumes if the volumes are inserted out of order.

Copy "<file_name>" (y/n)?
Restore "<file_name>" (y/n)?
If the user specifies the 'p' option, the program prints one of these prompts before it takes any action. A response of 'n' or 'N' indicates that the operation should not be performed for the given file. Any other response is interpreted as "yes".

link "<file_name_1>" to "<file_name_2>"
copy "<file_name>"
==== Copying from "<dir_name>"
The program prints these messages as it takes the corresponding action during a creation operation.

ERROR MESSAGES

-- Formatting not allowed during Catalog/Restore
The user may not format a disk if the program is in either catalog or restore mode.

*** Invalid Volume Header -- Not a "backup" disk ***
The program validates each backup volume before using it. If this validation fails, the program prints this message to indicate that something is wrong. The user then has another chance to insert the proper volume and continue. If validation fails while the program is in append mode, the user may abort from append mode and create a totally new backup instead.

(continued)

** Warning: directory "<dir_name>" is too large!
** Some files were ignored
** Warning: directory "<dir_name>" is too large!
** Some directories were ignored

The program uses some internal tables during the backup process (not during restore or catalog). If the limits of these tables are exceeded (highly unlikely), these messages are printed.

Read error! - file "<file_name>"
Write error! - file "<file_name>"

An I/O error occurred during the transfer of a file either to or from the backup. An auxiliary message is printed indicating the nature of the error. The program tries to continue for all errors except "device full" during restore mode.

"<file_name>" not located - try again?

When using the program in restore mode, the user may specify which files or directories to restore. If the program cannot find a specified file or directory after searching the entire backup, it prints this message. If the response is not 'n' or 'N', the program searches the entire archive again. This option is allowed because volumes need not be inserted in order of their creation when the program is in restore mode. If one volume is left out or if the final volume is inserted before the entire archive has been processed, some files might not be processed. Note that if the user specifies more than one file name or directory name, the program processes the entire archive for each file before proceeding to the next one.

Unknown option: <char>

The character <char> is not a valid option for the program.

"backup" must run with system manager privileges!
Currently running as: <user_number>

Some features of "backup" require privileges only available to the system manager. In most installations the program will be installed so that these privileges are given to the program. If this message is printed, the user should check with the system manager.

('t' or 'a') and ('C' or 'R') are incompatible options
'A' and 'C' are incompatible options
'A' and 'R' are incompatible options
'R' and 'C' are incompatible options
'a' and 't' are incompatible options

The program can run in only one of its four operating modes at a time. Specifying certain combinations of options implies the execution of more than one operating mode and is therefore illegal.

(continued)

"<dev_name>" is not a block device

The destination device for the backup must be a block device. This message indicates that the specified device (which is always the first argument) is not such a device.

SEE ALSO

format

basename

Extract the base file-name from a path name.

SYNTAX

```
basename <path_name> [<suffix>]
```

DESCRIPTION

The "basename" command extracts the simple file-name from the path name <path>, removes the suffix from the simple file-name (if a suffix was specified), and produces a base file-name. It then writes this base file-name, followed by a carriage return, to standard output.

Arguments

<path_name>	The path name from which to derive a base file-name.
<suffix>	A suffix to remove, if present.

EXAMPLES

1. basename hello.c .c
2. basename /usr/joe/docs.txt .txt
3. basename /usr/jan/sorted_data
4. basename /usr/kim/program.p .c

The first example writes the string "hello", followed by a carriage return, to standard output. The "basename" command removes the suffix ".c" from the path name "hello.c", but it does not remove a directory name since there is none to remove.

The second example writes the string "docs", followed by a carriage return, to standard output. The "basename" command removes the directory name "/usr/joe/" and the suffix ".txt" from the path name "/usr/joe/docs.txt".

The third example writes the string "sorted_data", followed by a carriage return, to standard output. The command removes the directory name "/usr/jan/" from the given path name, but it does not remove a suffix since none is specified.

The fourth example writes the string "program.p", followed by a carriage return, to standard output. The command removes the directory name "/usr/kim/" from the path name, but does not remove the suffix ".c" since the path name does not end with that suffix.

(continued)

NOTES

. If the argument <path_name> ends with a slash character, '/', or is a null or blank string, the "basename" command writes a carriage return to standard output.

ERROR MESSAGES

usage: basename <path_name> [<suffix>]
The "basename" command requires at least one and no more than two arguments. This message indicates that the argument count is wrong.

SEE ALSO

dirname

basicerror

Give an English interpretation of a BASIC error number.

SYNTAX

basicerror <number_list>

DESCRIPTION

The "basicerror" command gives an English interpretation of each of the BASIC error numbers specified as arguments.

Arguments

<number_list> A list of one or more BASIC error numbers.

EXAMPLES

basicerror 30 72 208

In the example, "basicerror" prints an English interpretation of BASIC errors 30, 72, and 208.

NOTES

- The English interpretations of the BASIC error numbers less than 201 are stored in the file "/gen/errors/basic". This file is in a special format which the user should not try to modify.
- The interpretations of error numbers greater than 200 are stored in the file "/gen/errors/system", which may not exist on earlier versions of the operating system. If "basicerror" cannot find the file, it prints the corresponding UniFLEX error number. The file is in a special format which the user should not try to modify.

ERROR MESSAGES

Error reading "/gen/errors/basic".
The operating system returned an error when "basicerror" tried to read the file "/gen/errors/basic".

File "/gen/errors/basic" cannot be opened.
The operating system returned an error when "basicerror" tried to open the file "/gen/errors/basic". This message is preceded by an interpretation of the error returned by the operating system.

(continued)

File "/gen/errors/basic" cannot be located.
The file "/gen/errors/basic" does not exist.

<argument> is not a valid BASIC error number.
The specified argument either is not a number or is outside the range of valid BASIC error numbers.

SEE ALSO

basic

boottime

Display the date and time of the last system boot.

SYNTAX

```
boottime
```

DESCRIPTION

The "boottime" command reports the date and time of the last system boot. It accepts no arguments or options. Boot time is defined as the time at which the system manager entered the date and time.

EXAMPLES

```
boottime
```

This example is the only valid form of the "boottime" command. It displays the date and time of the last system boot.

ERROR MESSAGES

Syntax error - no arguments expected.
The user specified an argument or option to the "boottime" command, which accepts neither.

calendar

Display a calendar for the specified year or month.

SYNTAX

calendar [<month>] [<year>]

DESCRIPTION

The "calendar" command prints a calendar for any year or month.

Arguments

<month> The month to display.
<year> The year to display. Default is the current year.

Format for Arguments

<month> <name_or_number>
<year> <number>

The <month> argument may be either a number between 1 and 12 inclusive (January is 1) or a character string representing the name of the month. The name of a month is recognized by any sequence of adjacent characters, beginning with the first character, that is unique to that month. For example, "ja", "jan", "janu", "janua", "januar", and "january" are all valid representations of the first month of the year. However, 'j' is not a valid representation because two other months, June and July, also start with that sequence.

The <year> argument may be any number from 1 to 9999 inclusive. Numbers under 100 are considered to be the corresponding year within the twentieth century unless they contain a leading 0.

If the user supplies only one argument, and it is a number, "calendar" must decide whether the argument specifies a month or a year. If the number has a leading 0, it is always considered a year. If the number is between 1 and 12 inclusive (without a leading 0), it is considered a month. Otherwise, it is considered a year.

EXAMPLES

1. calendar
2. calendar sept 85
3. calendar 7 1776
4. calendar 085

(continued)

The first example prints a calendar for the current year.

The second examples prints a calendar for the month of September, 1985.

The third examples prints a calendar for the month of July, 1776.

The fourth examples prints a calendar for the year 0085 (first century).

NOTES

. By September, 1752, when the Gregorian calendar was adopted, the vernal equinox had been displaced by eleven days. To correct for this displacement the new calendar skipped eleven days. Therefore, the calendar for that month contains nineteen, rather than thirty, days.

ERROR MESSAGES

Invalid date.
The user specified either an invalid name for a month or a number for the month or year outside the ranges accepted by "calendar".

crt_termcap

Create a file defining the capabilities of each terminal on the system.

SYNTAX

/etc/termcap <ttycap_file> <ttyassoc_file> <termcap_file>

DESCRIPTION

The "crt_termcap" command creates a file ("termcap") which describes the capabilities of each terminal on the system. This file makes it possible for programs to operate on many different terminals regardless of the individual characteristics of the terminals. The "crt_termcap" command must be used to create the file "termcap" before any programs which need that file can operate successfully.

Arguments

<ttycap_file>	File describing functional capabilities of each terminal.
<ttyassoc_file>	File associating each active port with a particular kind of terminal.
<termcap_file>	Output file combining information from the two input files.

The "crt_termcap" command uses two input files to produce one output file. The first input file describes the functional capabilities of each terminal on the system. The second file indicates what type of terminal is on each active port in the system. These files must each conform to a particular format. The following two sections describe these formats.

Format of the File "ttycap"

The file "ttycap" contains one logical entry for each terminal on the system. The format of an entry is

<terminal_name> : <capability_list> :

where <terminal_name> is a character string (it may contain as many as ten characters) which identifies the terminal, and <capability_list> is a list describing the capabilities of the terminal. Each item in this list has the following format:

<keyword> = <value_list>

where <keyword> is a two-character sequence representing a function such

(continued)

as clearing the screen and <value_list> is a list containing decimal values, hexadecimal values, or both. Each value in the list must be separated from the following one by a plus sign, '+'. No spaces may appear between the values and the plus signs. All hexadecimal values must consist of two digits preceded by a dollar sign, '$'. The following are valid value strings:

```
1+2+3
$01+$ff+$80
```

The keywords currently supported are

ho	Home cursor.
cu	Move cursor up without modifying display.
cd	Move cursor down.
cl	Move cursor left.
cr	Move cursor right.
cs	Clear entire screen.
nr	Number of rows on screen (first row is 1).
nc	Number of columns on screen (first column is 1).
wt	Number of seconds to wait between clearing the screen function and sending more information to the terminal.
is	A string of characters sent to the terminal when processing begins in order to initialize the terminal.
bl	Clear the current line from the present cursor position through the end of the line without moving the cursor.
bm	Place the terminal in "background" mode. In this mode characters are written to the terminal with a lower intensity (brightness) than usual.
fm	Place the terminal in foreground (normal) mode.
pc	Position the cursor to an absolute location.
ku	Sent by the terminal in response to the up-arrow key.
kd	Sent by the terminal in response to the down-arrow key.
kl	Sent by the terminal in response to the left-arrow key.
kr	Sent by the terminal in response to the right-arrow key.
kh	Sent by the terminal in response to the home key.
k0-k9	Sent by the terminal in response to other special keys.

The following entry describes a ct82 terminal manufactured by Southwest Technical Products, Corporation:

```
ct82:ho=16 cu=01 cd=02 cr=09 cl=04 cs=30+07+12 nr=20 nc=82
     ku=01 kd=02 kr=09 kl=04 is=28+18+30+19+30+20+30+07 bl=06
     bm=28+05  fm=28+21+30+07  wt=1 pc=$0b+$ff+$01+$ff+$80:
```

Not all terminals can support all the functions described here. All the

(continued)

information required to create the list of values should be contained in the manual describing the particular terminal. As can be seen from the previous example, definitions for all keywords are not necessary for the correct functioning of the utilities that use the file "termcap". However, definitions for the following keywords are essential:

cs, ho, nr, nc, and either pc or cu, cd, cl, and cr

The keyword "pc" enables a utility to position the cursor to any absolute location on the screen. Some terminals do not support this feature, in which case the utilities must use relative positioning of the cursor (using ho, cu, cd, cl, and cr) instead. Since the absolute positioning of the cursor is different on almost all terminals, the value string associated with the keyword "pc" is rather complex. At a minimum, absolute cursor positioning requires the desired row and column numbers to be sent to the terminal as part of a control sequence. Different terminals require the row and column portions of the sequence to be in different forms. There must be a method of transforming the desired row and column numbers, supplied in the range of 0 to some maximum, into the form required by a specific terminal. Two special portions of the the "pc" value string, called escape sequences, accomplish this transformation. Each escape sequence is replaced by the row or column number in the proper form, as defined by the information in the escape sequence. The escape sequence has the following format:

$ff+<flag>[+<bias>]

where <flag> is a set of 8 bits which describes the particular operation. These bits are explained in the following table (0 is the rightmost bit):

Bit	Value	Meaning
0	0	This is a row reference.
	1	This is a column reference.
1	0	No bias is necessary.
	1	A bias must be added to the value before use.
2	0	Use the value as is.
	1	Subtract the value from the maximum row or column before use.
3	0	Do not convert the value to BCD.
	1	Convert the value to BCD.
4	0	Do not convert the value to decimal ASCII.
	1	Convert the value to decimal ASCII.
5	***	Unused. Must be 0.
6	***	Unused. Must be 0.
7	0	Value required if bits 0-6 are not all 0.
	1	Value required if bits 0-6 are all 0.

A bias must be specified as part of the escape sequence if an only if bit 1 of the flag is set to 1. Some examples should make this mechanism

(continued)

clearer. Consider the following "pc" string for the ct82:

pc=$0b+$ff+$01+$ff+$80

The $0b is required by the ct82 to initiate cursor positioning. The $ff is the start of an escape sequence. Its flag of $01 means that bit 0 is 1, and all o ther bits are 0. Thus, the escape sequence is a row reference with no bias. The binary value is used as is. It is not converted to either binary coded decimal (BCD) or decimal ASCII. The second $ff starts the next escape sequence. Its flag of $80 means that all bits from bit 0 through bit 6 are 0. Bit 7 is 1, as it must be if all other bits are 0. This flag is a column reference (bit 0 is 0), but in all other respects it means the same thing as the previous flag.

In order to position the cursor on a ct82 terminal to row 18, column 10, a utility must send the following characters to the terminal:

Character	Meaning
$0b	Initiate cursor positioning.
$09	Column 10 (0 is the leftmost column).
$11	Row 18 (0 is the uppermost row).

As another example consider the following cursor-positioning string for an Ambassador terminal manufactured by Ann Arbor Terminals:

pc=$1b+$5b+$ff+$12+$01+$3b+$ff+$13+$01+$48

The $1b and $5b are required by this terminal to initiate cursor positioning. The $ff begins the first escape sequence. Its flag of $12 means that bits 1 and 4 are 1, and all other bits are 0. Thus, the escape sequence is a row reference with a bias (the next number in the value list, $01) that must be added to the binary value before use. The binary value is used as is. It should be sent in decimal ASCII. The $3b following the bias is required by this terminal as a row/column separator. The second $ff starts the second escape sequence. Its flag of $13 means the same thing as a flag of $12 except that bit 0 is 1, indicating that this flag is a column reference. The $48 immediately following the bias of $01 is required by this terminal to terminate cursor positioning.

In order to position the cursor on this terminal to row 18, column 10, a utility must send the following characters to the terminal:

(continued)

Character	Meaning
$1b $5b	Initiate cursor positioning.
$31 $38	Row 18 in decimal ASCII (0 relative, +1 bias).
$3b	Row/column separator.
$31 $30	Column 10 in decimal ASCII (0 relative, +1 bias).
$48	Terminate cursor positioning.

As a final example, consider the cursor-positioning string for an Infoton 100 terminal from Infoton:

pc=$1b+$66+$ff+$03+$20+$ff+$02+$20

The $1b and the $66 are required by the terminal to initiate cursor positioning. The $ff starts the first escape sequence. Its escape flag of $03 means that bits 0 and 1 are 1, and all other bits are 0. Thus, the escape sequence is a column reference with a bias of $20 that must be added to the binary value before use. The next escape sequence has the flag $02, which means that only bit 1 is 1. All other bits are 0. Thus, this escape sequence is a row reference with a bias of $20. In order to position the cursor on this terminal to row 18, column 10, a utility must send the following characters to the terminal:

Character	Meaning
$1b $66	Initiate cursor positioning.
$29	Column 10 (0 relative, +$20 bias).
$31	Row 18 (0 relative, +$20 bias).

Format of the File "ttyassoc"

The second file used by the "crt_termcap" command contains a list indicating what type of terminal is actually connected to each port on the system. This file, called "ttyassoc" for terminal association file, can be created very simply using the file "/etc/ttylist", which is supplied on all systems. The file "ttylist" can, in fact, be used as the file "ttyassoc" if it is modified to appear exactly as described in this section. All that is required in "ttylist" is a plus or minus sign, '+' or '-', in column 1, followed by a space, followed by a two-digit number representing a terminal. The file "ttyassoc" requires two additional fields. Existing programs which use the file "ttylist" ignore anything beyond the terminal number, so the file can be modified to look like the file "ttyassoc" without affecting the rest of the system.

The file "ttyassoc" contains one line for each terminal on the system. Each line has the following format:

(continued)

+ <nn> : <terminal_type> : [<name>] :

An explanation of this format follows:

<nn>	A two-digit number representing the terminal.
<terminal_type>	The type of terminal attached to the port. This name should be one of the terminals described in the file "ttycap". The name may contain as many as ten characters.
<name>	A descriptive name, used primarily for documentation. This name is normally the name of the person most commonly using the terminal. However, it has no functional meaning and need not be present.

If the file "/etc/ttylist" is used as the "ttyassoc" file, inactive ports require as an absolute minimum the entry

- nn:::

for the "crt_termcap" command to function properly. The colons used as field separators must appear even if the fields are empty.

To access the file "ttylist" as the file "ttyassoc" the user must specify "/etc/ttylist" as the second argument on the command line.

EXAMPLES

1. /etc/crt_termcap /etc/ttycap /etc/ttyassoc /etc/termcap

This example combines the information in the files "/etc/ttycap" and "etc/ttyassoc" to form the file "/etc/termcap".

NOTES

- All three arguments must be supplied; there are no default file names. Arguments must appear in the order specified in the syntax statement.
- The user can specify any file names as the three arguments to "crt_termcap".
- All utilities which use the "termcap" functions expect the file "/etc/termcap" to exist. Although any name can be specified as the third argument to the "crt_termcap" command, if the output file is not named "/etc/termcap", it should be linked to a file that is.

(continued)

ERROR MESSAGES

*** Can't access ttyassoc file "<file_name>"
The utility did not have read permissions in the file specified as the "ttyassoc" file.

*** Can't access ttycap file "<file_name>"
The utility did not have read permissions in the file specified as the "ttycap" file.

Can't find description of the terminal "<term_name>".
A terminal name specified in the "ttyassoc" file was not one of the terminal names contained in the "ttycap" file. The "termcap" file will not be created.

*** ERROR : <system_error_message> while <action>
This general class of error messages describes any system errors encountered while performing such functions as reading, writing, opening, or closing files.

Syntax: crt_termcap <ttycap_file> <ttyassoc_file> <termcap_file>
The utility expects exactly three arguments. This message indicates that the argument count is wrong.

*** Unrecognized option "<char_1>", Terminal = <terminal_name>,
Last valid option "<char_2>",
The option shown is not one of the legal options allowed in the "ttycap" file. The file "termcap" will not be built.

crt_termcap

Create a file defining the capabilities of each terminal on the system.

SYNTAX

/etc/crt_termcap <ttycap_file> <ttyassoc_file> <termcap_file>

DESCRIPTION

The "crt_termcap" command creates a file ("termcap") which describes the capabilities of each terminal on the system. This file makes it possible for programs to operate on many different terminals regardless of the individual characteristics of the terminals. The "crt_termcap" command must be used to create the file "termcap" before any programs which need that file can operate successfully.

Arguments

<ttycap_file>	File describing functional capabilities of each terminal.
<ttyassoc_file>	File associating each active port with a particular kind of terminal.
<termcap_file>	Output file combining information from the two input files.

The "crt_termcap" command uses two input files to produce one output file. The first input file describes the functional capabilities of each terminal on the system. The second file indicates what type of terminal is on each active port in the system. These files must each conform to a particular format. The following two sections describe these formats.

Format of the File "ttycap"

The file "ttycap" contains one logical entry for each terminal on the system. The format of an entry is

<terminal_name> : <capability_list> :

where <terminal_name> is a character string (it may contain as many as ten characters) which identifies the terminal, and <capability_list> is a list describing the capabilities of the terminal. Each item in this list has the following format:

<keyword> = <value_list>

where <keyword> is a two-character sequence representing a function such

(continued)

as clearing the screen and <value_list> is a list containing decimal values, hexadecimal values, or both. Each value in the list must be separated from the following one by a plus sign, '+'. No spaces may appear between the values and the plus signs. All hexadecimal values must consist of two digits preceded by a dollar sign, '$'. The following are valid value strings:

```
1+2+3
$01+$ff+$80
```

The keywords currently supported are

ho	Home cursor.
cu	Move cursor up without modifying display.
cd	Move cursor down.
cl	Move cursor left.
cr	Move cursor right.
cs	Clear entire screen.
nr	Number of rows on screen (first row is 1).
nc	Number of columns on screen (first column is 1).
wt	Number of seconds to wait between clearing the screen function and sending more information to the terminal.
is	A string of characters sent to the terminal when processing begins in order to initialize the terminal.
bl	Clear the current line from the present cursor position through the end of the line without moving the cursor.
bm	Place the terminal in "background" mode. In this mode characters are written to the terminal with a lower intensity (brightness) than usual.
fm	Place the terminal in foreground (normal) mode.
pc	Position the cursor to an absolute location.
ku	Sent by the terminal in response to the up-arrow key.
kd	Sent by the terminal in response to the down-arrow key.
kl	Sent by the terminal in response to the left-arrow key.
kr	Sent by the terminal in response to the right-arrow key.
kh	Sent by the terminal in response to the home key.
k0-k9	Sent by the terminal in response to other special keys.

The following entry describes a ct82 terminal manufactured by Southwest Technical Products, Corporation:

```
ct82:ho=16 cu=01 cd=02 cr=09 cl=04 cs=30+07+12 nr=20 nc=82
     ku=01 kd=02 kr=09 kl=04 is=28+18+30+19+30+20+30+07 bl=06
     bm=28+05  fm=28+21+30+07  wt=1 pc=$0b+$ff+$01+$ff+$80:
```

Not all terminals can support all the functions described here. All the information required to create the list of values should be contained in the manual describing the particular terminal. As can be seen from the previous example, definitions for all keywords are not necessary for the

(continued)

correct functioning of the utilities that use the file "termcap". However, definitions for the following keywords are essential:

cs, ho, nr, nc, and either pc or cu, cd, cl, and cr

The keyword "pc" enables a utility to position the cursor to any absolute location on the screen. Some terminals do not support this feature, in which case the utilities must use relative positioning of the cursor (using ho, cu, cd, cl, and cr) instead. Since the absolute positioning of the cursor is different on almost all terminals, the value string associated with the keyword "pc" is rather complex. At a minimum, absolute cursor positioning requires the desired row and column numbers to be sent to the terminal as part of a control sequence. Different terminals require the row and column portions of the sequence to be in different forms. There must be a method of transforming the desired row and column numbers, supplied in the range of 0 to some maximum, into the form required by a specific terminal. Two special portions of the the "pc" value string, called escape sequences, accomplish this transformation. Each escape sequence is replaced by the row or column number in the proper form, as defined by the information in the escape sequence. The escape sequence has the following format:

$ff+<flag>[+<bias>]

where <flag> is a set of 8 bits which describes the particular operation. These bits are explained in the following table (0 is the rightmost bit):

Bit	Value	Meaning
0	0	This is a row reference.
	1	This is a column reference.
1	0	No bias is necessary.
	1	A bias must be added to the value before use.
2	0	Use the value as is.
	1	Subtract the value from the maximum row or column before use.
3	0	Do not convert the value to BCD.
	1	Convert the value to BCD.
4	0	Do not convert the value to decimal ASCII.
	1	Convert the value to decimal ASCII.
5	***	Unused. Must be 0.
6	***	Unused. Must be 0.
7	0	Value required if bits 0-6 are not all 0.
	1	Value required if bits 0-6 are all 0.

A bias must be specified as part of the escape sequence if an only if bit 1 of the flag is set to 1. Some examples should make this mechanism clearer. Consider the following "pc" string for the ct82:

pc=$0b+$ff+$01+$ff+$80

(continued)

The $0b is required by the ct82 to initiate cursor positioning. The $ff is the start of an escape sequence. Its flag of $01 means that bit 0 is 1, and all other bits are 0. Thus, the escape sequence is a row reference with no bias. The binary value is used as is. It is not converted to either binary coded decimal (BCD) or decimal ASCII. The second $ff starts the next escape sequence. Its flag of $80 means that all bits from bit 0 through bit 6 are 0. Bit 7 is 1, as it must be if all other bits are 0. This flag is a column reference (bit 0 is 0), but in all other respects it means the same thing as the previous flag.

In order to position the cursor on a ct82 terminal to row 18, column 10, a utility must send the following characters to the terminal:

Character	Meaning
$0b	Initiate cursor positioning.
$09	Column 10 (0 is the leftmost column).
$11	Row 18 (0 is the uppermost row).

As another example consider the following cursor-positioning string for an Ambassador terminal manufactured by Ann Arbor Terminals:

pc=$1b+$5b+$ff+$12+$01+$3b+$ff+$13+$01+$48

The $1b and $5b are required by this terminal to initiate cursor positioning. The $ff begins the first escape sequence. Its flag of $12 means that bits 1 and 4 are 1, and all other bits are 0. Thus, the escape sequence is a row reference with a bias (the next number in the value list, $01) that must be added to the binary value before use. The binary value is used as is. It should be sent in decimal ASCII. The $3b following the bias is required by this terminal as a row/column separator. The second $ff starts the second escape sequence. Its flag of $13 means the same thing as a flag of $12 except that bit 0 is 1, indicating that this flag is a column reference. The $48 immediately following the bias of $01 is required by this terminal to terminate cursor positioning.

In order to position the cursor on this terminal to row 18, column 10, a utility must send the following characters to the terminal:

Character	Meaning
$1b $5b	Initiate cursor positioning.
$31 $38	Row 18 in decimal ASCII (0 relative, +1 bias).
$3b	Row/column separator.
$31 $30	Column 10 in decimal ASCII (0 relative, +1 bias).
$48	Terminate cursor positioning.

(continued)

As a final example, consider the cursor-positioning string for an Infoton 100 terminal from Infoton:

pc=$1b+$66+$ff+$03+$20+$ff+$02+$20

The $1b and the $66 are required by the terminal to initiate cursor positioning. The $ff starts the first escape sequence. Its escape flag of $03 means that bits 0 and 1 are 1, and all other bits are 0. Thus, the escape sequence is a column reference with a bias of $20 that must be added to the binary value before use. The next escape sequence has the flag $02, which means that only bit 1 is 1. All other bits are 0. Thus, this escape sequence is a row reference with a bias of $20. In order to position the cursor on this terminal to row 18, column 10, a utility must send the following characters to the terminal:

Character	Meaning
$1b $66	Initiate cursor positioning.
$29	Column 10 (0 relative, +$20 bias).
$31	Row 18 (0 relative, +$20 bias).

Format of the File "ttyassoc"

The second file used by the "crt_termcap" command contains a list indicating what type of terminal is actually connected to each port on the system. This file, called "ttyassoc" for terminal association file, can be created very simply using the file "/etc/ttylist", which is supplied on all systems. The file "ttylist" can, in fact, be used as the file "ttyassoc" if it is modified to appear exactly as described in this section. All that is required in "ttylist" is a plus or minus sign, '+' or '-', in column 1, followed by a space, followed by a two-digit number representing a terminal. The file "ttyassoc" requires two additional fields. Existing programs which use the file "ttylist" ignore anything beyond the terminal number, so the file can be modified to look like the file "ttyassoc" without affecting the rest of the system.

The file "ttyassoc" contains one line for each terminal on the system. Each line has the following format:

+ <nn> : <terminal_type> : [<name>] :

An explanation of this format follows:

<nn>	A two-digit number representing the terminal.
<terminal_type>	The type of terminal attached to the port. This name should be one of the terminals described in the file "ttycap". The name may contain as many as ten characters.

(continued)

<name>	A descriptive name, used primarily for documentation. This name is normally the name of the person most commonly using the terminal. However, it has no functional meaning and need not be present.

If the file "/etc/ttylist" is used as the "ttyassoc" file, inactive ports require as an absolute minimum the entry

```
- nn:::
```

for the "crt_termcap" command to function properly. The colons used as field separators must appear even if the fields are empty.

To access the file "ttylist" as the file "ttyassoc" the user must specify "/etc/ttylist" as the second argument on the command line.

EXAMPLES

```
/etc/crt_termcap /etc/ttycap /etc/ttyassoc /etc/termcap
```

This example combines the information in the files "/etc/ttycap" and "etc/ttyassoc" to form the file "/etc/termcap".

NOTES

- All three arguments must be supplied; there are no default file names. Arguments must appear in the order specified in the syntax statement.
- The user can specify any file names as the three arguments to "crt_termcap".
- All utilities which use the "termcap" functions expect the file "/etc/termcap" to exist. Although any name can be specified as the third argument to the "crt_termcap" command, if the output file is not named "/etc/termcap", it should be linked to a file that is.

ERROR MESSAGES

usage : ++ crt_termcap ttycap ttyassoc termcap
The utility expects exactly three arguments. This message indicates that the argument count is wrong.

*** ERROR : <system_error_message> while <action>
This general class of error messages describes any system errors encountered while performing such functions as reading, writing, opening, or closing files.

(continued)

*** Can't access ttycap file "<file_name>"
The utility did not have read permissions in the file specified as the "ttycap" file.

*** Can't access ttyassoc file "<file_name>"
The utility did not have read permissions in the file specified as the "ttyassoc" file.

*** Unrecognized option "<char_1>", Terminal = <terminal_name>,
Last valid option "<char_2>".
The option shown is not one of the legal options allowed in the "ttycap" file. The file "termcap" will not be built.

*** Can't find description of the terminal "<term_name>".
A terminal name specified in the "ttyassoc" file was not one of the terminal names contained in the "ttycap" file. The "termcap" file will not be created.

dircompare

Compare two directories and list the files found in one directory but not in the other.

SYNTAX

```
dircompare <dir_1> <dir_2>
```

DESCRIPTION

The "dircompare" command compares the names in the directory <dir_1> with the names in the directory <dir_2>. It first reports any names found in <dir_1> but not in <dir_2>, then reports any names found in <dir_2> but not in <dir_1>. It sorts the lists of names alphabetically.

Arguments

<dir_1>	First directory to compare.
<dir_2>	Second directory to compare.

EXAMPLES

1. dircompare /proj/utils /rls/utl
2. dircompare . ../bkup

The first example compares the names in the directory "/proj/utils" with the names in the directory "/rls/utl".

The second example compares the names in the directory "." (the working directory) with the names in the directory "bkup", which is found in the parent of the working directory ("..").

NOTES

- The "dircompare" command compares neither the characteristics nor the contents of the entries found in the directories specified. It compares only the names in the directories.

ERROR MESSAGES

usage: dircompare <dir_1> <dir_2>
The "dircompare" command requires exactly two arguments. This message indicates that the argument count is wrong.

(continued)

Not a directory: <arg>
The argument <arg> is not a directory and cannot be used as an argument to the "dircompare" command.

Directories are too large to compare
Not enough memory is available to contain the two specified directories. The combined number of entries must exceed 3,000 for this error to occur.

Read error on directory: <dir_name>
The UniFLEX Operating System reported a read error while trying to read from the directory <dir_name>.

MESSAGES

These are in <dir_1> but not in <dir_2>
The names that follow were found in the directory <dir_1> but not in the directory <dir_2>. This message is written to standard output.

All entries in <dir_1> are in <dir_2>
All of the names in the directory <dir_1> are also in the directory <dir_2>. This message is written to standard output.

Directories <dir_1> and <dir_2> are identical
The directories <dir_1> and <dir_2> contain exactly the same names. This message is written to standard output.

SEE ALSO

ls

dirname

Extract the directory name from a path name.

SYNTAX

dirname <path_name>

DESCRIPTION

The "dirname" command extracts the directory-name prefix from a path name and writes it, followed by a carriage return, to standard output.

Arguments

<path_name>	The path name from which to derive a directory name.

EXAMPLES

1. dirname hello.c
2. dirname /usr/joe/docs.txt
3. dirname ../jan/sorted_data

The first example writes only a carriage return to standard output since no directory name is prefixed to the string "hello.c".

The second example writes the string "/usr/joe", followed by a carriage return, to standard output.

The third example writes the string "../jan", followed by a carriage return, to standard output.

NOTES

- If the argument <path_name> does not contain a slash character, '/', "dirname" writes only a carriage return to standard output.
- If the argument <path_name> is just a slash character (indicating the root directory of the file system), "dirname" writes a slash, followed by a carriage return, to standard output.

ERROR MESSAGES

usage: dirname <path_name>
The "dirname" command requires exactly one argument. This message indicates that the argument count is wrong.

(continued)

SEE ALSO

basename

diskinfo

Display information about the size and contents of the specified disk.

SYNTAX

```
diskinfo <device_name> [<device_name_list>]
```

DESCRIPTION

The "diskinfo" command displays information about the specified disk or list of disks. The information includes the following:

- Disk name
- Date of creation
- Date of last update
- Disk size
- Size of fdn space
- Size of file space
- Size of swap space
- Free space remaining
- Free fdns remaining
- Disk type (e.g., Double-sided, single-density floppy)

The disk from which the information is being obtained need not be mounted on the file system.

Arguments

<device_name> Name of block disk device.

EXAMPLES

```
diskinfo /dev/fd1
```

This example displays information about the floppy disk inserted in device "/dev/fd1" (floppy disk drive #1).

NOTES

- The determination of disk type is based on disk types known to "diskinfo" at the time of its release. If <device_name> refers to a nonstandard disk, the disk type displayed may not be valid.

(continued)

ERROR MESSAGES

"<device_name>" is not a block device.
The argument specified as the "<device_name>" is not the name of a block device.

Can't get status for "<device_name>".
An error occurred while trying to obtain the status of the specified device.

Can't open "<device_name>".
The program was unable to open the specified device for reading.

Can't read SIR.
An I/O error occurred while attempting to read the System Information Record (block 1) from the specified device.

Must specify device.
No device was specified on the command line.

dsd

Produce a dynamic display of system use.

SYNTAX

dsd [<sleep_time>]

DESCRIPTION

The "dsd" command, which stands for "dynamic system display", displays the current status of most of the system resources--including CPU usage, disk usage, and memory usage. At the top of the screen "dsd" displays a dynamic bar graph, representing the overall system load. The longer the line, the busier the system. The command updates the display every five seconds unless the user specifies another length of time.

Most items displayed are self-explanatory. The boot time is defined as the time at which the system manager entered the date and time. The disk efficiency is the ratio of the number of times UniFLEX found the block it needed in its buffer cache to the number of times it had to go to the disk to get a block. The maximum possible efficiency is approximately 67%. "Swap busy" represents the amount of swap space that is currently in use.

The "dsd" command requires the "termcap" facility to be installed on the system. It also requires a terminal with at least twenty lines per screen. On a terminal with less than twenty-four lines "dsd" does not display swap statistics. The user terminates the display by typing a keyboard interrupt (control-C).

Arguments

<sleep_time>	The amount of time (in seconds) to sleep between displays. Default is 5.

EXAMPLES

```
dsd 30
```

This example displays the system information on the user's terminal and updates it every thirty seconds.

(continued)

ERROR MESSAGES

Syntax error - dsd [<sleep_time>]
The "dsd" command expects no more than one argument. This message indicates that the argument count is wrong.

Can't initialize terminal from "termcap" data.
Either "dsd" could not find the file "termcap" or the user's terminal is not described in the file.

Needs at least 20 lines on the terminal.
The terminal being used displays less than twenty lines at a time.

Can't open system memory.
The system-memory file, "/dev/smem", could not be opened.

Can't open physical memory.
The system physical-memory file, "/dev/pmem", could not be opened.

SEE ALSO

status

filedevice

Report the name of the device on which a specified file resides.

SYNTAX

```
filedevice <file_name> [<file_name_list>]
```

DESCRIPTION

The "filedevice" command reports the full device name of the disk device on which the specified file or files reside.

Arguments

<file_name> Name of file to report on.

EXAMPLES

```
filedevice .
```

This example prints the full name of the device on which the working directory (´.´) resides.

ERROR MESSAGES

Can´t determine device for "<file_name>".
The program was unable to find the name of the device which corresponds to the device number associated with the specified file. This error should never occur and indicates a serious problem.

Can´t obtain status for "<file_name>".
An error occurred while trying to obtain the status of the specified file.

Can´t open "/dev" directory.
An error occurred while trying to open the "/dev" directory. This directory must be read in order to determine the device names.

Error reading "/dev" directory.
An error occurred while trying to read the "/dev" directory. This directory must be read in order to determine the device names.

Must specify file.
No file was specified on the command line.

flex-rel

Convert a UniFLEX relocatable binary file to a file containing FLEX relocatable binary code.

SYNTAX

```
flex-rel <UniFLEX_file_name> <FLEX_file_name>
```

DESCRIPTION

The "flex-rel" command converts a UniFLEX relocatable binary file, created by the program "relasmb", to a file containing FLEX relocatable binary code. The main purpose of this command is to allow the development of FLEX relocatable binary modules under the UniFLEX operating system. Note that the output of the "flex-rel" command is a UniFLEX file whose contents are in the FLEX relocatable binary format. This file may then be transported to a FLEX disk using the "flex-copy" utility or other means. Since the FLEX relocating assembler and linking-loader do not support segmentation, "flex-rel" combines the TEXT and DATA portions of the UniFLEX file. It also replaces any BSS segments by an appropriate number of zeros in the FLEX file.

Arguments

<UniFLEX_file_name>	The UniFLEX relocatable binary file to convert.
<FLEX_file_name>	The resulting FLEX relocatable binary file.

EXAMPLES

```
flex-rel file.r file.f
```

This command converts the UniFLEX relocatable binary file "file.r" into the FLEX relocatable binary file "file.f". Note that the output from this command is a UniFLEX file which must then be transported to a FLEX system.

ERROR MESSAGES

Invalid UniFLEX binary header
The UniFLEX file was not recognizable as a relocatable binary file.

File contains negative linking
The format for a FLEX relocatable binary file does not include the provision for negative linking. If the UniFLEX file uses this feature, "flex-rel" cannot transform it.

(continued)

SEE ALSO

link-edit
flex-copy
relasmb

hangup

Specify the action that the nshell program is to take when it receives a hangup interrupt.

SYNTAX

```
hangup <on_or_off>
```

DESCRIPTION

The "hangup" command, which is part of the nshell program, specifies whether or not the nshell program should terminate if it receives a hangup interrupt. When hangup is "on", the nshell program terminates on receipt of a hangup interrupt. When hangup is "off", the nshell program ignores hangup interrupts.

Arguments

<on_or_off>	Either the string "on" or the string "off".

EXAMPLES

1. hangup on
2. hangup off

The first example instructs the nshell program to terminate if a hangup interrupt is received.

The second example instructs the nshell program to ignore hangup interrupts.

NOTES

- The "hangup" command is only effective while the nshell program under which it is invoked is running. The "on" or "off" condition is propagated to all child tasks of the nshell program. That is, if hangup is "off", all child tasks of the nshell program ignore the hangup interrupt unless they specifically take action to handle the interrupt. If hangup is "on", all child tasks of the nshell program terminate on receiving a hangup interrupt unless they specifically take action to handle the interrupt.

SEE ALSO

nshell

keep

Retain files in the working directory.

SYNTAX

```
keep <file_name> [<file_name_list>] [+pq]
keep +p [+q]
```

DESCRIPTION

The "keep" command deletes all data files in the working directory except those whose names are specified as arguments or whose names start with a period.

Arguments

<file_name>	Name of one file to be kept.
<file_name_list>	Names of additional files to be kept.

Description of the Arguments

The file names specified as arguments must refer to files existing in the working directory. The names may not contain path information. If no file names are specified, the 'p' option, described later, must be specified.

If one or more of the file names specified as arguments either do not exist in the working directory or refer to devices or directories, "keep" issues messages to that effect and requests permission to continue from the user. The format of this request is

Continue?

The user must respond by typing a string followed by a carriage return, or by typing the end-of-file character. If the first character of the string is 'y', "keep" begins to delete files, leaving devices or directories intact. If the first character of the string is 'n', or if the user types the end-of-file character as the first character of a line, "keep" terminates. If the first character of the string is not one of these three characters, "keep" reissues the prompt.

Options Available

p	Prompt before deleting each file.
q	Use quiet mode.

(continued)

Complete descriptions of the options follow.

The ´p´ Option.

If the ´p´ option is specified, "keep" asks the user for permission to delete each data file in the working directory except those whose names appear as arguments or whose names begin with a period. The format of this request is

Delete "<file_name>"?

The user must respond to each request by typing a string followed by a carriage return or by typing the end-of-file character. If the first character of the string is ´y´, "keep" deletes the file; if it is ´n´, the file is not deleted. If the string starts with any other character except the end-of-file character, "keep" repeats the request. If the user types the end-of-file character as the first character on the line, "keep" terminates.

The ´q´ Option.

Normally "keep" lists the names of the files as they are deleted. Specifying the ´q´ option suppresses this list of names.

EXAMPLES

1. keep datafile
2. keep +p
3. keep *.c *.h +pq

In the first example, "keep" deletes all data files in the working directory except the file named "datafile" and any files whose names begin with a period. Messages are issued naming the files that are deleted.

In the second example, "keep" prompts the user for permission to delete every data file in the working directory except those whose names begin with a period. The user must grant or deny this permission for each file.

In the third example, "keep" requests permission to delete all files whose names do not end in ".c" or ".h" except those whose names begin with a period. The user must grant or deny this permission for each file. The ´q´ option suppresses the informative messages which list the names of the files as they are deleted.

(continued)

NOTES

- It is not possible to use "keep" to delete a directory, a device, or a file whose name begins with a period.

- Fatal error messages are sent to standard error; prompts and informative messages, to standard output.

ERROR MESSAGES

Error reading working directory.
The operating system returned an error when "keep" tried to read the working directory. This message is preceded by an interpretation of the error.

Cannot open working directory.
The operating system returned an error when "keep" tried to open the working directory. This message is preceded by an interpretation of the error.

No permission to delete files in this directory.
The user does not have write permission for the working directory.

Unknown option "<char>" ignored.
The option specified by <char> is not a valid option to "keep". This error is not fatal.

Argument "<file_name>" contains path information.
Arguments to the "keep" command may not contain path information (elements separated by the slash character, '/').

"<file_name>" not located.
The specified file does not exist in the working directory.

"<file_name>" is a device or a directory.
The specified name refers to a device or a subdirectory.

Cannot delete "<file_name>".
The operating system returned an error when "keep" tried to delete the specified file. This message is preceded by an interpretation of the error.

SEE ALSO

kill

libinfo

Display information about a library.

SYNTAX

libinfo <library_name> [<library_name_list>] [+emM]

DESCRIPTION

The "libinfo" command produces a list of the entry points and module names contained in a library produced by the "lib-gen" command. The user can optionally display only the entry points or only the module names. Information about a particular module within a library can also be displayed.

Options Available

e	Display only entry points in the specified library.
m	Display only module names in the specified library.
M=<mod_name>	Display information about module "<mod_name>".

EXAMPLES

1. libinfo testlib
2. libinfo runlib +m
3. libinfo /lib/mathlib +M=Arctan

The first example lists all entry points and module names in the library "testlib".

The second example lists all the module names contained in the library "runlib".

The third example displays the entry points and module names in the module "Arctan" in the library "/lib/mathlib".

NOTES

- The ´M´ option is incompatible with both the ´e´ and ´m´ options. If the user specifies incompatible options, "libinfo" uses the ´M´ option and ignores any others.

(continued)

ERROR MESSAGES

Unknown option ´<char>´ ignored.
An unknown option was found and ignored.

***** ´M´ taken, others ignored *****
The ´m´ and ´e´ options are incompatible with the ´M´ option. If the user specifies incompatible options, "libinfo" uses the ´M´ option and ignores any others.

Error opening ´<file_name>´ : <reason>
The operating system returned an error when "libinfo" tried to open the specified file.

Error reading ´<file_name>´ : <reason>
The operating system returned an error when "libinfo" tried to read the specified file.

Error seeking to <location> in ´<file_name>´ : <reason>
The operating system returned an error when "libinfo" tried to seek to the specified location (in hexadecimal) in the specified file.

´<file_name>´ is not a library!
The file specified does not have the correct format for a library created with the "lib-gen" command.

SEE ALSO

lib-gen
relinfo

more

Display ASCII data with user control.

SYNTAX

more [<file_name_list>]

DESCRIPTION

The "more" command displays data on the user's terminal. It lets the user both control the number of lines displayed at a time and skip lines.

If the list of file names is omitted, the "more" command accepts data from standard input. It displays enough lines to fill the terminal's screen, then prompts for a command. If the list of file names contains a single name, the "more" command displays enough lines to fill the terminal's screen, then prompts for a command. The prompt contains the percentage of the file that has been displayed. If the list of file names contains multiple names, the "more" command introduces each file with a prompt and indicates when it reaches the end of each file.

Arguments

<file_name_list>	The list of files to display with user control.

User Control

The "more" command prompts the user for a command with the prompt "More? ". Unless "more" is reading from standard input, this prompt is preceded by either the percentage of the file listed, "(<n>%) ", or the message, "Beginning: <file_name> " where <file_name> is the name of the file whose contents are about to be to displayed. In response to the prompt, the user types a single-character command telling the "more" command what to do next. The single-character command should not be followed by a carriage-return. The "more" command sends a control-G (bell) to the terminal if the character typed is not a command. A list of commands follows.

The space command, ' ', starts at the next line (or at the first line of the next file) and displays lines until it either fills the screen or reaches the end of the file.

The period command, '.', starts at the next line (or with the first line of the next file) and displays lines until it either displays enough lines to scroll half of the screen or

(continued)

reaches the end of the file.

The carriage-return command displays the next line (or the first line of the next file) if there is one.

The 's' or '/' response requests a search for a character string. When the "more" command issues the prompt, "Search string? ", the user should type the string to find, followed by a carriage return. The "more" command starts at the next line (or with the first line of the next file) and searches for that string. If it finds the string, it displays lines, starting with the line in which the search string first appears, until it either fills the screen or reaches the end of the file. If it does not find the string, it stops searching and begins to display the next file if there is one.

If "more" cannot accept a character, it sends a control-G (bell) to the terminal. The command does not accept control characters. Nor does it accept any characters after it fills the search-string buffer. Typing a character-delete character (usually a control-H) as the first character in response to the prompt, "Search string? ", or typing a line-delete character (usually a control-X) any time while entering the search string returns the "more" command to the "More? " prompt.

The 'n' response stops processing the current file and begins processing the next file in the list of file names, if there is one.

The 'q' response ends the "more" command. An end-of-file character (control-D) performs the same function.

EXAMPLES

1. more hello.c
2. more *.c
3. list hello.c | more

The first example displays the file "hello.c" at the terminal with user control. It first clears the screen, then lists enough lines from the file to fill the screen. It then requests a command from the user by issuing a prompt that indicates the percentage of the file that it has displayed.

The second example displays at the terminal with user control all of the files in the working directory whose names contain the suffix ".c" . It first clears the screen, then introduces the first file by issuing a prompt containing its name. This prompt is a request for a command. After executing the first command, "more" prompts the user for another

(continued)

command with a prompt that indicates the percentage of the file that it has already displayed. When "more" reaches the end of the first file, it introduces the next file. This process continues until "more" has processed all the files in the list.

The third example displays at the terminal with user control the output from the "list" command. It first clears the screen, then lists enough lines to fill the screen. After filling the screen, "more" prompts the user for a command.

NOTES

- The "more" command uses the UniFLEX terminal capabilities information ("termcap") if that information is available for the terminal being used. If the "more" command seems to be handling a terminal poorly, the system manager should verify that the terminal capabilities for that terminal are correctly set.

- If no terminal capabilities are available for the terminal in use, the "more" command assumes that there are eighty columns to a line and twenty lines on the screen. It also assumes that the backspace character (hexadecimal 08) moves the cursor one place to the left and that the space character (hexadecimal 20) moves the cursor one place to the right and clears the character at that place.

- The "more" command does not use the last column of a line. Some terminals automatically advance to the next line after writing to the last column of a line; others do not. The last column is not used to avoid having to differentiate between the two types of terminals. Lines longer than the width of the terminal are split and displayed as two lines.

- The "more" command displays all control characters as "^X" where 'X' is the key which, if struck while the "control" key is depressed, normally produces that control character. For example, it displays each embedded tab character (control-I) as "^I".

- The "more" command automatically clears the screen before displaying any data.

ERROR MESSAGES

Broken pipe
The "more" command caught a broken-pipe interrupt. A broken-pipe interrupt immediately stops the "more" command.

File is not a regular file: <file_name>
The file specified exists but is not a regular data file. The "more" command works only with regular data files.

(continued)

Hang up
The "more" command caught a hang-up interrupt. A hang-up interrupt immediately stops the "more" command.

Input must come from a file or a pipe
Data from standard input must come from a pipe or a redirected data file. This message indicates that standard input is something other than a pipe or a redirected data file, such as a terminal.

INTERRUPT!
The "more" command caught a keyboard interrupt. A keyboard interrupt immediately stops the "more" command.

Output must go to a terminal
Standard output must be a character-special file (i.e., a terminal). This message indicates that it is not.

Quit
The "more" command caught a quit interrupt. A quit interrupt immediately stops the "more" command.

SEE ALSO

list
page

newuser

Temporarily login as a new user.

SYNTAX

newuser [<user_name>]

DESCRIPTION

The "newuser" command allows the user to log in as another user without logging out. If a name is specified on the command line, that name becomes the new login name. If no name is specified, "system" is used. If a password exists for the login name specified, "newuser" prompts for the password. The advantage of this command is that when finished as this new user, the user does not need to log in again but simply logs out and returns to the state that existed prior to the execution of the "newuser" command. Note that no options are supported.

Arguments

<user_name>	The name of the user as whom to temporarily log in. Default is "system".

EXAMPLES

```
newuser mary
```

This example temporarily logs in the user as "mary" (assuming any existing password is known).

SEE ALSO

log
login

nice

Lower the priority of the specified command.

SYNTAX

```
nice <command_name>
```

DESCRIPTION

The "nice" command lowers the priority assigned to the specified task by subtracting 5 from the number that would normally be assigned (the lower the number, the lower the priority). For example, a user might send a long compilation, the results of which are not immediately needed, to the operating system with the "nice" command. The system executes more pressing tasks immediately and works on the compilation when nothing else is running.

Arguments

<command_name> The name of the command to execute.

EXAMPLES

1. nice ls
2. nice pascal test.p

The first example lowers the priority of the command "ls".

The second example lowers the priority of the compilation of the pascal program "test.p".

ERROR MESSAGES

No command or a built-in command specified.
The "nice" command expects exactly one argument, and that argument may not be one of the commands, such as "jobs" or "wait", that is a part of the nshell program.

SEE ALSO

nshell

nshell

DESCRIPTION

The nshell program is a command interpreter which is the primary interface between the user and the operating system. The nshell program collects and interprets the user's commands. It executes some commands, known as built-in commands (including "addpath", "chd", "dperm", "hangup", "jobs", "log", "login", "nice", "prompt", "setpath", "time", and "wait") itself. It passes others to the UniFLEX core which, in turn, performs the operations requested.

Generally, the nshell program is used interactively. In this form the command line consists of a command name, which may be followed by arguments and options, as appropriate. All elements of the command line must be separated by either spaces or commas (UniFLEX documentation usually shows spaces). The command may be one of the commands supplied with the operating system, the name of a binary file produced by either the assembler or a compiler, the name of a BASIC compiled file, or the name of a text file (with execute permission turned on) which contains a series of commands to execute. In all cases the nshell program spawns a subshell which executes the specified command. If the command name is the name of a BASIC compiled file, the nshell program spawns a subshell which loads and executes BASIC, which, in turn, executes the specified compiled file.

Search Path

Because most commands reside on disk, the nshell program must locate the command before executing it. By default, the nshell program sequentially searches the following directories: the user's working directory, "<home_dir>/bin", "/bin", and "/usr/bin". If the user is the system manager, the system also searches the directory "/etc" immediately after searching "<home_dir>/bin". (The home directory is the user's login directory, as specified in the password file.)

The list of directories searched by the nshell program is known as the search path. The user may add to this list with the "addpath" command or redefine it with the "setpath" command.

Multiple Commands on a Line

The user may specify more than one command on a command line by separating them with any of several special symbols. Parentheses may be used to group commands on a command line containing more than one command. The nshell program spawns a subshell to run each set of

(continued)

commands in parentheses.

The nshell program sequentially executes commands that are separated by a semicolon, ';'. If a task terminates abnormally, the nshell program stops executing the command line.

If the user follows a command with an ampersand, '&', the nshell program, as usual, spawns a subshell which executes the command. However, in this case the nshell does not wait for the task to complete before returning a prompt. Thus, the user may start another command while the first one is executing. A single nshell program can support a maximum of five of these "background tasks". Each time the user sends a task to the background, the nshell program reports the task ID assigned to that task, preceding it with a 'T', which is not part of the task ID. The user may need the task ID to execute the "wait" or "int" command. The task ID may also be obtained by executing the "jobs" command, which returns the task ID and starting time of all background tasks originated by the user at the current terminal from the nshell program. The ampersand may be used following a single command or separating one task from another on the command line.

Two additional command separators, the conjunction operator ("&&") and the disjunction operator ("||"), are available. These separators make execution of the command following the operator dependent on the outcome of the execution of the command preceding it. A command is "true" if it terminates with a termination status of zero, indicating successful completion, and "false" if it terminates with a nonzero termination status, indicating failure. When two commands are separated by the conjunction operator, the nshell program executes the second one only if it completes the first one successfully (it is "true"). When two commands are separated by the disjunction operator, the nshell program executes the second one only if the first one fails (it is "false"). Normally, the command line is evaluated from left to right; however, parentheses may be used to group commands. Commands in parentheses are treated as a single command. Commands separated by a pipe (see Redirected I/O) are also treated as one command.

The processing of the command separators may be summarized as follows:

&& If the command preceding the conjunction operator succeeds, the nshell program tries to execute the next command. If the command preceding the conjunction operator fails, the nshell program looks for a disjunction operator. If it finds one, it tries to execute the command which follows it. If it does not find one, processing of the command line ceases.

|| If the command preceding the disjunction operator succeeds, the nshell program looks for a semicolon, ';'. If it finds one, it tries to execute the command which follows it. If it does not find one, processing of the

(continued)

command line ceases. If the command preceding the disjunction operator fails, the nshell program tries to execute the next command.

; If the command preceding a semicolon succeeds, the nshell program tries to execute the next command. If the command preceding a semicolon fails, processing of the command line ceases.

& Whether the command preceding a single ampersand succeeds or fails, the nshell program processes the next command on the command line.

Consider the following example:

<task_1> && <task_2> || <task_3> && <task_4>

The nshell program first tries to execute <task_1>. If the task is unsuccessful, the nshell skips <task_2> and proceeds to <task_3>. If <task_3> fails, the nshell program skips <task_4>; if <task_3> succeeds, it tries to execute <task_4>. If, however, <task_1> succeeds, the nshell program tries to execute <task_2>. If <task_2> also succeeds, the nshell program skips the rest of the command line. If, after the successful execution of <task_1>, <task_2> fails, the nshell tries to execute <task_3>. If and only if <task_3> succeeds, it goes on to <task_4>.

The use of parentheses can change the interpretation of the same set of commands separated by the same operators:

<task_1> && (<task_2> || <task_3>) && <task_4>

In this case, the nshell once again begins by trying to execute <task_1>. If it fails, the nshell program skips the remaining tasks. If, on the other hand, <task_1> is successful, the nshell program spawns a subshell (because of the presence of the parentheses). This subshell tries to execute <task_2> and, if and only if it fails, it tries to execute <task_3>. If <task_2> succeeds, it returns a termination status of "true" to its parent nshell. If <task_2> fails but <task_3> succeeds, it also returns a termination status of "true". If, however, both <task_2> and <task_3> fail, the termination status returned is "false". If the termination status returned by the subshell is "true", the parent nshell tries to execute <task_4>.

Termination Status

Normally, the nshell program does not report the termination status of a command it executes unless the task terminates abnormally (because of a program interrupt). A list of the possible program interrupts appears

(continued)

in the documentation of the "int" command. The nshell program does, however, always report the termination status of a background task, even if it terminates normally.

Redirected I/O

The nshell program associates three files with every command it executes: standard input, standard output, and standard error. Standard input is the file from which a command takes its input. Standard output is the file to which a command sends its output. Standard error is the file to which many error messages are directed. By default, the system uses the user's keyboard as standard input and the user's terminal as both standard output and standard error. However, the user can direct the nshell to use another file for any of these standard files. This process is known as I/O redirection.

The symbol '<' tells the nshell program to redirect standard input to the file whose name follows the symbol. Similarly, the symbols '>' and '%' are used to redirect standard output and standard error. The file to which standard input is redirected must already exist. However, if the file to which standard output or standard error is redirected does not exist, the system creates it. In fact, if the file does already exist, the system deletes the contents of the file before executing the command. To avoid this effect, the user may instead direct the nshell program to append data to the file specified as standard error or standard output by duplicating the symbol used for redirection. For example, to execute the "time" command on "ls" redirecting standard output to the file "time_out" and standard error to the file "time_err", the user types

```
time ls >>time_out %%time_err
```

If either of the specified files already exists, its contents remain intact and the relevant output from the command is added to the end of the file.

It is also possible to redirect standard output, standard error, or both to another task. This form of redirection is accomplished by using a "pipe". A pipe is a function that connects programs so that the output from one program becomes the input for another. Standard output is piped from one task to another by using one of the symbols '|' or '^'. For instance, the user can get a listing of all the files in the working directory, format the listing with the "page" command, and print the listing on the printer "spr" with the following command:

```
ls . | page | spr
```

(continued)

Similarly, the user can redirect standard error with either of the symbols "%|" or "%^".

Although the user can place many pipes on the command line, a single task can only support one pipe. Thus, the user cannot pipe standard error and standard output to separate tasks. It is possible, however, to duplicate standard error onto standard output and to redirect them both to the same task. The user has a choice of symbols for duplicating standard error onto standard output: ">%" or "%>". Neither of these symbols takes an argument. After duplicating standard error onto standard output, the user redirects standard output to a file or a task in the usual way. For instance, the user can get a listing of all the files in the working directory, redirect both standard error and standard output to the "page" command, and print the results on the printer "spr" with the following command:

```
ls . >% | page | spr
```

Whenever standard error and standard output are routed to the same destination, their contents may be intermingled.

Finally, the following constructions redirect I/O from or to the null device, "/dev/null": "<-" for standard input, ">-" for standard output, and "%-" for standard error. If either standard output or standard error is redirected to the null device, its contents are lost. If the null device is used as standard input, an end-of-file character is read.

Continuation of the Command Line

Command lines may be continued across more than one physical line by terminating each line, except the last, with a backslash character, '\', immediately followed by a carriage return. By default, the nshell program uses the prompt "+>" to indicate that the line being entered is a continuation of the previous line (the user may change the prompt with the "prompt" command). When the nshell program processes the line, it replaces the backslash and the carriage return with a space. Typing a line-delete character (control-X) only affects the physical line being typed. The user may delete previous lines of a continued command line by typing a keyboard interrupt (control-C), which deletes the entire command line.

Matching Characters

The operating system recognizes several characters, known as matching characters, which allow the user to specify files with similar names without typing each name individually. The special characters are the asterisk, '*'; the question mark, '?'; and a pair of square brackets,

(continued)

"[]". The nshell program matches these special characters to characters in the names of the entries in the specified directory according to the rules described in this section. If the matching character appears in the last component of the file name, the nshell tries to match it to the names of all files in the specified directory (by default, the working directory). If the matching character appears in any other position in the file name, the nshell tries to match it to the names of directories only.

When the nshell program encounters an asterisk in the command line, it matches it to any character or characters, including the null string but not including a leading period. Thus, the command

```
list *.bak ^ spr
```

lists all files in the working directory whose names end in ".bak" and do not begin with a period. The output is printed on the device "spr".

The question mark matches any single character except the null character or a leading period. For example, the command

```
list chapter_?
```

lists all files whose names begin with the string "chapter_" and end with a single character other than the null character. It is possible to use more than one matching character at a time. For instance, in response to the command

```
list *.?
```

the nshell program lists all files in the working directory whose names end with a period followed by a single character (except, of course, those whose names begin with a period).

The use of square brackets allows the user to specify a set of characters to use in the matching process. The set of characters is defined by listing individual characters or by specifying two characters separated by a hyphen. In the former case, the nshell program looks for all file names which use any one of the enclosed characters in the appropriate place. In the latter, the two characters specify a class of characters containing the two characters themselves and any characters which lexically fall between them in the ASCII character set. For example, if the user's working directory contains nine files named "chapter1", "chapter2", "chapter3", and so forth, the following command may be used to list the first three chapters, the fifth chapter, and the last three chapters:

(continued)

list chapter[1-357-9]

If the nshell program cannot fnd a match for any of the arguments containing matching characters, it aborts the command. If it finds a match for at least one argument containing matching characters, it ignores any other arguments containing matching characters for which it cannot find a match.

If the name of a file does actually contain one of the matching characters or either a space or a comma, the user must enclose the name in single or double quotation marks. In such a case the nshell program passes the arguments to the command without performing any character matching.

Nshell Scripts

An nshell script is a file that contains a list of commands for the nshell program. Such a file might consist of a list of commands that are frequently executed in sequence or of a single, lengthy command that is often used. If the user sets execute permissions on such a file, the name of the file can be used as a command.

The user may add to the versatility of an nshell script by using arguments within the script. The arguments are specified within the script as "$1", "$2", "$3", and so forth. The argument "$0" specifies the name of the calling program. These arguments may appear anywhere in a command argument. For example, the following one-line script may be used to format a floppy disk:

```
/etc/formatfd +qnd=/dev/fd$1 +m=FD-$2
```

If this script is stored in an executable file named "f", the command

```
f 0 DD
```

formats the disk in drive 0 as double-sided, double-density.

If an argument being passed to a command actually contains a dollar sign, it must be enclosed in single quotation marks so that the nshell program does not try to perform any substitution. Note that single quotation marks prevent both substitution of arguments and the expansion of matching characters whereas double quotation marks prevent the expansion of matching characters but allow the substitution of arguments.

(continued)

The nshell program supports several commands that are used exclusively with nshell scripts. These commands--"verbose", "exit", "proceed", and "sabort"--are discussed in this section.

When the nshell program executes a script file, it does not normally echo the commands being executed. The "verbose" command causes the nshell program to echo commands from a script file as they are executed. Each line that is echoed is preceded by two hyphens and a space character.

The "verbose" command may be called without arguments or with one argument, which must be one of the strings "on" or "off". If called without an argument, the default is "on". The command may be executed by the login nshell or may be part of an nshell script. The verbose attribute is always passed from a parent nshell program to a child nshell, but never from a child to a parent.

The nshell program permits the user a limited amount of control over the processing of script files. Normally, it sequentially processes commands in a script file until either one of the commands fails or it reaches the end of the file. If one of the commands fails, the nshell program begins to search the remainder of the script file for a line that contains one of the commands "exit" or "proceed". If it encounters one of these commands, the nshell program resumes processing the script after that command. The only difference between the commands "exit" and "proceed" is that during successful execution of a script file the nshell program stops processing the file if it encounters an "exit" command, whereas it ignores a "proceed" command. The search for both these commands takes place before both the substitution of any arguments and the expansion of any matching characters. Thus, the nshell program does not see an "exit" or "proceed" command that is created as the result of either of these processes.

An example of the use of the "proceed" command follows:

```
/etc/mount /dev/fd0 /usr2
/usr2 runjob
echo "Successful execution."
proceed
/etc/unmount /dev/fd0
```

In this example, the nshell program mounts a disk and tries to execute the command "/usr2/runjob" on that disk. If the command succeeds, the nshell program echoes the message, "Successful execution." and proceeds to unmount the disk. If, on the other hand, the command fails, the nshell program skips all commands between the one that failed and the "proceed" command. It resumes execution with the "unmount" command. Thus, if "/usr2/runjob" fails, the user's disk is unmounted, but no message is sent to standard output.

(continued)

This example can be modified to notify the user of either successful or unsuccessful execution by using the "exit" command:

```
/etc/mount /dev/fd0 /usr2
/usr2/runjob
/etc/unmount /dev/fd0
echo "Successful execution."
exit
/etc/unmount /dev/fd0
echo "Unsuccessful execution."
```

In this example, if "/usr2/runjob" succeeds, the nshell program continues execution with the "unmount" command and echoes the string "Successful execution." to standard output. The "exit" command then causes the nshell program to stop processing the script because it encounters the "exit" command during normal execution. If "/usr2/runjob" fails, the nshell program skips all commands until it encounters the "exit" command. It then resumes execution with the "unmount" command and echoes the string "Unsuccessful execution." to standard output.

The user may at times wish to force the execution of every command in an nshell script regardless of the failure of previous commands. The "sabort" command can be used to turn off the search for either an "exit" or "proceed" command, thus forcing execution of every command in the script.

The "sabort" command may be called without arguments or with one argument, which must be one of the strings "on" or "off". When "sabort" is "on", the nshell program looks for an "exit" or "proceed" command whenever a command in the script fails. When "sabort" is off, the nshell program processes every command in the script. If the user executes the "sabort" command without an argument, it both rescinds the effect of any previous "sabort on" and fails. Thus, if it is executing an nshell script, the nshell program immediately begins looking for an "exit" or "proceed" command.

The "sabort" command may be executed by the login nshell or may be part of an nshell script. The attribute is always passed from a parent nshell program to a child nshell, but never from a child to a parent.

System Script Files

If the nshell program is renamed to "shell", replacing the standard operating system shell program, it recognizes and automatically executes several script files. When the system is booted, it executes a single-user shell program, which looks for the file ".begin" in the root

(continued)

directory. If the file does not exist, the shell program immediately sends a prompt to the console. If the file does exist, the shell sends the following message to the console:

Processing of ".begin" starts in 5 seconds.

and waits five seconds before it begins to execute the file. The user can prevent its execution by typing a control-C. If the command "log" or "logout" is included in the file, the system enters multi-user mode. When it enters multi-user mode, it destroys the single-user shell, so if either of these commands is used, it should be the last one in the file.

Whenever the system goes from single- to multi-user mode, it automatically executes the file "/etc/startup" if it exists. This file may, therefore, be used to execute commands which the system manager would otherwise have to execute manually each time the system is booted. Such commands might include the killing of any stray temporary files and the activation of all printer spoolers on the system.

When the system manager executes the command "/etc/shutup", which takes the system from multi- to single-user mode, the shell program looks for the file ".finishup" in the root directory. If the file does not exist, the system goes directly to single-user mode. If the file does exist, the shell sends the following message to the console:

Processing of ".finishup" starts in 5 seconds.

and waits five seconds before it begins to execute the file. The user can prevent its execution by typing a control-C.

In addition to this system script file, the system also supports startup files for individual users. Whenever a user logs in, the shell program looks for a file named ".startup" in the user's home directory (as defined in the password file). If the file exists and the user has read permissions in it, the shell executes the file before issuing the system prompt.

Noninteractive Nshell

The nshell program can also be used as a command in its own right. This form is used primarily to execute an nshell script for which execute permissions are not set, to call the nshell program from another program, or in the password file. The documentation for its use in this way follows.

(continued)

SYNTAX

nshell [+abclnsvx] [<argument_list>]

DESCRIPTION OF THE "NSHELL" COMMAND

If the "nshell" command is executed without any options or arguments, the operating system simply spawns another nshell for the user. This nshell program functions as a normal nshell, but because it is the child of the nshell program from which the command was executed, it does not know what the user's home directory is. The "log" command returns control to the parent nshell.

The "nshell" command can also be executed with options only. This form of the command also spawns an nshell program that interacts with the user. If used in the password file, the command should be executed with the '1' option (see Options Available).

Finally, the "nshell" command can be executed with arguments or with both options and arguments. This form may be used, for example, to execute an nshell script for which the user does not have execute permissions. Either of the following commands executes the file "script":

```
nshell script
nshell <script
```

The nshell program first checks to see that the file specified as an argument is actually a file containing commands. If it is not, the nshell does not execute it unless the user specifies the 'c' option (see Options Available).

Arguments

<argument_list> A list of arguments to pass to the nshell command. Each element in the argument list consists of a command name followed by the appropriate arguments and options. The elements in the list must be separated by a valid command separator (';', '&', "&&", or "||"). If any separator characters are used, the entire argument list must be enclosed in single or double quotation marks.

(continued)

Options Available

Options specified to the nshell program must appear immediately after the name "nshell" on the command line, so that they are not confused with options that pertain to the arguments passed to the nshell.

a	Start execution with the "sabort" attribute off.
b	Ignore control-C and control-\.
c	Process the argument list as a command.
l	Run as a login nshell. A login nshell tries to find the name of the user's home directory by looking in the file ".home?". It also automatically executes the file ".startup" in the working directory.
n	Run all background tasks with lowered priority (as does the "nice" command).
s	Execute the file ".startup" in the working directory.
v	Start execution with the verbose attribute on.
x	Execute the next command without forking unless necessary. This option is only used when calling an nshell program from another program.

NOTES

- It is impossible to specify a null string as an argument to a command because the nshell program removes null strings from the command line.

ERROR MESSAGES

Built-in commands may not use pipes.
Input to or output from the nshell built-in commands may not be routed through a pipe.

Cannot execute "<cmd_name>".
The operating system was unable to execute the specified command. Either the command does not exist or the user does not have execute permission.

Cannot initialize tables.
This error, which should not occur, is usually indicative of a hardware failure. If it does occur, contact the vendor.

Cannot open I/O redirection file.
The operating system returned an error when the nshell program tried to open the file specified for I/O redirection. Most probably, the path specified cannot be followed (one of the directories does not

(continued)

exist) or the user does not have the permissions necessary for opening the file. This message is preceded by an interpretation of the error produced by the operating system.

Cannot open pipe.
The operating system returned an error when the nshell program tried to open the specified pipe. This message is preceded by an interpretation of the error produced by the operating system.

Error opening a file.
The operating system returned an error when the nshell program tried to open the specified file. This message is preceded by an interpretation of the error produced by the operating system.

Error reading a file.
The operating system returned an error when the nshell program tried to read the specified file. This message is preceded by an interpretation of the error produced by the operating system.

Error writing a file.
The operating system returned an error when the nshell program tried to write to the specified file. This message is preceded by an interpretation of the error produced by the operating system.

I/O redirection conflict.
The user tried to redirect standard input, standard output, or standard error to more than one place.

I/O redirection error.
The operating system returned an error when the nshell program tried to perform the specified I/O redirection. This message is preceded by an interpretation of the error produced by the operating system.

Memory overflow.
There is not enough memory available to perform the specified command. Most probably, the expansion of the matching characters used on the command line, for which many matches were possible, caused the error.

Missing "]" or invalid character range.
Either the right-hand square bracket is missing from the specification of a range of matching characters, or the range specified is invalid.

No matching file names found.
Matching characters appear on the command line, but no file names match the specified pattern.

(continued)

Parenthesis usage error.
The parentheses used on the command line are unbalanced.

Too many tasks.
The nshell program tried to execute a fork, but too many tasks were running at the time. The limit to the number of tasks allowed either to the individual user or to the operating system as a whole was reached.

Unknown error.
This error should not occur. If it does, contact the vendor.

Unrecognized argument to builtin command.
The argument specified is not a valid argument to the built-in command in question.

Unterminated string.
The quotation marks used on the command line are unbalanced.

SEE ALSO

addpath
chd
dperm
hangup
jobs
log
login
nice
prompt
setpath
time
wait

nshell

An enhanced shell program.

SYNTAX

nshell [<+abclnvx>] [<argument_list>]

DESCRIPTION

The "nshell" program is an enhanced version of the UniFLEX program "shell". A description of the differences between the standard shell program as shipped with UniFLEX and "nshell" follows.

Features of the Enhanced Shell

1. Redirecting standard error.
   Standard error may be redirected to a file, piped to a program, or duplicated onto standard output. The percent sign, '%', is used as the symbol that redirects standard error. A single percent sign followed by a file name causes the system to create the specified file and to redirect standard error to it. Two percent signs followed by a file name cause the system to append standard error to the specified file.

   Standard error may be piped by concatenating the pipe symbol of choice (either the vertical bar, '|', or the caret, '^') to a single percent sign (the percent sign comes first). This procedure does not affect standard output. It is impossible to pipe standard error and standard output to different tasks. It is, however, possible to pipe both standard output and standard error to the same task by duplicating standard error onto standard output (described in the next paragraph) before piping standard output to the task.

   Standard error may be duplicated onto standard output by specifying either ">%" or "%>". These constructions do not take a file name. To redirect both standard error and standard output to the same file, the user must redirect standard output (using '>' or ">>") in addition to duplicating standard error onto standard output.

(continued)

2. Specification of the null device.
   The following constructions redirect I/O from or to the null device, "/dev/null": "<-" for standard input, ">-" for standard output, and "%-" for standard error.

3. Matching characters in file name specifications.
   Matching characters may appear in any or all components of a path. For example, "/usr?/*" is a valid path name. All components of a path name except the last component are only matched by directory names. For example, the command "echo /*/*" echoes the names of files and directories that are contained in first-level subdirectories of the root directory. It does not echo names of files in the root directory.

4. Script arguments.
   Arguments to "nshell" scripts may appear anywhere in a command argument. For example, the following one-line script may be used to format a floppy diskette:

```
/etc/formatfd +qnd=/dev/fd$1 +m=FD-$2
```

   If this script is stored in an executable file named 'f', the command

```
f 0 DD
```

   formats the diskette in drive 0 as double-sided, double-density.

5. Definition of the command search-path.
   The user may specify which directories "nshell" is to search for commands with the "addpath" and "setpath" commands. The "addpath" command adds its arguments to the list of paths to search. For example, the command

```
addpath /usr/games
```

   adds the path "/usr/games" to the list of paths to search.

   The "setpath" command is used either to display or to specify all paths for "nshell" to search. If called without an argument, "setpath" displays the current paths that are searched in the order in which they are searched. When called with arguments, the arguments replace the current set of search paths. The paths are searched in the order that they are specified as arguments. For example, the command

```
setpath . /bin /usr/games
```

(continued)

tells "nshell" to search the directories '.', "/bin", and "/usr/games", in that order.

The default directories are '.', "<home>/bin", "/bin", and "/usr/bin" where <home> is the user's home (login) directory. If "nshell" is not running as a login shell (see the 'l' option), it does not know what the home directory is and, therefore, cannot search it. The system manager is also given the directory "/etc".

6. Continuation of the command line.
Command lines may be continued across more than one physical line by terminating each line, except the last, with a backslash followed by a carriage return. The prompt "+>" is used to indicate that the line being entered is a continuation of the previous line. When "nshell" processes the line, it replaces the backslash and carriage return with a space. Typing a line-delete character (control-X) only affects the physical line being typed. The user may delete previous lines of a continued command line by typing a keyboard interrupt (control-C), which deletes the entire command line.

7. The "prompt" command.
The "prompt" command has been modified to allow the user to change both the prompt and continuation-prompt strings. If the user specifies only one argument, "nshell" changes only the regular prompt. If the user specifies two arguments, the first argument becomes the regular prompt, and the second argument becomes the continuation prompt. It is impossible to change the continuation prompt without specifying the regular prompt.

The prompt time-indicator (the tilde, '~') may appear anywhere in either or both prompt strings. If multiple tildes appear in a prompt string, only the first one is replaced by the time.

8. Conjunction and disjunction of commands.
Two additional command separators, "&&" and "||", are available for the conjunction and disjunction of commands. A command is "true" if it terminates with a zero termination status, indicating successful completion, and "false" if it terminates with a nonzero termination status, indicating failure. When two commands are separated by a conjunction operator, "nshell" executes the second one only if it completes the first one successfully (it is "true"). When two commands are separated by a disjunction operator, "nshell" executes the second one only if the first one fails (it is "false"). Conjunction has a higher priority than disjunction. Parentheses may be used to alter the priority (commands in parentheses are executed by a subshell).

(continued)

9. The "verbose" command and the 'v' option.
When "nshell" executes a script file, it does not normally echo the commands being executed. The "verbose" command and the 'v' option are used to cause "nshell" to echo commands from a script file as they are executed. Each line is preceded by two hyphens and a space character.

   The "verbose" command may be called without arguments or with one argument, which must be one of the strings "on" or "off". If called with no arguments, it assumes the default argument "on". Once it has executed "verbose on", "nshell" displays the commands from all scripts it executes until it executes the command "verbose off".

   If "nshell" is invoked explicitly with the name of a script file as an argument, verbose mode may be enabled by specifying the 'v' option to "nshell". In this case the 'v' option must appear before the name of the script file. If it appears after the file name, "nshell" interprets it as an argument to the script file. For example, the command

```
nshell +v runjob
```

   instructs "nshell" to process the script file "runjob" and to display the commands as it executes them.

10. The "exit", "proceed", and "sabort" commands.
The "nshell" program permits a limited amount of user control over the processing of script files. The standard shell program stops processing commands from a script file when one command fails (returns a nonzero status). When a command in an "nshell" script fails, "nshell" searches the remainder of the script file for a line that contains either "exit" or "proceed". If it encounters one of these commands, "nshell" resumes processing after that line. The difference between the commands "exit" and "proceed" is that during successful execution of a script file "nshell" stops processing the file if it encounters an "exit" command, whereas it ignores a "proceed" command. An example of the use of "proceed" follows:

```
/etc/mount /dev/fd0 /usr2
/usr2/runjob
echo "Successful execution."
proceed
/etc/unmount /dev/fd0
```

   In this example, "nshell" executes the "unmount" command whether or not "/usr2/runjob" fails. If "/usr2/runjob" does not fail, script processing continues with the "echo" command. If "/usr2/runjob" does fail, "nshell" skips commands until it encounters the

(continued)

"proceed" command. It resumes execution at the following line.

A similar example using the "exit" statement follows:

```
/etc/mount /dev/fd0 /usr2
/usr2/runjob
/etc/unmount /dev/fd0
echo "Successful execution."
exit
/etc/unmount /dev/fd0
echo "Unsuccessful execution."
```

In this example, if "/usr2/runjob" succeeds, execution continues with the "unmount" command, and the "echo" command proclaims successful execution. The "exit" command that follows the "echo" command causes "nshell" to stop processing the script because it has encountered the "exit" command during normal execution. If "/usr2/runjob" fails, "nshell" skips lines until it encounters the "exit" command. It then resumes execution with the "unmount" command, followed by the "echo" command which proclaims unsuccessful execution. The search for the "proceed" and "exit" commands takes place before any argument or file-name substitution takes place. Thus, these commands are not seen by "nshell" if they are created as the result of file-name or argument substitution.

The "sabort" command is used in script files to force execution of all lines in a script file in spite of command failures. Normally, if any command in a script file fails, "nshell" begins searching for a "proceed" or "exit" command. However, if "sabort" is "off", "nshell" acts as if it successfully completes each command.

The "sabort" command may be called without arguments or with one argument, which must be one of the strings "on" or "off". Once it has executed "sabort off", "nshell" processes commands sequentially from the script file even if one or more commands fail. The command "sabort on" rescinds the effect of "sabort off". When executed with no arguments, the "sabort" command rescinds the effect of a previous "sabort on" and also fails. Thus, "nshell" immediately begins searching for a "proceed" or "exit" command.

If "nshell" is invoked explicitly with the name of a script file as an argument, "sabort off" may be set by specifying the ´a´ option. In this case, the ´a´ option must appear before the name of the script file. If it appears after the file name, "nshell" interprets it as an argument to the script file. For example, the command

```
nshell +a runjob
```

instructs "nshell" to process the script file "runjob" with "sabort off". The effect of the ´a´ option may be rescinded inside the

(continued)

script file by an "sabort on" or "sabort".

11. The "wait any" command.
The "wait" command has been modified to accept an argument consisting of the word "any". When the user has more than one task running in the background, the "wait any" command is used to wait for any one task to terminate. The "wait any" command returns control to the user as soon as any background task started by "nshell" terminates, regardless of which task it is.

Options Available

The options to "nshell" must appear immediately after the name "nshell".

a	Start script with "sabort off".
b	Ignore control-C and control-\.
c	Process the next argument as a command.
l	Run as a "login" shell.
n	Run all background tasks with lowered priority.
v	Set "verbose on".
x	Execute next command without forking unless necessary.

NOTES

- The enhanced shell is substantially larger than the standard shell. It should not be used on systems with less than 256K of memory.

- If "nshell" cannot find a match for any of the arguments containing matching characters, it aborts the command. This behavior differs from that of the standard shell program which, in such a case, passes the arguments as typed by the user to the program. If "nshell" finds a match for at least one argument containing matching characters, it ignores any other arguments containing matching characters for which it cannot find a match.

- It is impossible to specify a null string as an argument to a command because "nshell" removes null strings from the command line.

- Whenever "nshell" receives an error from UniFLEX, it attempts to give an English interpretation of that error. These interpretations are read from the file "/gen/errors/system". Early versions of UniFLEX do not include this file. If the file does not exist, "nshell" merely reports the UniFLEX error number.

- The "nshell" program may replace the standard shell. Since shell programs are normally shared text, it is not possible to copy over one if it is currently being used. To replace the standard shell

(continued)

with "nshell", the following command sequence should be executed from single-user mode with the master disk for Utilities Package II in drive 1.

```
/etc/mount /dev/fd1 /usr2      (Mount utility master)
kill /bin/shell                (Kill standard shell)
copy /usr2/nshell /bin/shell   (Copy enhanced shell)
/etc/unmount /dev/fd1 /usr2    (Unmount utility master)
/etc/stop                      (Stop the system)
```

. If "nshell" coexists with the standard shell, individual users may specify "nshell" as their login program in the password file. In these cases, the '1' option should be specified so that "nshell" knows that it should act like a login shell program. A sample password-file entry demonstrating this follows.

```
username:123:encryptedpasswor:/usr/username:nshell +l
```

SEE ALSO

shell

objcmp

Compare two object files and report differences.

SYNTAX

objcmp <file_name_1> [<file_name_2>] [+cq]

DESCRIPTION

The "objcmp" command compares two object files or an object file and a core file by using the information encoded in the binary headers. It first displays differences detected in the binary headers, then differences found in the actual core images of the files. It compares the corresponding text and data segments of relocatable files. Offsets shown with the differences are relative to the beginning of the corresponding segment. Absolute files are compared record by record. Only records with the same load address can be compared, so records that do not match are skipped. When comparing an object file with a core file, "objcmp" relies on the information contained in the header of the object file to find the corresponding sections in the core file. A quiet mode is available which does not show the differences but simply states whether or not the files are the same. If only one file is specified on the command line, "objcmp" looks in the working directory for a file called "core" and uses it as the second file (automatically turning on the ´c´ option).

Arguments

<file_name_1>	The name of an object file.
<file_name_2>	The name of an object file or a core file. If it is a core file, the ´c´ option must be used.

Options Available

c	The second file on the command line is a core file.
q	Use quiet mode. Do not show the differences.

EXAMPLES

1. objcmp runner
2. objcmp tester oldcore +c
3. objcmp runner tester +q

The first example compares the object file "runner" with the file "core" in the working directory and reports any differences.

(continued)

The second example compares the object file "tester" with the file "oldcore". The ´c´ option informs "objcmp" that the second file is actually a core file, not an object file.

The third example compares the two object files "runner" and "tester". Since the ´q´ option is specified, "objcmp" does not show all the differences but simply reports whether or not the files are the same.

NOTES

- Only files of the same type can be compared. For example, absolute files cannot be compared with relocatable files, shared-text files cannot be compared with no-text files, and so forth.
- Only executable files can be compared to core files.
- Executable, shared-text files cannot be compared with core files.

ERROR MESSAGES

usage : objcmp <file_name_1> [<file_name_2>] [+cq]
  This message is issued when no files are present on the command line. At least one file name must be on the command line.

Extra file ´<file_name>´ : ignored.
  If more than two file names appear on the command line, only the first two are used, others are ignored.

Illegal option ´<char>´ : ignored.
  An illegal option was found on the command line and ignored.

´<file_name>´ is not a legal object file!
  The specified file does not have a valid binary header.

´<file_name_1>´ is <type_1> and ´<file_name_2>´ is <type_2> :
Cannot compare.
  The two specified files have incompatible types and cannot be compared.

SEE ALSO
  bcompare

pack

Compress ASCII data and write the compressed data to a file.

SYNTAX

```
pack [<infile_name>] <outfile_name>
```

DESCRIPTION

The "pack" command compresses ASCII data and writes the compressed data to the specified file. The ASCII data may come from a file or from standard input.

The command determines the frequency of each ASCII character in the file being compressed, then transforms the most frequently occurring characters into bit patterns between 2 and 7 bits long. Files of ASCII data are typically compressed between 25% and 45% of their original size.

Arguments

<infile_name>	Name of the file containing the data to compress. Default is standard input.
<outfile_name>	Name of the file to contain the compressed data.

EXAMPLES

1. pack listing packedlisting
2. pack packedlisting <listing
3. list listing | pack packedlisting

All three examples compress the data in the file "listing" and write the compressed data to the file "packedlisting".

The first example explicitly references the input file "listing" and the output file "packedlisting".

The second example uses input redirection to give the "pack" command its ASCII data from the file "listing".

The third example uses the UniFLEX "pipe" feature, in which the output from one command ("list") is used as the input to another command ("pack").

(continued)

NOTES

- If the file specified as the output file already exists and its permissions allow the user to write to it, "pack" replaces its contents with the compressed ASCII data.

- If the file specified as the output file does not exist, "pack" attempts to create it with read and write permissions for the user, read permission for others, and the current user as its owner.

ERROR MESSAGES

usage: pack [<infile_name>] <outfile_name>
The "pack" command requires at least one and no more than two arguments. This message indicates that the argument count is wrong.

Non-ASCII data read from standard input
The data being piped, directed, or otherwise sent to "pack" through standard input contain non-ASCII data. ASCII data are 8-bit byte values between 0 and 127 (decimal) inclusive.

File contains non-ASCII data: <infile_name>
The file containing the data to be packed contains non-ASCII data. ASCII data are 8-bit byte values between 0 and 127 (decimal) inclusive.

Unable to open temporary file
A temporary file needed to save the data read from standard input could not be opened. Another message follows to explain why the file could not be opened.

SEE ALSO

unpack

pwfcheck

Validate the password file.

SYNTAX

pwfcheck [<file_name>] [+nw]

DESCRIPTION

The "pwfcheck" command checks the entries in a password file for defects. The possible defects are divided into three classes: (1) errors, (2) warnings, and (3) notes. An "error" is a serious defect which the system manager should immediately correct. These "errors" may cause the "login" program to behave unpredictably. A "warning" is a defect which has a high probability of preventing a given user from logging in. A "note" is an inconsistency that is probably the result of an oversight by the system manager when adding the new user to the password file.

Arguments

<file_name>	The name of the password file. Defaults to "/etc/log/password".

Options Available

n	Suppress "note" messages.
w	Suppress both "warning" and "note" messages.

EXAMPLES

1. pwfcheck
2. pwfcheck /usr2/etc/log/password +n

The first example validates of the password file currently used by the system. No messages are suppressed.

The second example validates the file "/usr2/etc/log/password". The 'n' option instructs "pwfcheck" to suppress "note" messages.

NOTES

- The "pwfcheck" command requires system-manager permissions in order to perform all checks. If it does not have these permissions, it issues a message to that effect and runs as many checks as it can. The checks that cannot be run without system-manager permissions

(continued)

pertain to checking the accessibility of the login directory and the login program.

- The checks for execute permission on the login program and the login directory are not complete in that not all components of the path necessary to reach them are examined for the proper permissions.

- If an error exists in the field containing the user ID of an entry in the password file, the checks for directory and file permissions are suppressed for that entry.

ERROR MESSAGES

The program precedes error messages that refer to defects in an entry in the password file by a display of the defective entry. A defective entry is displayed only once. If several errors relate to that entry, "pwfcheck" groups them under the display of the entry.

Not running with system-manager permissions.
Not all checks can be performed.
The "pwfcheck" command requires system-manager permissions to check the accessibility of both the login directory and the login program.

Not enough memory to check for duplicates.
Checking for duplicate names and user IDs requires that the entries in the password file be sorted. If the password file is very large (several hundred entries), the sort cannot be performed. In this case, "pwfcheck" does not check for duplicate entries.

File "<file_name>" is a device or a directory.
The file name passed as an argument specified a device or a directory.

File "<file_name>" cannot be opened.
The operating system returned an error when "pwfcheck" tried to open the specified file. This message is preceded by an interpretation of the error returned by the operating system.

File "<file_name>" cannot be located.
The operating system returned an status when "pwfcheck" tried to locate the specified file. This message is preceded by an interpretation of the error returned by the operating system.

Note: The following names are duplicated
This message precedes a list of those names that appear in more than one entry in the password file.

(continued)

Note: The following user IDs are duplicated
This message precedes a list of those user IDs that appear in more than one entry in the password file.

Error: Entry longer than 128 characters
An entry in the password file may not exceed 128 characters, including the carriage return. Only the first 128 characters of the entry are displayed.

Error: There are not exactly four colons
A password file entry must contain exactly four colons.

Error: Field for the user name is empty
No information is in the field for the user name. This field precedes the first colon.

Warning: User name is longer than 8 characters
A user name may not contain more than eight characters.

Warning: User name contains uppercase or separator characters
A user name may not contain uppercase or separator characters.

Warning: User name starts with a digit
A user name may not start with a digit.

Note: User name contains digits
A user name that contains any digits cannot be used from a terminal that does not have lowercase capability. If such terminals are on the system, user names should not contain digits.

Warning: Password field is not 16 characters long
The encrypted form of the password should always be sixteen characters long.

Warning: Password field is not all lowercase characters
The encrypted form of the password should contain only lowercase characters.

Error: Field for the user ID is empty
No information is in the field for the user ID. This field is between the second and third colons.

Warning: User ID is larger than 32767
User IDs must be between 0 and 32767 inclusive.

Warning: Field for the user ID is not all digits
The user ID is a number, so the field must contain only digits.

(continued)

Error: Field for the login directory is empty
No login directory was specified for this entry. The field for the login directory is between the third and fourth colons.

relinfo

Display information about an object file.

SYNTAX

relinfo <file_name> [<file_name_list>] [+ehrs]

DESCRIPTION

The "relinfo" command displays information about the binary header, the symbol table, and both the relocation and external records in either an object file or all modules of a library. Normally, "relinfo" displays all the information. The available options restrict the display to the specified information (see Options Available).

Options Available

e	Display only information about external records.
h	Display only information about the binary header.
r	Display only information about relocation records.
s	Display only information about the global symbol table.

EXAMPLES

1. relinfo tester
2. relinfo /lib/mathlib +h
3. relinfo reporter +se

The first example displays information about the binary header, the symbol table, and both the relocation and external records in the object file "tester" in the working directory.

The second example displays the information about the binary headers from all the modules in the library "/lib/mathlib".

The third example displays the information about both the relocation and external records in the file "reporter" in the working directory.

ERROR MESSAGES

Unknown option '<char>' ignored.
An unknown option was found and ignored.

Error opening '<file_name>' : <reason>
The operating system returned an error when "relinfo" tried to open the specified file.

(continued)

Error reading ´<file_name>´ : <reason>
The operating system returned an error when "relinfo" tried to read the specified file.

Error seeking to <location> in ´<file_name>´ : <reason>
The operating system returned an error when "relinfo" tried to seek to the specified location (in hexadecimal) in the specified file.

´<file_name>´ is not a binary file!
The specified file does not have a valid binary header.

SEE ALSO
asmb
lib-gen
libinfo
link-edit
relasmb

strings

Write any ASCII strings contained in a file or files to standard output.

SYNTAX

```
strings [<file_name_list>]
```

DESCRIPTION

The "strings" command finds and sends to standard output the ASCII strings in a file or list of files. An ASCII string is defined as any sequence of four or more printable characters terminated by a null character or a carriage return. This command is useful for determining information about unknown binary files. The user may specify any number of file names. If no file names are specified, "strings" reads standard input. No options are supported.

Arguments

<file_name_list>	A list of the names of files to process.

EXAMPLES

```
strings /bin/dir test
```

This example first finds and sends to standard output the strings in the file "/bin/dir", then those in the file "test" in the working directory.

NOTES

- For efficiency, the algorithm used is a simple one. It may not find every string, and some strings it does find may be nonsense.

ERROR MESSAGES

Can't open `<file_name>'.
The specified file could not be found or could not be opened for reading.

swapstats

Report the current statistics about system swapping.

SYNTAX

```
swapstats
```

DESCRIPTION

The "swapstats" command writes to standard output a report on the current status of swap activity on the system. The statistics reported include the number of swap operations since boot time, the amount of swap space currently occupied (in blocks and as a percentage of total swap space), the percentage of time the system has spent swapping tasks, and the degree of memory utilization. No options are supported.

EXAMPLES

```
swapstats
```

This example is the only valid form of the "swapstats" command. It reports the current status of swap activity on the system.

NOTES

- In some cases "swapstats" comments on the extent of memory utilization. These comments are simply suggestions.

ERROR MESSAGES

Syntax error - no arguments expected.
The user specified an argument or option to the "swapstats" utility, which accepts neither.

Can't open system memory.
The system-memory file, "/dev/smem", cannot be opened.

trail

List a file as it grows.

SYNTAX

trail <file_name> [+flns]

DESCRIPTION

The "trail" command lists a file but does not terminate when it reaches the end of the file. Instead, "trail" sleeps for a specified interval, after which the listing process is resumed if the file has grown. "Trail" might be used, for example, to monitor the output of another program which is slowly writing data to a disk file.

Arguments

<file_name>	Name of the file to list.

Options Available

f	Wait for the file to appear.
l=<time_limit>	Specify the time limit in seconds.
n	List new material only.
s=<interval>	Specify the sleep interval in seconds.

Complete descriptions of the options follow.

The 'f' Option.

The 'f' option instructs "trail" to wait for the specified file to appear if it does not already exist. While waiting, "trail" uses the same sleep interval that it uses when waiting for new material to appear in the file and is subject to either the specified or the default time limit.

The 'l' Option.

The 'l' option specifies a time limit, in seconds, that is imposed on "trail" while it is waiting for either a file or new material to appear. The syntax for the 'l' option is

+l=<time_limit>

where <time_limit> is an integer between 0 and 32767 inclusive. If the

(continued)

file or new material does not appear before the specified time limit is exceeded, "trail" terminates. If this option is omitted, "trail" assumes a time limit of 600 seconds. Specifying a time limit of 0 seconds disables the time limit. In this case the user must terminate "trail" by typing a keyboard interrupt (control-C).

The 'n' Option.

The 'n' option instructs "trail" to skip over any material already in the file and to immediately begin waiting for additional material to appear.

The 's' Option.

The 's' option specifies the time interval, in seconds, for which "trail" is to sleep before checking to see if the file itself or new material in the file has appeared. The syntax for the 's' option is

+s=<interval>

where <interval> is an integer between 0 and 32767 inclusive. If this option is omitted, "trail" uses the default interval of fifteen seconds. If the specified time interval is 0, "trail" also uses the default interval.

EXAMPLES

1. trail output
2. trail listing +n1=60
3. trail out_file +fs=30

In the first example, "trail" lists the contents of the file "output", then checks at 15-second intervals to see if new information has appeared in that file. When "trail" detects new information, it lists this information. If no new information appears during an interval of 600 seconds, "trail" terminates.

In the second example, the 'n' option instructs "trail" to skip over any information that is already in the file "listing". "Trail" then waits for additional material to appear in the file. If no new information appears in the file during a period of sixty seconds, as specified by the 'l' option, "trail" terminates.

In the third example, the 'f' option instructs "trail" to wait for the file "out_file" to appear if it does not already exist. While waiting for the file, as well as while waiting for new material to appear in the file once it exists, "trail" uses a sleep interval of thirty seconds, as specified by the 's' option.

(continued)

NOTES

- If the time limit specified by the 'l' option is less than the sleep interval specified by the 's' option, "trail" terminates if it does not find the file or any new material in the file after one sleep interval.

- Material in a file is considered "new" only if it is appended to the file. "Trail" does not consider the alteration of existing data in the file as the addition of new material.

ERROR MESSAGES

File "<file_name>" not found.
The file named <file_name> was not located, and the 'f' option was not specified.

File "<file_name>" is a device or a directory.
The file name used specified a device or a directory, not a data file.

Cannot open "<file_name>".
The file named <file_name> was located, but "trail" could not open it for reading. This message is preceded by an interpretation of the error returned by the operating system when "trail" attempted to open the file.

Time limit is negative.
The time limit specified by the 'l' option was a negative number.

Sleep time is negative.
The sleep interval specified by the 's' option was a negative number.

Unknown option letter "<char>" ignored.
The option specified by <char> is not a valid option to "trail". This error is not fatal.

SEE ALSO

list

unique

Write sequentially unique lines.

SYNTAX

unique [<file_name_list>] [+d]

DESCRIPTION

The "unique" command reads ASCII data and writes sequentially unique lines to standard output. A sequentially unique line is one which is not identical to the line that immediately precedes it. If the list of file names is omitted, "unique" reads the ASCII data from standard input. Otherwise, it reads the ASCII data from the files in <file_name_list>.

If the lines of data are sorted, all duplicates of any given line appear sequentially. Thus, when "unique" operates on a sorted file, it lists one copy of each unique line.

Standard input may be referenced in the list of file names by using the plus-sign '+' as an argument in the list.

Arguments

<file_name_list>	The list of files to read. A '+' indicates standard input.

Options Available

d	Print the first of each sequentially duplicated line instead of each unique line.

EXAMPLES

1. unique sortedfile
2. list sortedfile | unique +d
3. unique sortedfile1 + sortedfile2 >uniquelines

The first example writes the sequentially unique lines from the file "sortedfile" to standard output.

The second example writes only the first copy of sequentially duplicated lines read from standard input to standard output. The data are piped to "unique" from the "list" command.

(continued)

The third example writes to standard output the sequentially unique lines from the file "sortedfile1", followed by those from standard input, followed by those from the file "sortedfile2". The shell program redirects standard output to the file "uniquelines".

NOTES

- The standard input specifier '+' should not be used more than once in <file_name_list> because the "unique" command cannot reopen standard input once it has closed it.
- The first line of a file is always a sequentially unique line, even if it is identical to the last line of the file immediately preceding it in the list of file names.

ERROR MESSAGES

Unknown option: <char>
The letter <char> is not a valid option to the "unique" command.

SEE ALSO

list

unpack

Expand packed ASCII data.

SYNTAX

unpack <infile_name> [<outfile_name>]

DESCRIPTION

The "unpack" command expands packed ASCII data generated by the "pack" command. It reads the packed ASCII data from the file <infile_name> and writes the expanded version to either <outfile_name> or standard output.

Arguments

<infile_name>	Data file containing packed ASCII data.
<outfile_name>	File to contain the expanded data. Default is standard output.

EXAMPLES

1. unpack packedlisting listing
2. unpack packedlisting >listing
3. unpack packedlisting | tee listing

All three examples do the same thing. First of all, "unpack" examines the file "packedlisting" to determine whether or not it contains packed ASCII data. If it does, "unpack" expands the data and writes them to the file "listing".

The first example explicitly references both an input and an output file . It instructs "unpack" to write the expanded data to the file "listing".

The second example instructs "unpack" to write the expanded data to standard output, which the shell program redirects to the file "listing".

The third example instructs "unpack" to write the expanded data to standard output, which the shell program pipes to the command "tee". (The "tee" command writes whatever it reads from standard input to both the specified file and to standard output.)

(continued)

NOTES

- If the file specified as the output file already exists and its permissions allow the user to write to it, "unpack" replaces its contents with the expanded ASCII data.

- If the file specified as the output file does not exist, "unpack" attempts to create it with read and write permissions for the user, read permission for others, and the current user as its owner.

ERROR MESSAGES

usage: unpack <infile_name> [<outfile_name>]
The "unpack" command requires at least one argument and no more than two arguments. This message indicates that the argument count is wrong.

File does not contain packed ASCII data: <infile_name>
The file <infile_name> does not contain ASCII data compressed by the "pack" command.

File has wrong version number: <infile_name>
The file <infile_name> contains ASCII data compressed by the "pack" command, but the format is obsolete and cannot be expanded. (Note that the version number of the data file containing packed ASCII data changes only when the format for packing ASCII data changes, not necessarily when the version number of the "pack" or "unpack" command changes.)

SEE ALSO

pack