III. Standard File Formats

TABLE OF CONTENTS

NAME

Files - special file formats

FUNCTION

Files under Idris can generally be thought of simply as ordered sequences of eight-bit bytes (characters), written on disk or tape. Since most programs access files just once, front to back, most data streams, such as are produced by terminals or interprogram pipelines, can be processed like ordinary files. There are, however, some special cases.

The manual pages in this section describe files whose formats are more restricted, typically because two or more programs must communicate via a common file and hence must impose additional structure on the information it contains. There are also a few "pseudo files", implemented by character special device handlers in the resident, that permit access to internal system information or provide useful services.

Additional file formats, of interest primarily to programmers, may be found in Section III of the Idris Programmers' Manual.

NAME
ASCII - standard character codes

FUNCTION
ASCII stands for American Standard Code for Information Interchange. It defines 128 character codes for printing graphics, controlling terminals, and formatting messages between text processing machines. All Idris utilities and the resident presume that text is encoded in ASCII, a safe bet because it is used universally -- except by IBM, the world's largest manufacturer of computing equipment, and a few other companies.

At any rate, the octal codes and their meanings are:

CODE	MEANS	CODE	MEANS	CODE	MEANS	CODE	MEANS
0000	NUL	0040	space	0100	@	0140	`
0001	SOH	0041	!	0101	A	0141	a
0002	STX	0042	"	0102	B	0142	b
0003	ETX	0043	#	0103	C	0143	c
0004	EOT	0044	$	0104	D	0144	d
0005	ENQ	0045	%	0105	E	0145	e
0006	ACK	0046	&	0106	F	0146	f
0007	BEL	0047	'	0107	G	0147	g
0010	BS	0050	(	0110	H	0150	h
0011	HT	0051	)	0111	I	0151	i
0012	NL	0052	*	0112	J	0152	j
0013	VT	0053	+	0113	K	0153	k
0014	NP	0054	,	0114	L	0154	l
0015	CR	0055	-	0115	M	0155	m
0016	SO	0056	.	0116	N	0156	n
0017	SI	0057	/	0117	O	0157	o
0020	DLE	0060	0	0120	P	0160	p
0021	DC1	0061	1	0121	Q	0161	q
0022	DC2	0062	2	0122	R	0162	r
0023	DC3	0063	3	0123	S	0163	s
0024	DC4	0064	4	0124	T	0164	t
0025	NAK	0065	5	0125	U	0165	u
0026	SYN	0066	6	0126	V	0166	v
0027	ETB	0067	7	0127	W	0167	w
0030	CAN	0070	8	0130	X	0170	x
0031	EM	0071	9	0131	Y	0171	y
0032	SUB	0072	:	0132	Z	0172	z
0033	ESC	0073	;	0133	[	0173	{
0034	FS	0074	<	0134	\	0174	|
0035	GS	0075	=	0135	]	0175	}
0036	RS	0076	>	0136	^	0176	~
0037	US	0077	?	0137	_	0177	DEL

Multiletter symbols are the accepted abbreviations for nonprinting characters -- their meanings (and usages) are often shrouded in antiquity.

SEE ALSO
print, text

NAME

block - block special files

FUNCTION

A block special file is a direct connection to a peripheral device capable of supporting direct (random access) reads and writes of 512-byte blocks. As many as 65,536 blocks can be accessed via a block special file, but any given device may support fewer, so long as those blocks actually present form a contiguous sequence starting with block zero. Devices with more than 65,536 blocks, and even many smaller ones, are conventionally partitioned into a number of smaller logical devices, so that all areas can be reached and each subarea can be administered separately.

Idris requires all filesystems to be written on block special devices, one to a logical device, if they are to be mounted. Conventional tape can often be treated as a block special device, so long as writes occur only sequentially and reads are never attempted past the highest numbered block written. Thus, a tape filesystem may be mounted read-only.

The Idris resident automatically provides for buffering of accesses to all block special files. This includes "write behind", or deferring writes as long as possible to combine multiple partial block writes, and "read ahead", or anticipating reads on the basis of a recent history of sequential block reads. Performance is enhanced, often dramatically, by this service, but at the risk of greater disk inconsistency on an unexpected shutdown. Thus, mechanisms are available, to the System Administrator, to cause an automatic flushing of the buffer cache at regular intervals, and to the System Generator, to defeat read ahead and/or write behind.

When read as a conventional file, a block special file will always deliver up a multiple of 512 bytes, and may report a read error at the first block beyond its defined areas, rather than a tidier end-of-file.

SEE ALSO

character

NAME

character - character special files

FUNCTION

A character special file is a direct connection to a peripheral device, about which little else can be said. One special category of character special files, the tty files, have a number of useful properties; all other files of this ilk are about as unstructured as anything supported by Idris. Typical character special files are:

raw tape - It is often desirable to write blocks larger than 512 bytes for the sake of storage economy, and it is often necessary to write blocks even smaller for the sake of portability to other systems. A raw tape interface will write records whose size is determined by the individual write requests, and will read records no larger than individual read requests. The standard utility dd can produce tailored requests, for use with raw tape devices, so that alien tapes can be administered by Idris.

raw disk - Raw disk typically permits the transfer of multiple 512-byte blocks with one read or write request, and hence can speed bulk copying. On devices with multiple sectors per block, such as diskettes, raw disk may also permit accesses to the individual sector level.

funny stuff - Printers, paper tape readers, realtime interfaces, etc. often have peculiar control requirements that don't fit completely within the confines of conventional Idris input/output. For these, the stty machinery can be used in nonstandard ways to provide the extra control needed, at the cost of requiring a special access program for each such device.

Block special devices typically provide an additional character special interface, if only to support multiple block reads and writes. Care should be taken when accessing the same physical disk area with two different logical devices, however, since the resident block buffering may be outsmarted.

Character special files may impose arbitrary constraints on read or write requests, such as: records must be an even number of bytes, or records must begin on a special storage boundary within the process, or offsets (seek pointers) within the file must be some multiple of bytes, or offsets may even be ignored. Thus, these devices are a major exception to the general rule that files are highly interchangeable among Idris utilities.

Avoid them.

SEE ALSO

block, dd(II), stty(II), tty

NAME

directory - directory files

FUNCTION

A directory is a file of named pointers to other files. The names are called "links" and the pointers are "inode numbers". Since directory entries can point at other directories, and since the "root" (inode 1) of nearly every filesystem is a directory, all the files in a filesystem are organized into a tree of arbitrary depth. To reach a given file from the root, it is necessary to traverse a path of links; hence a filename is sometimes called a "pathname". A directory consists of 16-byte records in the format:

bytes 0-1: the inode number, an unsigned short integer written less significant byte first. Zero indicates an erased link.

bytes 2-15: the link name, up to 14 characters left justified and NUL padded. A link may contain any character except a NUL or ´/´, since the former is taken as the end of a pathname and the latter as a separator between links in a pathname.

Although not required by the Idris resident, a number of constraints are imposed on the construction of links, to simplify the operation of numerous programs. First, each directory is created with two links, one called "." which points at the directory itself, and one called ".." which points at its parent. A parent directory is the one next closer to the root, except for the root itself, which is its own parent. Second, multiple parents are disallowed for a directory, to avoid loops in the tree structure and to make unambiguous what is meant by "next closer to the root". And third, a directory is never removed (the link from its parent erased) unless the directory contains only the links "." and "..", thus preventing the formation of orphan files.

Writes to a directory are disallowed, even to privileged programs, so that the integrity of the filesystem can be preserved; write permission for a directory merely grants the right to alter links. Similarly, execute permission means that the directory may be scanned while tracing a pathname, never that it can be executed. Reads work just as for plain files.

Note that directories are also like plain files in being unable to shrink. An erased link is left in place, for possible later reuse, but no attempt is made to shorten a directory by squeezing out erased links, or even by lopping off erased links at the end of the file. This can lead to noticeable performance degradation, as when hundreds of files are created in a directory, then later removed, leaving many erased links to skip over when scanning pathnames. The only fix is to create a new directory with the same parent and with new links to the remaining children, to remove the old, and to rename the new as the old.

It is also worth recording, but unlikely to matter, that the resident will not scan beyond the first 4,096 records in a directory.

SEE ALSO

filesystem, plain

NAME

dlog - incremental dump history

SYNOPSIS

/adm/dump

FUNCTION

/adm/dump gives a history of all incremental dumps performed. It consists of 24-byte records, whose format is:

bytes 0-19: the name of the filesystem device, left justified and NUL terminated.

bytes 20-23: the date of the dump.

A record is written to this file for each incremental dump performed.

SEE ALSO

dump

BUGS

The filesystem device should be a) a 64-byte pathname, or b) a 14-byte link in /dev, or c) the device major/minor number. Only the latest entry for a given device should be saved.

NAME

dump - dump file format

Dump files are produced by the dump command, for later use by the restor command, as a means of backing up the logical contents of a filesystem. (Both commands are described in Section IV of this manual.) They can be used to record the entire contents of a filesystem, or only those portions that have changed since an earlier dump date, in a space that is often much less than the filesystem proper. Since dump files are compatible across all implementations of Idris, and with UNIX/V6 dump tapes, they can be used as a means of communicating large numbers of files between UNIX and Idris systems.

A dump file consists of a dump header, a file map, and file header/data pairs. In the following description, unless otherwise stated, two byte quantities are unsigned short integers written less significant byte first; four-byte quantities are filesystem dates. The dump header consists of:

bytes 0-1: the number of (512-byte) blocks of inodes, 16 inodes to the block. Taken from the superblock of the dumped filesystem.

bytes 2-3: the number of blocks reserved for the filesystem, also from the superblock.

bytes 4-7: the date of the dump.

bytes 8-11: the date of last full dump. Incremental dumps consist of all files modified between this date and the date of the dump. Both dates are the same for a full dump.

bytes 12-13: the size in blocks of the dump file.

The file map consists of enough whole blocks to contain one unsigned short per inode in the dumped filesystem. The first entry corresponds to inode 1, etc. The value is interpreted as follows:

-1: file does not exist.

0: file exists, but not dumped. This is used for incremental dumps when the file is older than the date of the last dump.

otherwise: file exists, was dumped, and was one block shorter than the number given.

The file header/data blocks consist of a one block header containing a duplicate of the inode, followed by as many full blocks as necessary to store the actual file contents. Note that if the size field in the inode copy is different from the size as specified by the file map, the file was dumped with a "phase error". This may confuse a restor under UNIX, but is ignored by the Idris restor utility.

The dump header block and the file header blocks have the dump file block address in bytes 508 and 509, and a checksum in bytes 510 and 511. The block checksums to the octal value 031415, when summed as an array of shorts, each written less significant byte first.

SEE ALSO

dlog, filesystem, tp

NAME

filesystem - Idris filesystem structure

FUNCTION

A filesystem is a data structure which, when stored on a block special device, can be administered by the Idris resident as a collection of directories and plain files. Idris requires at least one filesystem, called the root, to be available at system startup and ever after; additional filesystems may be "mounted", or spliced into the existing directory structure, and "unmounted" at will by the system administrator. Since filesystems are compatible across all implementations of Idris and with UNIX/V6 filesystems, they can be used as a means of sharing large numbers of files between UNIX and Idris systems.

A filesystem is organized as a series of 512-byte blocks. Block zero is unused and may serve as a bootstrap, for some machines, that can load stand alone programs from the filesystem. Block one is the "superblock", immediately followed by a variable number of blocks of "inodes", which is followed in turn by a variable number of data blocks. The superblock specifies these variable numbers (which cannot change during the life of a filesystem) and provides handy lists of available inodes and data blocks. An inode holds all the attributes of a file except its name(s); and a data block holds actual file contents, or an array of pointers used to locate other data blocks.

In the descriptions that follow, all one-byte numbers are unsigned char numbers, i.e., in the range [0, 256). All two-byte numbers are unsigned short integers written less significant byte first; such numbers are in the range [0, 65,536). All dates are four-byte unsigned long integers written as two such shorts, more significant short first [honest]; a date is the time in seconds since midnight 1 January 1970 GMT, which will suffice through the 21st Century.

The format of the superblock is:

bytes 0-1: the number of blocks reserved for inodes.

bytes 2-3: the total number of contiguous blocks used by the filesystem.

bytes 4-5: the number of free data/pointer blocks represented in the superblock, in the range [0, 100]. Zero means entire free list is exhausted, one means no more free blocks in the superblock array, two means one free block is present at the beginning of the superblock array, etc.

bytes 6-7: the block number of the first entry in the remaining free list, if this and the preceding number are nonzero.

bytes 8-205: the superblock array of free data/pointer blocks, as left justified short integers.

bytes 206-207: the number of free inodes represented in the superblock, in the range [0, 100].

bytes 208-407: the superblock array of free inodes, as left justified short integers. Free inodes not represented in this array must be located by scanning the actual inode array.

bytes 408-411: unused by Idris, reserved for UNIX administration.

bytes 412-415: date of last superblock modification. Used primarily to guess the date on system startup.

bytes 416-511: unused.

The remaining free list consists of zero or more entries, each written in a free data block. A free list block has the format:

bytes 0-1: the number of free data/pointer blocks represented in this entry, in the range [0,100]. Same convention as in superblock.

bytes 2-3: the block number of the next entry in the free list, if this and the preceding number are nonzero.

bytes 8-205: the array of free data/pointer blocks, as left justified short integers.

Inodes are 32-byte records, hence packed sixteen to the block, commencing with block 2. There must be at least one inode block; more than 4095 are unusable. Inodes are thus numbered from 1 to 65,520. Inode 1 is called the "root" of the filesystem and must be a directory, if the filesystem is to contain more than one file.

The format of an inode is:

bytes 0-1: the attribute or mode flags. If the octal 0100000 (most significant) bit is set, the inode is allocated; otherwise its remaining contents should be ignored. If the next two bits, masked by octal 060000, are: 000000, the entry is a plain file; 020000, the entry is a character special file; 040000, the entry is a directory; 060000, the entry is a block special file. If the octal 010000 bit is set, the file is "large", as described below; otherwise it is "small". The low twelve bits, masked by octal 07777, specify the access permissions for the file, which are described at length in conjunction with the chmod command in Section II of this manual.

byte 2: the number of "links", or directory entries within the filesystem, that should be pointing at this inode.

byte 3: the userid of the owner of the file.

byte 4: the groupid of the owner of the file.

byte 5: the size of the file in multiples of 65,536 bytes.

bytes 6-7: the size of the file in bytes, to be added to the preceding field to obtain the total size.

bytes 8-23: an array of eight short integers, usually used to hold block numbers as described below. For a character special or block special file, however, the first of these is taken as the major device number (times 256) plus the minor device number of the physical device to access.

bytes 24-27: date of last access to this inode. Not updated when accessing a directory in the process of scanning a pathname.

bytes 28-31: date of last modification to this inode.

For a small file, which is not a character special or block special file, the actual file contents are in the blocks pointed at by the eight block numbers. A block number of zero means no data block is present; it is permissible for a file to have unallocated holes, but there should be no blocks specified for bytes beyond the end of the file. Bytes 0-511 of the file are in block zero, bytes 512-1023 in block one, etc. A file over 4,096 bytes long must be large.

For a large file, each of the first seven block numbers points at a block of 256 block pointers, which point in turn to data blocks. The last block number, if nonzero, points at a block of pointers to blocks of pointers to data blocks.

Thus, a filesystem may consist of up to 65,536 blocks, containing up to 65,520 files. Each file may be up to 16,777,215 bytes long. These limits can usually be obscured, however, by mounting multiple filesystems as one directory tree and by accessing files through appropriate utilities.

SEE ALSO

block, character, directory, dump, plain

NAME

log - login history file

SYNOPSIS

/adm/log

FUNCTION

/adm/log is written by the /odd/log program to inform the who command about who has logged in and out, and about major accounting milestones such as date changes and system startups. It contains one 26-byte record for each accounting event, written in order of increasing time.

The format of each record is:

bytes 0-13: the pathname, in the /dev directory, of the tty file on which the login occurred. Left-justified and NUL padded.

bytes 14-21: the loginid of the active user, or "old:time" for the time just before a date change, or "new:time" for the time just after a date change, or "reboot:" for an initializing call to login (presumably at startup time), or "" for an inactive port. Left justified and NUL padded.

bytes 22-25: the time of the entry, as a filesystem date. Binary long integer in native byte order.

Note that this is not a text file and hence is not easily displayed except by the who command.

SEE ALSO

filesystem, tty, who

BUGS

Date should be in filesystem byte order, for portability.

NAME

mount - mounted filesystems

SYNOPSIS

/adm/mount

FUNCTION

/adm/mount is a file, administered by the /etc/mount command described in Section IV of this manual, used to record the filesystems currently mounted. Other programs use this file to trace pathnames or to inspect all mounted filesystems.

The file consists of 66-byte records, one for each filesystem, with the format:

bytes 0-1: the device minor and major numbers, of the mounted filesystem, in native byte order. The major number is multiplied by 256 and added to the minor number.

byte 2: the status flag. A null ´\0´ indicates no longer mounted, \&´m´ means mounted read/write, and ´r´ means mounted read only.

bytes 3-65: the pathname of the node mounted upon, left justified and NUL padded.

Since /adm/mtab is not a text file, it is best inspected with the help of the /etc/mount command.

NAME

null - bottomless pit

SYNOPSIS

/dev/null

FUNCTION

/dev/null is a character special file that reports end of file on any read and that trashes anything written to it, without complaint. It is used by the shell as a safe place to redirect standard input (STDIN) for a background process that has not otherwise redirected its input. It is also a convenient place to send large quantities of output when all that is desired is some side effect of its production.

NAME

passwd - the system password file

SYNOPSIS

/adm/passwd

FUNCTION

/adm/passwd contains login information about each possible user; hence it is used extensively for checking and for mapping userid and groupid numbers to human readable loginids. It is a text file, with one line per user, where a line consists of colon separated fields with the following meaning:

FIELD	CONTENTS
0	loginid
1	encrypted password
2	userid
3	groupid
4	long form user name
5	login directory
6	login command

loginid is the name that the user types in response to the prompt from the login program. It should be one to eight characters, preferably printable, and distinct from all other loginids in /adm/passwd.

The encrypted password is an empty string, if the user has no password, or a twelve-character sequence generated by the passwd command. Placing anything else in this field is a sure way to prohibit logins with that userid -- a practice often followed with group name entries. Note that there is (intentionally) no obvious relation between the encrypted password and the secret password typed by a user, which is one to eight ASCII characters. Thus /adm/passwd may be safely read by all.

userid is a number, between 0 and 255, assigned by the System Administrator and stored as part of the attributes of a file, used to encode ownership and identity in just one byte of information. userid zero is reserved for the "superuser", a mythical being who is frequently granted special dispensation by the resident, including the power to cloud men's minds. Each userid should be unique within /adm/passwd, so that the ownership of files can be correctly reported and enforced. It is never necessary to know a userid outside this context, since all utilities refer to /adm/passwd to map userids to loginids.

groupid is a number just like userid, except that it designates the group membership corresponding to this loginid. The space of groupids is distinct from userids, so there can be up to 256 working groups who can share access to common files.

The long form user name is essentially unused, except to the extent that it serves to identify a loginid more fully.

The login directory is made the user´s current directory on a successful login, just before the login command is executed.

The login command is the end result of the login process. For normal users it is some form of command interpreter (shell), usually "/bin/sh". The command may consist of quoted strings (single or double quotes), redirection of STDIN and STDOUT with ´<´ and ´>´ respectively, and other whitespace separated arguments. After processing redirection and eliminating the quotes, the login program executes the file named by the first argument, with all arguments passed to it as a parsed command line.

Setting up a password file is discussed, with examples, in Section IV of this manual.

NAME

pipe - pipeline pseudo files

FUNCTION

A pipe is a data connection that can be written as a sequential file on one end and read as a sequential file on the other; typically pipes are setup between cooperating programs by the shell. It is implemented as an unnamed plain file in some filesystem (determined by the Idris resident), with synchronization to keep the writer from getting too far ahead of the reader and to alert a waiting reader when data is written to the pipe.

Arbitrary eight-bit bytes can be passed through a pipe, and arbitrary numbers of bytes may be written or read at a time. There will never be more than 4,096 bytes stored in the pipeline, however, and the reader is often given just what is available, if data is present but less than what is requested. If all writers close their end of the pipe, then an end of file will be delivered to all readers once the pipe drains. If all readers close their end of the pipe, then the "broken pipe" signal is sent to all writers.

Any attempt to perform direct access (seeking) on a pipe will report failure, since the positioning information (seek offset) is commandeered for internal use by the pipe administrator.

If the system is brought down unexpectedly, as on loss of power, while a pipe is active, it may appear in its host filesystem as an allocated inode with no links to it. Its resources may be reclaimed by clearing the inode and reclaiming the lost free blocks (see fcheck and icheck in Section IV of this manual).

SEE ALSO

filesystem, plain

NAME

plain - plain files

FUNCTION

A plain file is the simplest type of file, one residing in a filesystem and not being used as a directory. It is implemented as an "inode", or central repository of status information, with zero or more blocks of data (and blocks of pointers to data blocks) defining its contents. It appears to the world as just an ordered sequence of eight-bit bytes, whose contents are arbitrary and whose length is determined by the highest numbered byte ever written; unwritten intermediate bytes read as zeros. It is reachable by one or more "pathnames" using the filesystem directory structure.

Plain files can be accessed directly (random reads or writes), an arbitrary number of contiguous bytes at a time. There are, of course, performance benefits in accessing plain files as 512-byte blocks, and many programs take advantage of this benefit. Plain files are truncated to zero length when "created" (first made to exist or whenever completely rewritten), then are extended simply by writing to ever higher byte positions. A plain file can be as long as 16,777,215 bytes, assuming there is sufficient space on its host filesystem.

SEE ALSO

directory, filesystem

NAME

print - printable file restrictions

FUNCTION

A print file is a text file, with the added restrictions that it contain no funny characters. It also should contain no lines too long for the printing device, but that is a restriction that varies widely among devices and hence is not enforced by most text processing programs; each device is at liberty to truncate or fold long lines as it sees fit.

A funny character is, basically, one that the typical printer can't deal with. In the ASCII character set, the octal codes [0040, 0177) cause certain graphics to print, so they're all right. There are also defined codes for "whitespace", or standard actions:

CODE	SYMBOL	KEY	MEANING
0007	'\7'	ctl-G	bell
0010	'\b'	ctl-H	backspace
0011	'\t'	ctl-I	horizontal tab
0012	'\n'	ctl-J	line feed (newline)
0013	'\v'	ctl-K	vertical tab
0014	'\f'	ctl-L	form feed (newpage)
0015	'\r'	ctl-M	carriage return

Of these, only the newline is reliably nonfunny, others being provided at the whim of the hardware designer, or emulated at the discretion of the software. (Most terminals provide at least some of these codes as special keys, so it is not always necessary to type the control sequence shown to input the character. On the other hand, because most terminals provide a big fat carriage return key and a skinny little line feed key, the Idris resident is easily persuaded to map carriage returns to newlines on input.)

SEE ALSO

ASCII, text

NAME

salt - encryption salt file

SYNOPSIS

/adm/salt

FUNCTION

/adm/salt is used, by programs that must encrypt a password, as a starting value for the encryption process. Up to eight characters are read from /adm/salt, if possible, to overwrite the builtin string "password", which is then exclusive-ored, on a character by character basis, with the user′s loginid. The resulting mish mash is then encrypted. Including the loginid with the salt means that two users on the same system may elect the same password and never know it. And by varying the contents of /adm/salt among different systems, users may indulge in the same password multiple times and not have their bad habits revealed. Moreover, if the salt has no read permission, the work needed to crack an encrypted password increases dramatically.

Since all programs that must encrypt a password tend to be privileged, it is thus permissible as well as advisable to deny read permission to all but the superuser for /adm/salt.

NAME

stty - predefined terminal attributes

SYNOPSIS

/adm/stty

FUNCTION

/adm/stty is the text file used by the standard utility stty, in conjunction with the -type# flag, to set terminal attributes to a predefined set. Each line consists of the terminal type, used to match #, followed by attributes written just as for the stty command.

SEE ALSO

stty(II)

NAME

text - text file restrictions

FUNCTION

The commonest restriction imposed on a file or data stream is that it be a "text file". Colloquially, this means that it is human readable -- it can be copied to a terminal or printer without doing violence to either.

More precisely, a text file is presumed to consist of zero or more "lines", where a line consists of zero or more characters followed by a newline (usually written "\n" when it must be made visible, and usually causing the same effect as a typewriter carriage return when displayed). This implies, of course, that a non-empty text file must end in a newline; and indeed, any text processing program reserves the right to deal with an "incomplete last line" (one not ending in a newline) by: a) discarding it, b) leaving it incomplete, or c) appending a newline to it.

No program that processes a text file is obliged to deal with a line longer than 512 characters, counting the terminating newline; on the other hand, all programs are obliged to deal with lines at least this long. If a line is longer than 512 characters, a text processing program reserves the right to: a) truncate it and leave it incomplete, b) truncate it and append a newline, c) fold it by inserting newlines, or d) process it completely.

There are no restrictions on the character set that may appear in a text file, although clearly the newline has a special meaning. Nonprinting characters of all sorts, including NULs, should be properly processed by all programs designed to manipulate text. If the file is indeed destined to be printed out, however, additional restrictions apply, as discussed under print files.

SEE ALSO

ASCII, print

NAME

tp - tape archive format

FUNCTION

The standard utility tp administers a file, typically written to tape, that records the attributes and contents of a number of files. Since tp files are compatible across all implementations of Idris, and with UNIX/V6 tp tapes, they can be used as a means of communicating small numbers of files between UNIX and Idris systems.

A tp file is organized as a series of 512-byte blocks. Block zero is unused and may serve as a bootstrap, for some machines, that can load stand alone programs from the tp file. Blocks 1-62 contain 496 directory records, each containing 64 bytes in the format:

bytes 0-31: the NUL-terminated pathname of the archived file. If the first two bytes are NULs, the entry is empty.

bytes 32-33: the attributes or mode of the file, taken from its inode, written less significant byte first.

byte 34: the userid of the file owner.

byte 35: the groupid of the file owner.

byte 36: unused.

bytes 37-39: the size of the file in bytes, written most significant byte, then least significant byte, then middle byte [sic].

bytes 40-43: the date of last modification, written as a filesystem date.

bytes 44-45: the tp file relative block number of the start of the file contents.

bytes 46-61: unused.

bytes 62-63: the checksum, such that the record, taken as 32 short integers written less significant byte first, sums to zero.

Blocks 63 on up hold file contents, an integral number of blocks for each file, taken in order. If a directory record is empty, its contents occupy zero bytes.

SEE ALSO

filesystem

NAME

tty - terminal files

FUNCTION

A tty file is a direct connection to a peripheral device, which is a character special file employing a standard protocol for supporting "teletype" (terminal) input/output. Reading a tty file waits for keystrokes from a terminal keyboard; writing it sends characters to the terminal display; positioning requests (seeks) are simply ignored.

A tty is capable of editing its input on a line by line basis, of echoing its input to its output for full duplex terminal operation, of throttling its output under keyboard control, of inserting tailored delays in its output for sluggish displays, and of turning certain keystrokes into signals for controlling active processes. The stty command can be used to display or alter the operating mode of any tty file in a standard fashion.

An important special feature of tty files is their ability to send signals to processes associated with them. When a tty file is "opened", i.e. requested for use by a login process (typically), it makes the process wait until a connection is actually established (someone dials up, or hooks up a terminal, or turns it on) -- assuming the hardware is capable of detecting such a change. It then appoints itself "controlling tty" for the process, unless some other tty has already claimed the honor. As a controlling tty, it interprets certain keystrokes as immediate requests to broadcast associated signals to that process and to any descendants it may have; thus, a misbehaving process can be terminated, or a long editor display can be cut short.

Using stty, a tty can have its baud rates changed for input and/or output, assuming the physical device can respond to such requests; it can also be placed in "raw" mode. A raw tty outputs eight-bit bytes transparently (unchanged), with no delays, and accepts all keystrokes without interpretation as eight bit bytes. The only control left is to break the connection (hang up, or unplug the terminal, or turn it off), which causes a hangup signal to be sent to all processes under control of the tty -- assuming the physical device can detect a broken connection! Thus, raw mode should be left to programs that know what they are doing.

When not in raw mode, a tty cares about a number of other options:

parity - The state of the "eighth bit" accompanying seven-bit ASCII codes can be checked on input (bad bytes discarded) and determined on output (to please parity checkers in connected devices). Four combinations are provided: 1) accept any parity, generate zero parity bit; 2) accept even parity (sum of all bits in the byte is even, counting parity bit), generate even parity; 3) accept odd parity, generate odd parity; and 4) accept any parity, generate one parity bit. In all cases, bytes input have their parity bits set to zero.

map CR - A carriage return can be mapped on input to a newline, and a newline can be mapped on output to a carriage return followed by a line feed (newline). The former service is needed for terminals that provide a large return key, rather than a large newline key; the latter is needed for the many terminals that require both codes to advance to the next line.

expand HT - Each horizontal tab output can be expanded to the appropriate number of spaces to simulate tab stops every four columns. Useful on terminals that can´t interpret tabs.

map uppercase - Each uppercase letter can be mapped to lowercase on input, unless preceded by the escape character ´\´. Also, certain poorly supported characters can be generated on input by two character escape sequences: "\´" becomes ´`´, "\(" becomes ´{´, "\!" becomes ´|´, "\)" becomes ´)´, and "\^" becomes ´~´. With this option, each lowercase letter output is mapped to uppercase, and each of the above mappings is reversed on output. Needed on primitive terminals that support only one case of letters.

echo - Each input character can be written out immediately. Needed to support full duplex operation with decent human feedback.

hangup on close - The hardware device can be encouraged to hangup a phone line, assuming it has the ability to do so, when "closed", i.e. when a user logs out (typically). Used to minimize the chance of security breaches.

delay - Certain characters can be followed on output by delays, to give mechanical devices time to move, typically. The supported delays, in units of 1/60 second, are: horizontal tab [0, 4, 8, 12], carriage return [0, 4, 8, 12], newline [0, 4, 8, 12], backspace [0, 8], formfeed or vertical tab [0, 128].

erase - The character typed to erase the last character from the input, normally a backspace, can be changed to any other character, or disabled.

kill - The character typed to kill an entire line of input, normally ´@´, can be changed to any other character, or disabled.

Note that any pending input is discarded when a tty is first opened or when its mode is altered by stty; the former is convenient for eliminating trash at startup time, but the latter can come as a surprise to those who have grown accustomed to the convenience of type ahead. Programs that seek keyboard reassurance, with messages such as "are\ you\ sure?\ ", also perform an internal stty to be sure of getting only timely answers.

A tty file tends to deliver up a line at a time when read, with three exceptions. If the file is in raw mode, as many characters as possible are delivered. If the number of characters requested is less than the total line, a partial line is delivered. And if an EOT (ctl-D) is typed, that character is discarded and any partial line is delivered, even if it has no characters. Thus, typing an EOT at the beginning of a line serves to report an end of file to the reader, a process that can be repeated any number of times during a terminal session. Reporting end of file (by typing EOT) to the shell is the conventional way of logging out.

The complete set of keystrokes with special meaning is:

EOT - (ctl-D) Discard EOT and deliver partial line. Used to simulate end of file.

BS - (ctl-H or backspace) Discard last character on current line, if any, along with BS; output as BS, space, BS. Possibly delay. Default erase character, can be altered.

HT - (ctl-I or horizontal tab) Possibly expand to 1-4 spaces on output. Possibly delay.

NL - (ctl-J or newline) Deliver current line, terminated by newline. Possibly map to carriage return, line feed on output. Possibly delay.

VT - (ctl-K or vertical tab) Possibly delay.

NP - (ctl-L or newpage) Possibly delay.

CR - (ctl-M or carriage return) Possibly map to newline on input. Possibly delay.

DC1 - (ctl-Q) Discard DC1 and resume suspended output.

DC3 - (ctl-S) Discard DC3 and suspend output.

FS - (ctl-\) Discard FS and send quit signal.

\&´@´ - Discard current line, if any, along with ´@´; output as ´@´, NL. Default kill character, can be altered.

\&´\´ - If next character is erase or kill, discard ´\´ and take next literally; if next character is uppercase and mapping is in effect, discard ´\´ and don´t map; otherwise take ´\´ literally.

DEL - (or rubout) Discard DEL and send interrupt signal.

BREAK - same as DEL.

If a tty file is connected to a source of high speed input, such as another computer, it runs the risk of being overwhelmed with input. Thus, if the reader gets behind by approximately 128 characters, a ctl-S is automatically output; the balancing ctl-Q is output only when the reader has caught up more. If this fails to stop the source of input, a more drastic action is taken -- if the reader gets behind by 256 characters, all input is discarded without remark. Care must thus be taken when connecting Idris directly to inconsiderate machines.

SEE ALSO
character, stty(II)

NAME

where - system identification psuedo file

SYNOPSIS

/dev/where

FUNCTION

/dev/where is a character special device which reads out a text string from the resident, typically to advertise the presumed location and nature of the resident CPU. /dev/where can also be rewritten.

The text strings /dev/where deals with are at most 40 bytes in length and terminate, when read, with the first newline.

BUGS

Length should be 64 characters.

NAME

who - login active file

SYNOPSIS

/adm/who

FUNCTION

/adm/who is written by the login program to inform the who command about who is currently active on the system. It contains one 26-byte record for each login port, as determined at system initialization time (described in Section IV of this manual).

The format of each record is the same as for the history file /adm/log, except that no entries are made for time changes or system startups.

SEE ALSO

log

NAME

zone - time zone information

SYNOPSIS

/adm/zone

FUNCTION

/adm/zone is used by all programs that need to display system time in terms of local usage. It is a text file whose first ten characters are interpreted as:

bytes 0-3: four decimal digits for timezone, expressed in hours and minutes west of Greenwich Meridian (Boston MA, New York NY, and Secaucus NJ are all in zone "0500").

bytes 4-6: three ASCII characters for name of timezone during standard time (e.g. "EST").

bytes 7-9: three ASCII characters for name of timezone during daylight savings time (e.g. "EDT"). If this field is absent, or begins with a newline, it is assumed that the U.S. daylight savings laws do not apply, so standard time is always used.

If the file cannot be read and understood, builtin defaults are used. As shipped, these correspond to the parenthetical examples given, i.e. the file contents "0500ESTEDT".