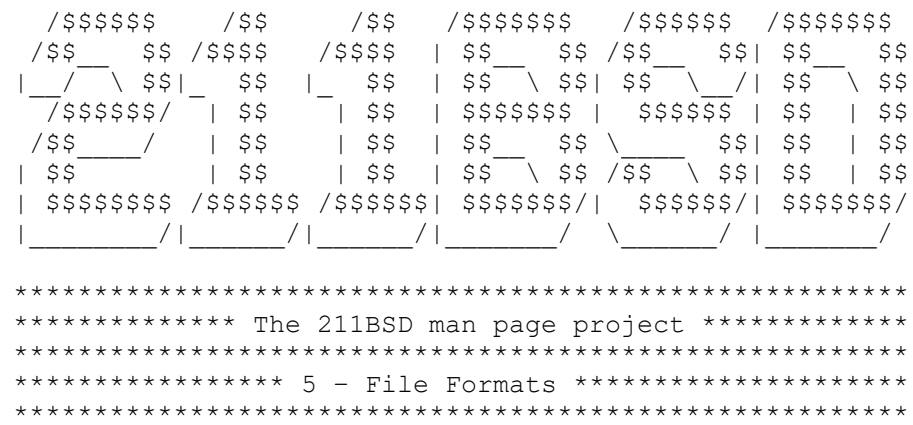


Inspired by:
==========

SimH	http://simh.trailing-edge.com/
PiDP11	https://obsolescence.wixsite.com/obsolescence/pidp-11
BSD 2.11	https://wfjm.github.io/home/211bsd/

Presented by the ShadowTron Blog
==========

https://www.youtube.com/c/shadowtronblog
www.shadowtron.com
shadowtronblog <at> gmail <dot> com

Other manuals in the series
==========

```
    Manual 1  - Commands and Application Programs
    Manual 2  - System Calls
    Manual 3  - C Library Subroutines
    Manual 3F - Fortran Library
    Manual 4  - Special Files
==> Manual 5  - File Formats
    Manual 6  - Games
    Manual 7  - Miscellaneous
    Manual 8  - System Maintenance
```

Version 3.0

****
**** Manual 5 - File Formats ****
****

Page	Command	Description
3	a.out	assembler and link editor output
8	acct	execution accounting file
10	aliases	aliases file for sendmail
11	core	format of memory image file
12	dbx	dbx symbol table information
18	dir	format of directories
20	disktab	disk description file
22	dtab	automatic system configuration device table
23	dump	incremental dump format
26	fs	format of file system volume
32	fstab	static information about the filesystems
35	gettytab	terminal configuration data base
38	group	group file
39	hosts	host name data base
40	L-devices	UUCP device description file
43	L-dialcodes	UUCP phone number index file
45	L.aliases	UUCP hostname alias file
46	L.cmds	UUCP remote command permissions file
47	L.sys	UUCP remote host description file
53	map3270	database for mapping ascii keystrokes into IBM 3270 keys
59	networks	network name data base
60	passwd	password file
62	phones	remote host phone number data base
63	plot	graphics interface
65	printcap	printer capability data base
68	protocols	protocol name data base
69	remote	remote host description file
71	resolver	resolver configuration file
72	services	service name data base
73	shells	shell database
74	stack	stack frame conventions
76	syserrlst	error message file format
78	tar	tape archive file format
80	termcap	terminal capability data base
99	tp	DEC/mag tape formats
100	ttys	terminal initialization data
102	types	primitive system data types
104	tzfile	time zone information
106	USERFILE	UUCP pathname permissions file
109	utmp	login records
111	uuencode	format of an encoded uuencode file
112	vfont font	formats for the Benson-Varian or Versatec
114	vgrindefs	vgrind's language definition data base

NAME

a.out - assembler and link editor output

SYNOPSIS

```
#include <a.out.h>
```

DESCRIPTION

A.out is the output file of the assembler as(1) and the link editor ld(1). Both programs make a.out executable if there were no errors and no unresolved external references. Layout information as given in the include file for the PDP11 is:

```
/*
 * Header prepended to each a.out file.
 */
struct   exec {
       int      a_magic;          /* magic number */
       unsigned int  a_text;      /* size of text segment */
       unsigned int  a_data;      /* size of initialized data */
       unsigned int  a_bss;       /* size of uninitialized data */
       unsigned int  a_syms;      /* size of symbol table */
       unsigned int  a_entry;     /* entry point */
       unsigned int  a_unused;    /* not used */
       unsigned int  a_flag;      /* relocation info stripped */
};

#define  NOVL         15          /* number of overlays */
struct   ovlhdr {
       int      max_ovl;          /* maximum overlay size */
       unsigned int  ov_siz[NOVL]; /* size of i'th overlay */
};

struct   xexec {
       struct exec e;
       struct ovlhdr o;
};

#define  A_MAGIC1     0407     /* normal */
#define  A_MAGIC2     0410     /* read-only text */
#define  A_MAGIC3     0411     /* separated I&D */
#define  A_MAGIC4     0405     /* overlay */
#define  A_MAGIC5     0430     /* auto-overlay (nonseparate) */
#define  A_MAGIC6     0431     /* auto-overlay (separate) */

/*
 * Macros which take exec structures as arguments and tell whether
 * the file has a reasonable magic number or offset to text.
 */
#define  N_BADMAG(x) \
       (((x).a_magic)!=A_MAGIC1 && ((x).a_magic)!=A_MAGIC2 && \
       ((x).a_magic)!=A_MAGIC3 && ((x).a_magic)!=A_MAGIC4 && \
```

```
        ((x).a_magic)!=A_MAGIC5 && ((x).a_magic)!=A_MAGIC6)

#define  N_TXTOFF(x) \
        ((x).a_magic==A_MAGIC5 || (x).a_magic==A_MAGIC6 ? \
        sizeof(struct ovlhdr) + sizeof(struct exec) : sizeof(struct exec))

/*
 * The following were added as part of the new object file format.  They
 * call functions because calculating the sums of overlay sizes was too
 * messy (and verbose) to do 'inline'.
 *
 * NOTE: if the magic number is that of an overlaid object the program
 * must pass an extended header ('xexec') as the argument.
*/

off_t    n_stroff(), n_symoff(), n_datoff(), n_dreloc(), n_treloc();

#define  N_STROFF(e) (n_stroff(&e))
#define  N_SYMOFF(e) (n_symoff(&e))
#define  N_DATOFF(e) (n_datoff(&e))
#define  N_DRELOC(e) (n_dreloc(&e))
#define  N_TRELOC(e) (n_treloc(&e))
```

The file has five sections: a header, the program text and data, relocation information, a symbol table and a strings table (in that order). The last three may be omitted if the program was loaded with the `-s' option of ld or if the symbols and relocation have been removed by strip(1).

In the header the sizes of each section are given in bytes, but are even. The size of the header is not included in any of the other sizes.

When an a.out file is executed, three or four logical segments are set up: the text segment, a possible text overlay segment, the data segment (with uninitialized data, which starts off as all 0, following initialized), and a stack. The text segment begins at 0 in the core image; the header is not loaded.

Non-overlaid objects: If the magic number in the header is A_MAGIC1 (0407), it indicates that the text segment is not to be write-protected and shared, so the data segment is immediately contiguous with the text segment. This is the oldest kind of executable program and is the default; it should not be used for production binaries. If the magic number is A_MAGIC2 (0410), the data segment begins at the first 0 mod 8K byte boundary following the text segment, and the text segment is not writable by the program; if other processes are executing the same file, they will share the text segment. If the magic number is A_MAGIC3 (0411), the text segment is again pure, write-protected, and shared, and

moreover instruction and data space are separated; the text and data segment both begin at location 0. This format is only runnable on processors which support separate instruction and data space but can provide significantly more data space than an A_MAGIC2 format of the same object.

Text replacement objects : If the magic number is A_MAGIC4 (0405), the text segment is overlaid on an existing non-overlaid pure (A_MAGIC2 or A_MAGIC3) or text replacement (A_MAGIC4) text segment and the existing data segment is preserved. The text segment of the previous memory image must be the same size as that of the text replacement object being loaded. There is, unfortunately, no loader support to help achieve this requirement. The text replacement format is useful for objects which need a large amount of data space on non-separate I&D processors.

Overlaid objects : If the magic number is A_MAGIC5 (0430), a base text segment is write-protected and shared and is followed by a text overlay segment. There are a maximum of NOVL overlays, all pure and shared. The base segment runs from 0 to txtsiz. The overlay region begins at the next 0 mod 8k byte boundary, which is as large as the largest overlay. When running, any one of the overlays can be mapped into this region. The data segment begins at the following 0 mod 8k byte boundary. If the magic number is A_MAGIC6 (0431), the situation is the same as for type A_MAGIC5 except that instruction and data spaces are separated and both begin at location 0. As with the A_MAGIC3 format, an a.out in A_MAGIC6 format can only be run on a processor which supports separate I&D, but again can provide significantly more data space than A_MAGIC5 format. Both A_MAGIC5 and A_MAGIC6 executable files have a second header between the normal a.out header and the start of the text image; it contains the maximum overlay size and the sizes of each of the overlays. The text images of the overlays follow the text in the object file.

The stack segment will occupy the highest possible locations in the core image: growing downwards from 0177776(8). The stack segment is automatically extended as required. The data segment is only extended as requested by brk(2).

The include file a.out.h defines _AOUT_INCLUDE_, the include file nlist.h does not. This permits compile time initialization of the n_name field for programs that are not looking at the executable header.

The layout of a symbol table entry and the principal flag values that distinguish symbol types are given in the include file as follows:

```
struct   nlist {
#ifdef   _AOUT_INCLUDE_
       union {
                char *n_name;/* In memory address of symbol name */
                off_t n_strx;/* String table offset (file) */
       } n_un;
#else
       char       *n_name;   /* symbol name (in memory) */
#endif
       u_char     n_type;    /* Type of symbol - see below */
       char       n_ovly;    /* Overlay number */
       u_int      n_value;   /* Symbol value */
};

/*
 * Simple values for n_type.
 */
#define  N_UNDF       0x0        /* undefined */
#define  N_ABS        0x1        /* absolute */
#define  N_TEXT       0x2        /* text symbol */
#define  N_DATA       0x3        /* data symbol */
#define  N_BSS        0x4        /* bss symbol */
#define  N_REG        0x14       /* register name */
#define  N_FN         0x1f       /* file name symbol */

#define  N_EXT        0x20       /* external bit, or'ed in */
#define  N_TYPE       0x1f       /* mask for all the type bits */

/*
 * Format for namelist values.
 */
#define  N_FORMAT     "%06o"
```

If a symbol's type is undefined external, and the value field is non-zero, the symbol is interpreted by the loader ld as the name of a common region whose size is indicated by the value of the symbol.

The value of a word in the text or data which is not a portion of a reference to an undefined external symbol is exactly that value which will appear in memory when the file is executed. If a word in the text or data involves a reference to an undefined external symbol, as indicated by the relocation information, then the value stored in the file is an offset from the associated external symbol. When the file is processed by the link editor and the external symbol becomes defined, the value of the symbol will be added into the word in the file.

If relocation information is present, it amounts to one word per word of program text or initialized data. There is no relocation information if the `relocation info stripped'

flag in the header is on. Automatic-overlay (A_MAGIC5 and A_MAGIC6) files do not contain relocation information.

Bits 1-3 of a relocation word indicate the segment referred to by the text or data word associated with the relocation word:

000 absolute number
002 reference to text segment
004 reference to initialized data
006 reference to uninitialized data (bss)
010 reference to undefined external symbol

Bit 0 of the relocation word indicates, if 1, that the reference is relative to the pc (e.g. `clr x'); if 0, that the reference is to the actual symbol (e.g., `clr *$x').

The remainder of the relocation word (bits 15-4) contains a symbol number in the case of external references, and is unused otherwise.

The string table begins with a longword containing the length of the string table (including the longword itself). All strings are null terminated.

The first symbol is numbered 0, the second 1, etc.

SEE ALSO

as(1), ld(1), nm(1), strip(1), nlist(3)

BUGS

The current implementation places a maximum length of 32 characters for symbol names in a.out files. This is (relatively) easily raised with the caveat that the linker and other programs which look at symbol tables will slow down even more than they already have.

The 4BSD a.out format has been implemented. This involved modifying the first phase of the C compiler (/lib/c0), the assembler (/bin/as), the debugger adb(1), the linker ld(1), and then simply porting the 4.3BSD/Net-2 ar(1), nm(1), ranlib(1), strip(1) and nlist(3).

As part of this effort the include file short_names.h has gone away.

```
NAME
     acct - execution accounting file

SYNOPSIS
     #include <sys/acct.h>

DESCRIPTION
     The acct(2) system call arranges for entries to be made in
     an accounting file for each process that terminates.  The
     accounting file is a sequence of entries whose layout, as
     defined by the include file is:

     /*
      * Copyright (c) 1982, 1986 Regents of the University of California.
      * All rights reserved.  The Berkeley software License Agreement
      * specifies the terms and conditions for redistribution.
      *
      *      @(#)acct.h              2.2(2.11BSD) 1999/2/19
      */

     /*
      * Accounting structures;
      * these use a comp_t type which is a 3 bits base 8
      * exponent, 13 bit fraction ``floating point'' number.
      * Units are 1/AHZ seconds.
      */
     typedef u_short comp_t;

     struct  acct
     {
           char     ac_comm[10];  /* Accounting command name */
           comp_t   ac_utime;     /* Accounting user time */
           comp_t   ac_stime;     /* Accounting system time */
           comp_t   ac_etime;     /* Accounting elapsed time */
           time_t   ac_btime;     /* Beginning time */
           uid_t    ac_uid;       /* Accounting user ID */
           gid_t    ac_gid;       /* Accounting group ID */
           short    ac_mem;       /* average memory usage */
           comp_t   ac_io;        /* number of disk IO blocks */
           dev_t    ac_tty;       /* control typewriter */
           char     ac_flag;      /* Accounting flag */
     };

     #define AFORK    0001       /* has executed fork, but no exec */
     #define ASU      0002       /* used super-user privileges */
     #define ACOMPAT  0004       /* used compatibility mode */
     #define ACORE    0010       /* dumped core */
     #define AXSIG    0020       /* killed by a signal */
     #define ASUGID   0040       /* setuser/group id privileges used */

     /*
      * 1/AHZ is the granularity of the data encoded in the various

Printed      May 19, 1986                  1
ACCT(5)            UNIX Programmer's Manual         ACCT(5)
```

```
 * comp_t fields. This is not necessarily equal to hz.
 */
#define AHZ 64

#ifndef KERNEL
#define _PATH_ACCTD     "/usr/libexec/acctd"
#define _PATH_ACCTFILE  "/usr/adm/acct"
#define _PATH_ACCTDPID  "/var/run/acctd.pid"
#define _PATH_ACCTDCF   "/etc/acctd.cf"
#define _PATH_DEVALOG   "/dev/acctlog"
#endif
```

If the process was created by an execve(2), the first 10 characters of the filename appear in ac_comm. The accounting flag contains bits indicating whether execve(2) was ever accomplished, and whether the process ever had super-user privileges.

SEE ALSO

acct(2), execve(2), sa(8)

NAME

aliases - aliases file for sendmail

SYNOPSIS

/etc/aliases

DESCRIPTION

This file describes user id aliases used by /usr/sbin/sendmail. It is formatted as a series of lines of the form

name: name_1, name2, name_3, . . .

The name is the name to alias, and the name_n are the aliases for that name. Lines beginning with white space are continuation lines. Lines beginning with `#' are comments.

Aliasing occurs only on local names. Loops can not occur, since no message will be sent to any person more than once.

After aliasing has been done, local and valid recipients who have a ``.forward'' file in their home directory have messages forwarded to the list of users defined in that file.

This is only the raw data file; the actual aliasing information is placed into a binary format in the files /etc/aliases.dir and /etc/aliases.pag using the program newaliases(1). A newaliases command should be executed each time the aliases file is changed for the change to take effect.

SEE ALSO

newaliases(1), dbm(3X), sendmail(8)
SENDMAIL Installation and Operation Guide.
SENDMAIL An Internetwork Mail Router.

BUGS

Because of restrictions in dbm(3X) a single alias cannot contain more than about 1000 bytes of information. You can get longer aliases by ``chaining''; that is, make the last name in the alias be a dummy name which is a continuation alias.

NAME

core - format of memory image file

SYNOPSIS

#include <sys/param.h>

DESCRIPTION

The UNIX System writes out a memory image of a terminated process when any of various errors occur. See sigvec(2) for the list of reasons; the most common are memory violations, illegal instructions, bus errors, and user-generated quit signals. The memory image is called `core' and is written in the process's working directory (provided it can be; normal access controls apply).

The core file consists of the u. area, whose size (in 64 byte `clicks') is defined by the USIZE manifest in the <sys/param.h> file. The u. area starts with a user structure as given in <sys/user.h>. The rest of the u. area consists of the kernel stack for the terminated process which includes (among other things) the processor registers at the time of the fault; see the system listings for the format of this area. The remainder of the core file consists first of the data pages and then the stack pages of the process image. The amount of data space image in the core file is given (in clicks) by the variable u_dsize in the u. area. If the text segment was not write-only and and shared it is included as the first etext bytes of the data image where etext is taken from the symbol table of the object file which generated the memory image. The amount of stack image in the core file is given (in clicks) by the variable u_ssize in the u. area.

In general the debugger adb(1) is sufficient to deal with core images.

SEE ALSO

adb(1), sigvec(2), stack(5)

NAME

dbx - dbx symbol table information

DESCRIPTION

The compiler symbol information generated for dbx(1) uses the same structure as described in stab(5), with additional type and scope information appended to a symbol's name. The assembler directive used to describe symbol information has the following format:

stabs ``string'',kind,0,size,value

String contains the name, source language type, and scope of the symbol, kind specifies the memory class (e.g., external, static, parameter, local, register), and size specifies the byte size of the object, if relevant. The third field (0 above) is unused. For a global variable or a type, value is unused; for a local variable or parameter, it is the offset from the frame pointer, for a register variable, it is the associated register number.

The different kinds of stab entries are interpreted by dbx as follows:

N_GSYM The symbol is a global variable (e.g., .comm variable). The variable's address can be found from the corresponding ld(1) symbol entry, thus the value field for N_GSYM symbols is ignored. For example, a global variable ``x'' will have both an N_GSYM entry and an ld(1) entry (e.g., N_BSS + N_EXT). See a.out(5) for details about these other entries. of

N_FUN The symbol is a procedure or function. The size field contains the line number of the entry point. The value field contains the address of the entry point (in the text segment).

N_STSYM The symbol is a statically allocated variable for which an initial value has been specified. The value field contains the address of the variable (in the data segment).

N_LCSYM The symbol is statically allocated, but not initialized.

N_RSYM The symbol is a register variable whose value is kept in the register denoted by the value field.

N_PSYM The symbol is a parameter whose value is pushed on the stack before the call. The value field contains the offset from the argument base pointer

(on the VAX, the ap register).

N_LSYM The symbol is a local variable whose value is stored in the most recently defined procedure's stack frame. The value is the (often negative) offset from the frame pointer (on the VAX, the fp register).

N_PC, N_MOD2
The symbol defines separate compilation information for pre-linking checking for Berkeley Pascal and DEC Modula-2 programs respectively. For Pascal, the value field contains the line number that the symbol is defined on. The value field is not used for Modula-2.

Most of the source level information about a symbol is stored in the string field of the stab entry. Since strings are kept in a separate string table in the a.out file, they can be arbitrarily long. Thus there are no restrictions on the kind or length of information in the string field, and it was not necessary to modify the assembler or loader when extending or modifying the format of this information.

Below is a grammar describing the syntax of the symbol string. Except in the case of a constant whose value is a string, there are no blanks in a symbol string.

```
NAME:     [a-zA-Z_][a-zA-Z_0-9]*
INTEGER:  [-][0-9][0-9]*
REAL:     [+-][0-9]*(.[0-9][0-9]*|)([eE]([+-]|)[0-9][0-9]*|)
STRING:   ``.*''
BSTRING:  .*

String:
  NAME `:' Class
  `:' Class

Class:
  `c' `=' Constant `;'
  Variable
  Procedure
  Parameter
  NamedType
  `X' ExportInfo  -- export or import information (for N_MOD2 only)

Constant:
  `i' INTEGER
  `r' REAL
  `c' OrdValue
  `b' OrdValue
```

```
  `s' STRING
  `e' TypeId `,' OrdValue
  `S' TypeId `,' NumElements `,' NumBits `,' BSTRING

OrdValue:
  INTEGER

NumElements:
  INTEGER

NumBits:
  INTEGER

Variable:
  TypeId          -- local variable of type TypeId
  `r' TypeId  -- register variable of type TypeId
  `S' TypeId  -- module variable of type TypeId (static global in C)
  `V' TypeId  -- own variable of type TypeId (static local in C)
  `G' TypeId  -- global variable of type TypeId

Procedure:
  Proc                          -- top level procedure
  Proc `,' NAME `,' NAME        -- local to first NAME,
                                -- second NAME is corresponding ld symbol

Proc:
  `P'   -- global procedure
  `Q'   -- local procedure (static in C)
  `I'   -- internal procedure (different calling sequence)
  `F' TypeId  -- function returning type TypeId
  `f' TypeId  -- local function
  `J' TypeId  -- internal function

Parameter:
  `p' TypeId  -- value parameter of type TypeId
  `v' TypeId  -- reference parameter of type TypeId

NamedType:
  `t' TypeId  -- type name for type TypeId
  `T' TypeId  -- C structure tag name for struct TypeId

TypeId:
  INTEGER                       -- Unique (per compilation) number of type
  INTEGER `=' TypeDef           -- Definition of type number
  INTEGER `=' TypeAttrs TypeDef

--
-- Type attributes are extra information associated with a type,
-- such as alignment constraints or pointer checking semantics.
-- Dbx interprets some of these, but will ignore rather than complain
-- about any it does not recognize.  Therefore this is a way to add
-- extra information for pre-linking checking.
```

```
--
TypeAttrs:
  `@' TypeAttrList `;'

TypeAttrList:
  TypeAttrList `,' TypeAttr
  TypeAttr

TypeAttr:
  `a' INTEGER      -- align boundary
  `s' INTEGER      -- size in bits
  `p' INTEGER      -- pointer class (e.g., checking)
  BSTRING          -- something else

TypeDef:
  INTEGER
  Subrange
  Array
  Record
  `e' EnumList `;'   -- enumeration
  `*' TypeId         -- pointer to TypeId
  `S' TypeId         -- set of TypeId
  `d' TypeId         -- file of TypeId
  ProcedureType
  `i' NAME `:' NAME `;'    -- imported type ModuleName:Name
  `o' NAME `;'             -- opaque type
  `i' NAME `:' NAME `,' TypeId `;'
  `o' NAME `,' TypeId `;'

Subrange:
  `r' TypeId `;' INTEGER `;' INTEGER

Array:
  `a' TypeId `;' TypeId      -- array [TypeId] of TypeId
  `A' TypeId                 -- open array of TypeId
  `D' INTEGER `,' TypeId     -- N-dim. dynamic array
  `E' INTEGER `,' TypeId     -- N-dim. subarray

ProcedureType:
  `f' TypeId `;'            -- C function type
  `f' TypeId `,' NumParams `;' TParamList `;'
  `p' NumParams `;' TParamList `;'

NumParams:
  INTEGER

Record:
  `s' ByteSize FieldList `;'      -- structure/record
  `u' ByteSize FieldList `;'      -- C union

ByteSize:
  INTEGER
```

```
FieldList :
  Field
  FieldList Field

Field:
  NAME `:' TypeId `,' BitOffset `,' BitSize `;'

BitSize:
  INTEGER

BitOffset:
  INTEGER

EnumList:
  Enum
  EnumList Enum

Enum:
  NAME `:' OrdValue `,'

ParamList:
  Param
  ParamList Param

Param:
  NAME `:' TypeId `,' PassBy `;'

PassBy:
  INTEGER

TParam:
  TypeId `,' PassBy `;'

TParamList :
  TParam
  TParamList TParam

Export:
  INTEGER ExportInfo

ExportInfo:
  `t' TypeId
  `f' TypeId `,' NumParams `;' ParamList `;'
  `p' NumParams `;' ParamList `;'
  `v' TypeId
  `c' `=' Constant
```

A `?' indicates that the symbol information is continued in the next stab entry. This directive can only occur where a `;' would otherwise separate the fields of a record or constants in an enumeration. It is useful when the number of

elements in one of these lists is large.

SEE ALSO
dbx(1), stab(5), a.out(5)

```
NAME
     dir - format of directories

SYNOPSIS
     #include <sys/types.h>
     #include <sys/dir.h>

DESCRIPTION
     A directory behaves exactly like an ordinary file, save that
     no user may write into a directory.  The fact that a file is
     a directory is indicated by a bit in the flag word of its
     i-node entry; see fs(5).  The structure of a directory entry
     as given in the include file is:

  /*
   * A directory consists of some number of blocks of DIRBLKSIZ
   * bytes, where DIRBLKSIZ is chosen such that it can be transferred
   * to disk in a single atomic operation (e.g. 512 bytes on most machines).
   *
   * Each DIRBLKSIZ byte block contains some number of directory entry
   * structures, which are of variable length.     Each directory entry has
   * a struct direct at the front of it, containing its inode number,
   * the length of the entry, and the length of the name contained in
   * the entry.  These are followed by the name padded to a 4 byte boundary
   * with null bytes.  All names are guaranteed null terminated.
   * The maximum length of a name in a directory is MAXNAMLEN.
   *
   * The macro DIRSIZ(dp) gives the amount of space required to represent
   * a directory entry.  Free space in a directory is represented by
   * entries which have dp->d_reclen > DIRSIZ(dp).  All DIRBLKSIZ bytes
   * in a directory block are claimed by the directory entries.  This
   * usually results in the last entry in a directory having a large
   * dp->d_reclen.  When entries are deleted from a directory, the
   * space is returned to the previous entry in the same directory
   * block by increasing its dp->d_reclen.  If the first entry of
   * a directory block is free, then its dp->d_ino is set to 0.
   * Entries other than the first in a directory do not normally have
   * dp->d_ino set to 0.
   */

  #define DIRBLKSIZ 512
        #define MAXNAMLEN 63

  /*
   * The DIRSIZ macro gives the minimum record length which will hold
   * the directory entry.  This requires the amount of space in struct direct
   * without the d_name field, plus enough space for the name with a
   * terminating null byte (dp->d_namlen+1), rounded up to a 4 byte boundary.
   */
  #undef DIRSIZ
  #define DIRSIZ(dp) \


Printed      May 15, 1985                    1
DIR(5)               UNIX Programmer's Manual         DIR(5)
```

```
((((sizeof (struct direct) - (MAXNAMLEN+1)) + (dp)->d_namlen+1) + 3) &~ 3)

   struct  direct {
        ino_t    d_ino;
        short    d_reclen;
        short    d_namlen;
        char     d_name[MAXNAMLEN + 1];
        /* typically shorter */
   };

   struct _dirdesc {
        int      dd_fd;
        long     dd_loc;
        long     dd_size;
        char     dd_buf[DIRBLKSIZ];
   };
```

By convention, the first two entries in each directory are for `.' and `..'. The first is an entry for the directory itself. The second is for the parent directory. The meaning of `..' is modified for the root directory of the master file system ("/"), where `..' has the same meaning as `.'.

SEE ALSO

fs(5)

BUGS

The 63 character MAXNAMLEN value is shorter than the 255 characters allowed by 4BSD. This could lead to file name portability problems in unusual circumstances.

The disk format of directories is only slightly different from the 4BSD directory format, the inode number is of type ino_t rather than u_long to reduce the amount of 32 bit arithmetic in the kernel.

NAME

disktab - disk description file

SYNOPSIS

#include <disktab.h>

DESCRIPTION

Disktab is a simple date base which describes disk geometries and disk partition characteristics. The format is patterned after the termcap(5) terminal data base. Entries in disktab consist of a number of `:' separated fields. The first entry for each disk gives the names which are known for the disk, separated by `|' characters. The last name given should be a long name fully identifying the disk.

The following list indicates the normal values stored for each disk entry.

Name	Type	Description
ns	num	Number of sectors per track
nt	num	Number of tracks per cylinder
nc	num	Total number of cylinders on the disk
b0	str	Filename of block zero primary bootstrap for device
ba	num	Block size for partition `a' (bytes)
bd	num	Block size for partition `d' (bytes)
be	num	Block size for partition `e' (bytes)
bf	num	Block size for partition `f' (bytes)
bg	num	Block size for partition `g' (bytes)
bh	num	Block size for partition `h' (bytes)
fa	num	Fragment size for partition `a' (bytes)
fd	num	Fragment size for partition `d' (bytes)
fe	num	Fragment size for partition `e' (bytes)
ff	num	Fragment size for partition `f' (bytes)
fg	num	Fragment size for partition `g' (bytes)
fh	num	Fragment size for partition `h' (bytes)
pa	num	Size of partition `a' in sectors
pb	num	Size of partition `b' in sectors
pc	num	Size of partition `c' in sectors
pd	num	Size of partition `d' in sectors
pe	num	Size of partition `e' in sectors
pf	num	Size of partition `f' in sectors
pg	num	Size of partition `g' in sectors
ph	num	Size of partition `h' in sectors
se	num	Sector size in bytes (default 512)
sf	bool	supports bad144-style bad sector forwarding
so	bool	partition offsets in sectors
ty	str	Type of disk (e.g. removable, winchester)

Disktab entries may be automatically generated with the diskpart program.

FILES
     /etc/disktab

SEE ALSO
     newfs(8), diskpart(8), getdiskbyname(3)

BUGS
     This file shouldn't exist, the information should be stored
     on each disk pack.

NAME

dtab - automatic system configuration device table (2BSD)

DESCRIPTION

The dtab file contains a list of the devices that the autoconfig(8) program will attempt to attach to the system.

Each line describes one device which may be present on the system; additional devices of the same type require additional lines. The line contains at least six fields separated by white space. Blank lines and comments can appear anywhere in the file except between fields; comments are delimited by `#' and new line. The fields are (in order):

dev — The device name, usually two characters

unit — Either a unit number or a `?' indicating automatic selection of unit numbers.

addr — The address of the device's first register, as a 16-bit octal number.

vector — The interrupt vector, in octal.

BR — The priority at which the device interrupts.

handler(s) — One or more routine names with which the interrupt vector is filled.

EXAMPLE

```
#                Device Configuration Table
#        Clock and console are pre-configured and should not be listed here
#
# Dev#  Addr  Vector    Br Handler(s)     # Comment
lp   ?  177514  200      4  lpio          # lp-11 line printer
ht   0  172440  224      5  htintr        # tu 16 massbus tape
xp   ?  176700  254      5  xpio          # xp driver
dm   ?  170500  310      4  dmin          # dm11 dh modem control
dh   ?  160020  320      5  dhin dhou     # dh11 terminal mux
dz   ?  160110  330      5  dzin dzdma    # dz11 terminal mux
```

FILES

/etc/dtab — device table

SEE ALSO

autoconfig(8)

BUGS

/Etc/dtab is unique to the PDP-11 and 2BSD.

NAME

dump, ddate - incremental dump format

SYNOPSIS

```
#include <sys/types.h>
#include <sys/ino.h>
#include <dumprestor.h>
```

DESCRIPTION

Tapes used by dump and restor(8) contain:

- a header record
- two groups of bit map records
- a group of records describing directories
- a group of records describing files

The format of the header record and of the first record of each description as given in the include file <dumprestor.h> is:

```
#if  UCB_NKB == 1
#define NTREC    10
#endif
#ifndef          UCB_NKB
#define NTREC    20
#endif
#define MLEN     16
#define MSIZ     4096

#define TS_TAPE  1
#define TS_INODE 2
#define TS_BITS  3
#define TS_ADDR  4
#define TS_END   5
#define TS_CLRI  6
#define MAGIC    (int)60011
#define CHECKSUM (int)84446
struct       spcl
{
   int       c_type;
   time_t    c_date;
   time_t    c_ddate;
   int       c_volume;
   daddr_t   c_tapea;
   ino_t     c_inumber;
   int       c_magic;
   int       c_checksum;
   struct    dinodec_dinode;
   int       c_count;
   char      c_addr[BSIZE];
} spcl;
```

```
struct  idates
{
   Char     id_name[16];
   Char     id_incno;
   time_t   id_ddate;
};
```

NTREC is the number of BSIZE (sys/param.h) byte records in a physical tape block. MLEN is the number of bits in a bit map word. MSIZ is the number of bit map words.

The TS_ entries are used in the c_type field to indicate what sort of header this is. The types and their meanings are as follows:

TS_TAPE Tape volume label

TS_INODE
: A file or directory follows. The c_dinode field is a copy of the disk inode and contains bits telling what sort of file this is.

TS_BITS A bit map follows. This bit map has a one bit for each inode that was dumped.

TS_ADDR A subrecord of a file description. See c_addr below.

TS_END End of tape record.

TS_CLRI A bit map follows. This bit map contains a zero bit for all inodes that were empty on the file system when dumped.

MAGIC All header records have this number in c_magic.

CHECKSUM
: Header records checksum to this value.

The fields of the header structure are as follows:

c_type The type of the header.

c_date The date the dump was taken.

c_ddate The date the file system was dumped from.

c_volume The current volume number of the dump.

c_tapea The current number of this (512-byte) record.

c_inumber
: The number of the inode being dumped if this is of type TS_INODE.

c_magic This contains the value MAGIC above, truncated as needed.

c_checksum
: This contains whatever value is needed to make the record sum to CHECKSUM.

c_dinode This is a copy of the inode as it appears on the file system; see filsys(5).

c_count The count of characters in c_addr.

c_addr An array of characters describing the blocks of the dumped file. A character is zero if the block

associated with that character was not present on the file system, otherwise the character is non-zero. If the block was not present on the file system, no block was dumped; the block will be restored as a hole in the file. If there is not sufficient space in this record to describe all of the blocks in a file, TS_ADDR records will be scattered through the file, each one picking up where the last left off.

Each volume except the last ends with a tapemark (read as an end of file). The last volume ends with a TS_END record and then the tapemark.

The structure idates describes an entry of the file /etc/ddate where dump history is kept. The fields of the structure are:

id_name The dumped file system is `/dev/id_nam'.
id_incno The level number of the dump tape; see dump(8).
id_ddate The date of the incremental dump in system format see types(5).

FILES

/etc/ddate

SEE ALSO

filsys(5), types(5), dump(8), dumpdir(8), restor(8)

NAME

fs, inode - format of file system volume (2BSD)

SYNOPSIS

```
#include <sys/types.h>
#include <sys/fs.h>
#include <sys/inode.h>
```

DESCRIPTION

Every file system storage volume (e.g. disk) has a common format for certain vital information. Every such volume is divided into a certain number of blocks. The block size is DEV_BSIZE bytes; specified in <sys/param.h> - currently 1024.

Each disk drive contains some number of file systems each laid out on a contiguous partition of the disk. A file system consists of a boot block, followed by a super block, followed by an inode area, followed by a data block area which takes up the remainder of the disk partition. The layout of the super block as defined in <sys/fs.h> is:

```
#define MAXMNTLEN 12

/*
 * Structure of the super-block
 */
struct fs
{
    u_short fs_isize;          /* first block after i-list */
    daddr_t fs_fsize;          /* size in blocks of entire volume */
    short   fs_nfree;          /* number of addresses in fs_free */
    daddr_t fs_free[NICFREE];  /* free block list */
    short   fs_ninode;         /* number of inodes in fs_inode */
    ino_t   fs_inode[NICINOD]; /* free inode list */
    char    fs_flock;          /* lock during free list manipulation */
    char    fs_ilock;          /* lock during i-list manipulation */
    char    fs_fmod;           /* super block modified flag */
    char    fs_ronly;          /* mounted read-only flag */
    time_t  fs_time;           /* last super block update */
    daddr_t fs_tfree;          /* total free blocks */
    ino_t   fs_tinode;         /* total free inodes */
    short   fs_step;           /* optimal step in free list pattern */
    short   fs_cyl;            /* number of blocks per pattern */
    char    fs_fsmnt[MAXMNTLEN]; /* ordinary file mounted on */
    ino_t   fs_lasti;          /* start place for circular search */
    ino_t   fs_nbehind;         /* est # free inodes before s_lasti */
    u_short fs_flags;          /* mount time flags */
};
```

File system: A file system is described by its super-block. Block 0 of each file system partition is unused and is

available to contain a bootstrap program, pack label, or other information. Block 1 (SUPERB) is the super block. The inode area starts immediately after the super-block, in block 2. Fs_isize is the address of the first block after the inode area. Thus the inode area is fs_isize-2 blocks long. Fs_fsize is the address of the first block not potentially available for allocation to a file. Thus the data block area is fs_fsize - fs_isize blocks long.

Super block: The path name on which the file system is mounted is maintained in fs_fsmnt. Fs_flock, fs_ilock, fs_fmod, fs_ronly and fs_flags are flags maintained in the in core copy of the super block while its file system is mounted and their values on disk are immaterial. Fs_fmod is used as a flag to indicate that the super-block has changed and should be copied to the disk during the next periodic update of file system information. Fs_ronly is a write-protection indicator. It is a copy of the mount flags fs_flags anded with MNT_RDONLY(see/sys/h/mount.h).

Fs_time is the last time the super-block of the file system was changed. During a reboot, the fs_time of the super-block for the root file system is used to set the system's idea of the time.

Inode: The inode is the focus of all file activity in the UNIX file system. There is a unique inode allocated for each active file, each current directory, each mounted-on file, text file, and the root. An inode is `named' by its device/i-number pair.

Inodes are 64 bytes long, so 16 of them fit into a block if DEV_BSIZE is 1024. The root inode is the root of the file system. Inode 0 can't be used for normal purposes and historically bad blocks were linked to inode 1, thus the root inode is 2 (inode 1 is no longer used for this purpose, however numerous dump tapes make this assumption, so we are stuck with it). No other i-number has a built-in meaning.

The format of an inode as given in <sys/inode.h> is:

```
/*
 * Inode structure as it appears on
 * a disk block.
 */
struct dinode {
     u_short di_mode;    /* mode and type of file */
     short   di_nlink;   /* number of links to file */
     uid_t   di_uid;     /* owner's user id */
     gid_t   di_gid;     /* owner's group id */
     off_t   di_size;    /* number of bytes in file */
     daddr_t di_addr[7]; /* 7 block addresses 4 bytes each */
```

```
        u_short di_reserved[5];/* pad of 10 to make total size 64 */
        u_short di_flags;
        time_t  di_atime;   /* time last accessed */
        time_t  di_mtime;   /* time last modified */
        time_t  di_ctime;   /* time created */
};

/*
 * 28 of the di_addr address bytes are used; 7 addresses of 4
 * bytes each: 4 direct (4Kb directly accessible) and 3 indirect.
 */
#define NADDR  7

/* modes */

#define IFMT   0170000 /* type of file */
#define IFCHR  0020000 /* character special */
#define IFDIR  0040000 /* directory */
#define IFBLK  0060000 /* block special */
#define IFREG  0100000 /* regular */
#define IFLNK  0120000 /* symbolic link */
#define IFSOCK 0140000 /* socket */
#define ISUID  04000   /* set user id on execution */
#define ISGID  02000   /* set group id on execution */
#define ISVTX  01000   /* save swapped text even after use */
#define IREAD  0400    /* read, write, execute permissions */
#define IWRITE 0200
#define IEXEC  0100
```

Di_mode identifies the type of file the inode represents; it is encoded identically to the st_mode field of stat(2). Di_nlink is the number of directory entries (links) that refer to this inode. Di_uid and di_gid are the owner's user and group IDs. Di_size is the number of bytes in the file. Di_atime and di_mtime are the times of last access and modification of the file contents (read, write or create); Di_ctime records the time of last modification to the inode or to the file, and is used to determine whether it should be dumped by dump(8).

Special files are recognized by their modes. A block-type special file is one which can potentially be mounted as a file system; a character-type special file cannot, though it is not necessarily character-oriented. For special files, the first two bytes of the di_addr field are occupied by the device code (see types(5)). The device codes of block and character special files overlap.

Disk addresses of plain files and directories are kept in the array di_addr. For a DEV_BSIZE of 1K bytes, 7 addresses are kept in di_addr using 28 of the 40 bytes. The first 4 addresses specify device blocks directly. The last 3

addresses are singly, doubly and triply indirect and point to blocks containing 256 further block pointers. There are 3 block addresses reserved as a pad to bring the total size of an inode to 64 bytes. All block addresses are of type daddr_t (see types(5)).

For block b in a file to exist, it is not necessary that all blocks less than b exist. A zero block number indicates that the corresponding block has never been allocated. Such a missing block reads as if it contained all zero bytes.

Free block list: The free data block list for each volume is maintained as follows. Fs_free[1], ... , fs_free[fs_nfree-1], contain up to NICFREE free block numbers (NICFREE is a configuration constant defined in <sys/param.h>). Fs_free[0] is the block address of the head of a chain of blocks constituting the free list. The layout of each block of the free chain as defined in <sys/fs.h> is:

```
struct fblk
{
     short   df_nfree;        /* number of addresses in df_free */
     daddr_t df_free[NICFREE]; /* free block list */
};
```

The fields df_nfree and df_free in a free block are used exactly like fs_nfree and fs_free in the super block.

The algorithm used to allocate a block is: decrement fs_nfree, and the new block number is fs_free[fs_nfree]. If the new block address is 0, there are no blocks left, so give an error. If fs_nfree became 0, read the new block into fs_nfree and fs_free.

To free a block: check if fs_nfree is NICFREE; if so, copy fs_nfree and the fs_free array into the newly freed block, write it out, and set fs_nfree to 0. In any event set fs_free[fs_nfree] to the freed block's address and increment fs_nfree.

Fs_isize and fs_fsize are used by the system to check for bad block addresses; if an `impossible' block address is allocated from or returned to the free list, a diagnostic is written on the console. Moreover, the free array is cleared, to prevent further allocation from a presumably corrupted free list.

Fs_step and fs_cyl determine the block interleaving of files for fastest access; traditionally these were referred to as s_m and s_n respectively. Fs_step is the distance between successive blocks and fs_cyl is the number of blocks before the pattern repeats. A file system's interleaving factors

are determined when it is first created by mkfs(8). Mkfs lays out the initial free list with these parameters and fsck(8) can be used to rebuild the free list optimally (and assign new interleaving factors if necessary).

Free inode list: Fs_ninode is the number of free inode numbers in the fs_inode array.

To allocate an inode: if fs_ninode is greater than 0, decrement it and return fs_inode[fs_ninode]. If it was 0, read through the inode area and place the numbers of all free inodes (up to NICINOD) into the fs_inode array, then try again. If a search for free inodes is necessary, the search will start at the beginning of the inode area if fs_nbehind >= 4 x NICINOD, otherwise starting at fs_lasti and continuing at the beginning of the inode area if NICINOD free inodes aren't found when the end of the inode area is reached. When a search completes the i-number of the first inode of the last block scanned in the search is left in fs_lasti.

To free an inode, provided fs_ninode is less than NICINODE, place its number into fs_inode[fs_ninode] and increment fs_ninode. If fs_ninode is already NICINODE, don't bother to enter the freed inode into any table (fs_inode is only to speed up the allocation process; the information as to whether the inode is really free or not is maintained in the inode itself). If the i-number of the freed inode is less than fs_lasti increment fs_nbehind.

SEE ALSO

stat(2), dir(5), types(5), dcheck(8), fsck(8), icheck(8), mkfs(8), mount(8)

BUGS

It isn't the 4BSD fast file system. The 2BSD file system is a direct descendent of the V7 file system and exists little changed from that ancestor. There are many performance holes in the file system.

Some changes from the original V7 file system have resulted in better performance: The larger block size (1Kb as opposed to the 512 byte block size of V7) cuts the average number of system calls necessary to access a file by a factor of two; the smaller (in core) inodes allowed by the smaller number of direct links kept in inodes saves valuable kernel data space allowing the kernel buffer cache to be made larger while sacrificing only 1Kb of direct file accessing; and starting free inode searches at the position the last search ended cuts the time to gather free inodes significantly.

However, the separation of inodes and data blocks into completely different areas of the disk, the handling of the free list, the lack of any file allocation layout policy encouraging locality such as that found in the 4BSD file system and the still too small block size often leads to extremely poor performance.

The separation of inodes and data blocks in the file system means that to access a file a seek will have to be made to the beginning of the disk partition containing the file system followed another to the the actual data blocks of the file (often quite distant from the inode area).

The free list which is laid out at file system creation for optimal file block allocation, becomes scrambled over time on an active file system. This process is slowed down by the kernel which always frees blocks from unlink'ed or truncated files in reverse order thereby maintaining strings of optimally laid out free blocks in the free list. Eventually, however, since both freed and allocated blocks use the head of the free list, it's possible (and quite probable) to have most of the free list laid out optimally with the first portion totally scrambled. As a trade off, a file system's free list may be rebuilt fairly frequently via icheck -s or fsck -s and most blocks allocated will be localized as close to the the inode area as possible. Because of this problem, files are sometimes scattered across a file system generating an unpleasant amount of disk arm movement. A nasty oscillation also occurs in the free block list when fs_nfree hovers around NICFREE and 0 causing the free array to be constantly written out and read back in as blocks are freed and allocated.

For a more in depth analysis of the 2BSD file system, its shortcomings, and a description of the changes made for the 4BSD file system see "A Fast File System for UNIX" by M. McKusick; W. Joy; S. Leffler; and R. Fabry.

NAME

fstab - static information about the filesystems

SYNOPSIS

#include <fstab.h>

DESCRIPTION

The file fstab contains descriptive information about the various file systems. fstab is only read by programs, and not written; it is the duty of the system administrator to properly create and maintain this file. Each filesystem is described on a separate line; fields on each line are separated by tabs or spaces. The order of records in fstab is important because fsck(8), mount(8), and umount(8) sequentially iterate through fstab doing their thing.

The first field, fs_spec, describes the block special device or remote filesystem to be mounted. For filesystems of type ufs, the special file name is the block special file name, and not the character special file name. If a program needs the character special file name, the program must create it by appending a ``r'' after the last ``/'' in the special file name.

The second field, fs_file, describes the mount point for the filesystem. For swap partitions, this field should be specified as ``none''.

The third field, fs_vfstype, describes the type of the filesystem. The system currently supports only two types of filesystems:

ufs a local UNIX filesystem

swap a disk partition to be used for swapping

The fourth field, fs_mntops, describes the mount options associated with the filesystem. It is formatted as a comma separated list of options. It contains at least the type of mount (see fs_type below) plus any additional options appropriate to the filesystem type.

If the option ``quotas'' is specified, the filesystem is automatically processed by the quotacheck(8) command, and user disk quotas are enabled with quotaon(8). Filesystem quotas are maintained in the file named quotas located at the root of the associated filesystem. This restriction on the location of the quotas file is needlessly imposed by the kernel but may be lifted in the future. Thus, if the user quota file for /tmp is stored in /var/quotas/tmp.user, this location can be specified as:

```
     quotas=/var/quotas/tmp.user
```

The type of the mount is extracted from the fs_mntops field and stored separately in the fs_type field (it is not deleted from the fs_mntops field). If fs_type is ``rw'' or ``ro'' then the filesystem whose name is given in the fs_file field is normally mounted read-write or read-only on the specified special file. If fs_type is ``sw'' then the special file is made available as a piece of swap space by the swapon(8) command at the end of the system reboot procedure. The fields other than fs_spec and fs_type are unused. If fs_type is specified as ``xx'' the entry is ignored. This is useful to show disk partitions which are currently unused.

The fifth field, fs_freq, is used for these filesystems by the dump(8) command to determine which filesystems need to be dumped. If the fifth field is not present, a value of zero is returned and dump(8) will assume that the filesystem does not need to be dumped.

The sixth field, fs_passno, is used by the fsck(8) program to determine the order in which filesystem checks are done at reboot time. The root filesystem should be specified with a fs_passno of 1, and other filesystems should have a fs_passno of 2. Filesystems within a drive will be checked sequentially, but filesystems on different drives will be checked at the same time to utilize parallelism available in the hardware. If the sixth field is not present or zero, a value of zero is returned and fsck(8) will assume that the filesystem does not need to be checked.

```
#define   FSTAB_RW  "rw" /* read-write device */
#define   FSTAB_RO  "ro" /* read-only device */
#define   FSTAB_SW  "sw" /* swap device */
#define   FSTAB_XX  "xx" /* ignore totally */

struct fstab {
   char *fs_spec;       /* block special device name */
   char *fs_file;       /* filesystem path prefix */
   char *fs_vfstype;    /* type of filesystem */
   char *fs_mntops;     /* comma separated mount options */
   char *fs_type;       /* rw, ro, sw, or xx */
   int  fs_freq;        /* dump frequency, in days */
   int  fs_passno;      /* pass number on parallel dump */
};
```

The proper way to read records from fstab is to use the routines getfsent(3), getfsspec(3), getfstype(3), and getfsfile(3).

FILES
     /etc/fstab   The file fstab resides in /etc.

SEE ALSO
     getfsent(3)

HISTORY
     The fstab file format appeared in 4.0BSD.

NAME

gettytab - terminal configuration data base

SYNOPSIS

/etc/gettytab

DESCRIPTION

Gettytab is a simplified version of the termcap(5) data base used to describe terminal lines. The initial terminal login process getty(8) accesses the gettytab file each time it starts, allowing simpler reconfiguration of terminal characteristics. Each entry in the data base is used to describe one class of terminals.

There is a default terminal class, default, that is used to set global defaults for all other classes. (That is, the default entry is read, then the entry for the class required is used to override particular settings.)

CAPABILITIES

Refer to termcap(5) for a description of the file layout. The default column below lists defaults obtained if there is no entry in the table obtained, nor one in the special default table.

Name	Type	Default	Description
ap	bool	false	terminal uses any parity
bk	str	0377	alternate end of line character (input break)
cb	bool	false	use crt backspace mode
ce	bool	false	use crt erase algorithm
ck	bool	false	use crt kill algorithm
cl	str	NULL	screen clear sequence
co	bool	false	console - add \n after login prompt
ds	str	^Y	delayed suspend character
dx	bool	false	set DECCTLQ
ec	bool	false	leave echo OFF
ep	bool	false	terminal uses even parity
er	str	^?	erase character
et	str	^D	end of text (EOF) character
ev	str	NULL	initial enviroment
f0	num	unused	tty mode flags to write messages
f1	num	unused	tty mode flags to read login name
f2	num	unused	tty mode flags to leave terminal as
fl	str	^O	output flush character
hc	bool	false	do NOT hangup line on last close
he	str	NULL	hostname editing string
hf	bool	false	enable hardware (rts/cts) flow control
hn	str	hostname	hostname
ht	bool	false	terminal has real tabs
ig	bool	false	ignore garbage characters in login name
im	str	NULL	initial (banner) message
in	str	^C	interrupt character

is	num	unused	input speed
kl	str	^U	kill character
lc	bool	false	terminal has lower case
lm	str	login:	login prompt
ln	str	^V	``literal next'' character
lo	str	/bin/login	program to exec when name obtained
nl	bool	false	terminal has (or might have) a newline character
nx	str	default	next table (for auto speed selection)
op	bool	false	terminal uses odd parity
os	num	unused	output speed
pc	str	\0	pad character
pe	bool	false	use printer (hard copy) erase algorithm
pf	num	0	delay between first prompt and following flush (seconds)
ps	bool	false	line connected to a MICOM port selector
qu	str	^\	quit character
rp	str	^R	line retype character
rw	bool	false	do NOT use raw for input, use cbreak
sp	num	unused	line speed (input and output)
su	str	^Z	suspend character
tc	str	none	table continuation
to	num	0	timeout (seconds)
tt	str	NULL	terminal type (for enviroment)
ub	bool	false	do unbuffered output (of prompts etc)
we	str	^W	word erase character
xc	bool	false	do NOT echo control chars as ^X
xf	str	^S	XOFF (stop output) character
xn	str	^Q	XON (start output) character

If no line speed is specified, speed will not be altered from that which prevails when getty is entered. Specifying an input or output speed will override line speed for stated direction only.

Terminal modes to be used for the output of the message, for input of the login name, and to leave the terminal set as upon completion, are derived from the boolean flags specified. If the derivation should prove inadequate, any (or all) of these three may be overriden with one of the f0, f1, or f2 numeric specifications, which can be used to specify (usually in octal, with a leading '0') the exact values of the flags. Local (new tty) flags are set in the top 16 bits of this (32 bit) value.

Should getty receive a null character (presumed to indicate a line break) it will restart using the table indicated by the nx entry. If there is none, it will re-use its original table.

The cl screen clear string may be preceded by a (decimal) number of milliseconds of delay required (a la termcap). This delay is simulated by repeated use of the pad character pc.

The initial message, and login message, im and lm may include the character sequence %h or %t to obtain the hostname or tty name respectively. (%% obtains a single '%' character.) The hostname is normally obtained from the system, but may be set by the hn table entry. In either case it may be edited with he. The he string is a sequence of characters, each character that is neither '@' nor '#' is copied into the final hostname. A '@' in the he string, causes one character from the real hostname to be copied to the final hostname. A '#' in the he string, causes the next character of the real hostname to be skipped. Surplus '@' and '#' characters are ignored.

When getty execs the login process, given in the lo string (usually "/bin/login"), it will have set the enviroment to include the terminal type, as indicated by the tt string (if it exists). The ev string, can be used to enter additional data into the environment. It is a list of comma separated strings, each of which will presumably be of the form name=value.

If a non-zero timeout is specified, with to, then getty will exit within the indicated number of seconds, either having received a login name and passed control to login, or having received an alarm signal, and exited. This may be useful to hangup dial in lines.

Output from getty is even parity unless op is specified. Op may be specified with ap to allow any parity on input, but generate odd parity output. Note: this only applies while getty is being run, terminal driver limitations prevent a more complete implementation. Getty does not check parity of input characters in RAW mode.

SEE ALSO

login(1), termcap(5), getty(8).

BUGS

The special characters (erase, kill, etc.) are reset to system defaults by login(1). In all cases, '#' or '^H' typed in a login name will be treated as an erase character, and '@' will be treated as a kill character.

The delay stuff is a real crock. It has been removed from the system entirely. The he capability is stupid.

Termcap format is horrid, something more rational should have been chosen.

NAME

group - group file

DESCRIPTION

Group contains for each group the following information:

group name
encrypted password
numerical group ID
a comma separated list of all users allowed in the group

This is an ASCII file. The fields are separated by colons; Each group is separated from the next by a new-line. If the password field is null, no password is demanded.

This file resides in directory /etc. Because of the encrypted passwords, it can and does have general read permission and can be used, for example, to map numerical group ID's to names.

FILES

/etc/group

SEE ALSO

setgroups(2), initgroups(3X), crypt(3), passwd(1), passwd(5)

BUGS

The passwd(1) command won't change the passwords.

NAME

hosts - host name data base

DESCRIPTION

The hosts file contains information regarding the known hosts on the network. For each host a single line should be present with the following information:

official host name
Internet address
aliases

Items are separated by any number of blanks and/or tab characters. A ``#'' indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines which search the file.

When using the name server named(8), this file provides a backup when the name server is not running. For the name server, it is suggested that only a few addresses be included in this file. These include address for the local interfaces that ifconfig(8C) needs at boot time and a few machines on the local network.

This file may be created from the official host data base maintained at the Network Information Control Center (NIC), though local changes may be required to bring it up to date regarding unofficial aliases and/or unknown hosts. As the data base maintained at NIC is incomplete, use of the name server is recommend for sites on the DARPA Internet.

Network addresses are specified in the conventional ``.'' notation using the inet_addr() routine from the Internet address manipulation library, inet(3N). Host names may contain any printable character other than a field delimiter, newline, or comment character.

FILES

/etc/hosts

SEE ALSO

gethostbyname(3N), ifconfig(8C), named(8)
Name Server Operations Guide for BIND

NAME

L-devices - UUCP device description file

DESCRIPTION

The L-devices file is consulted by the UUCP daemon uucico(8) under the direction of L.sys(5) for information on the devices that it may use. Each line describes exactly one device.

A line in L-devices has the form:

Caller Device Call_Unit Class Dialer [Expect Send]....

Each item can be separated by any number of blanks or tabs. Lines beginning with a `#' character are comments; long lines can be continued by appending a `\' character to the end of the line.

Caller denotes the type of connection, and must be one of the following:

ACU Automatic call unit, e.g., autodialing modems such as the Hayes Smartmodem 1200 or Novation ``Smart Cat''.

DIR Direct connect; hardwired line (usually RS-232) to a remote system.

DK AT&T Datakit.

MICOM Micom Terminal switch.

PAD X.25 PAD connection.

PCP GTE Telenet PC Pursuit.

SYTEK Sytek high-speed dedicated modem port connection.

TCP Berkeley TCP/IP or 3Com UNET connection. These are mutually exclusive. Note that listing TCP connections in L-devices is superfluous; uucico does not even bother to look here since it has all the information it needs in L.sys(5).

Device is a device file in /dev/ that is opened to use the device. The device file must be owned by UUCP, with access modes of 0600 or better. (See chmod(2)).

Call_Unit is an optional second device file name. True automatic call units use a separate device file for data and for dialing; the Device field specifies the data port, while the Call_unit field specifies the dialing port. If the

Call_unit field is unused, it must not be left empty. Insert a dummy entry as a placeholder, such as ``0'' or ``unused.''

Class is an integer number that specifies the line baud (for dialers and direct lines) or the port number (for network connections).

The Class may be preceded by a non-numeric prefix. This is to differentiate among devices that have identical Caller and baud, but are distinctly different. For example, ``1200'' could refer to all Bell 212-compatible modems, ``V1200'' to Racal-Vadic modems, and ``C1200'' to CCITT modems, all at 1200 baud. Similarly, ``W1200'' could denote long distance lines, while ``L1200'' could refer to local phone lines.

Dialer applies only to ACU devices. This is the "brand" or type of the ACU or modem.

DF02 DEC DF02 or DF03 modems.

DF112 Dec DF112 modems. Use a Dialer field of DF112T to use tone dialing, or DF112P for pulse dialing.

att AT&T 2224 2400 baud modem.

cds224 Concord Data Systems 224 2400 baud modem.

dn11 DEC DN11 Unibus dialer.

hayes Hayes Smartmodem 1200 and compatible autodialing modems. Use a Dialer field of hayestone to use tone dialing, or hayespulse for pulse dialing. It is also permissible to include the letters `T' and `P' in the phone number (in L.sys) to change to tone or pulse midway through dialing. (Note that a leading `T' or `P' will be interpreted as a dialcode!)

hayes2400
Hayes Smartmodem 2400 and compatible modems. Use a Dialer field of hayes2400tone to use tone dialing, or hayes2400pulse for pulse dialing.

novation
Novation ``Smart Cat'' autodialing modem.

penril Penril Corp ``Hayes compatible'' modems (they really aren't or they would use the hayes entry.)

rvmacs Racal-Vadic 820 dialer with 831 adapter in a MACS configuration.

va212 Racal-Vadic 212 autodialing modem.

va811s Racal-Vadic 811s dialer with 831 adapter.

va820 Racal-Vadic 820 dialer with 831 adapter.

vadic Racal-Vadic 3450 and 3451 series autodialing modems.

ventel Ventel 212+ autodialing modem.

vmacs Racal-Vadic 811 dialer with 831 adapter in a MACS configuration.

Expect/Send is an optional Expect/Send script for getting through a smart port selector, or for issuing special commands to the modem. The syntax is identical to that of the Expect/Send script of L.sys. The difference is that the L-devices script is used before the connection is made, while the L.sys script is used after.

FILES

/etc/uucp/L-devices
/etc/uucp/UUAIDS/L-devices L-devices example

SEE ALSO

uucp(1), uux(1), L.sys(5), uucico(8)

NAME

L-dialcodes - UUCP phone number index file

DESCRIPTION

The L-dialcodes file defines the mapping of strings from the phone number field of L.sys(5) to actual phone numbers.

Each line in L-dialcodes has the form:

```
alpha_string   phone_number
```

The two items can be separated by any number of blanks or tabs. Lines beginning with a `#' character are comments.

A phone number in L.sys can be preceded by an arbitrary alphabetic character string; the string is matched against the list of alpha_strings in L-dialcodes. If a match is found, phone_number is substituted for it. If no match is found, the string is discarded.

L-dialcodes is commonly used either of two ways:

(1) The alphabetic strings are used as prefixes to denote area codes, zones, and other commonly used sequences. For example, if L-dialcodes included the following lines:

```
chi  1312
mv   1415
```

In L.sys you could enter:

```
chivax Any ACU 1200 chi5551234  ogin:--ogin: nuucp
mvpyr  Any ACU 1200 mv5556001   ogin:--ogin: Uuucp
```

instead of

```
chivax Any ACU 1200 13125551234  ogin:--ogin: nuucp
mvpyr  Any ACU 1200 14155556001  ogin:--ogin: Uuucp
```

(2) All phone numbers are placed in L-dialcodes, one for each remote site. L.sys then refers to these by name. For example, if L-dialcodes contains the following lines:

```
chivax    13125551234
mvpyr     14155556601
```

then L.sys could have:

```
chivax Any ACU 1200 chivax  ogin:--ogin: nuucp
mvpyr  Any ACU 1200 mvpyr   ogin:--ogin: Uuucp
```

This scheme allows a site administrator to give users read access to the table of phone numbers, while still protecting the login/password sequences in L.sys.

FILES

/etc/uucp/L-dialcodes
/etc/uucp/UUAIDS/L-dialcodes L-dialcodes example

SEE ALSO

uucp(1), uux(1), L.sys(5), uucico(8).

NAME

L.aliases - UUCP hostname alias file

DESCRIPTION

The L.aliases file defines mapping (aliasing) of system names for uucp. This is intended for compensating for systems that have changed names, or do not provide their entire machine name (like most USG systems). It is also useful when a machine's name is not obvious or commonly misspelled.

Each line in L.aliases is of the form:

real_name alias_name

Any amount of whitespace may separate the two items. Lines beginning with a `#' character are comments.

All occurrences of alias_name are mapped to real_name by uucico(8), uucp(1), and uux(1). The mapping occurs regardless of whether the name was typed in by a user or provided by a remote site. An exception is the -s option of uucico; only the site's real hostname (the name in L.sys(5)) will be accepted there.

Aliased system names should not be placed in L.sys; they will not be used.

FILES

/etc/uucp/L.aliases /etc/uucp/UUAIDS/L.aliases L.aliases example

SEE ALSO

uucp(1), uux(1), L.sys(5), uucico(8)

NAME

L.cmds - UUCP remote command permissions file

DESCRIPTION

The L.cmds file contains a list of commands, one per line, that are permitted for remote execution via uux(1).

The default search path is /bin:/usr/bin:/usr/ucb. To change the path, include anywhere in the file a line of the form:

```
PATH=/bin:/usr/bin:/usr/ucb
```

Normally, an acknowledgment is mailed back to the requesting site after the command completes. If a command name is suffixed with ,Error, then an acknowledgment will be mailed only if the command fails. If the command is suffixed with ,No, then no acknowledgment will ever be sent. (These correspond with the -z and -n options of uux, respectively.)

For most sites, L.cmds should only include the lines:

```
rmail
ruusend
```

News sites should add:

```
PATH=/bin:/usr/bin:/usr/ucb:/usr/new
rnews,Error
```

While file names supplied as arguments to uux commands will be checked against the list of accessible directory trees in USERFILE(5), this check can be easily circumvented and should not be depended upon. In other words, it is unwise to include any commands in L.cmds that accept local file names. In particular, sh(1) and csh(1) are extreme risks.

It is common (but hazardous) to include uucp(1) in L.cmds; see the NOTES section of USERFILE.

FILES

/etc/uucp/L.cmds
/etc/uucp/UUAIDS/L.cmds L.cmds example.

SEE ALSO

uucp(1), uux(1), USERFILE(5), uucico(8), uuxqt(8)

NAME

L.sys - UUCP remote host description file

DESCRIPTION

The L.sys file is consulted by the UUCP daemon uucico(8) for information on remote systems. L.sys includes the system name, appropriate times to call, phone numbers, and a login and password for the remote system. L.sys is thus a privileged file, owned by the UUCP Administrator; it is accessible only to the Administrator and to the superuser.

Each line in L.sys describes one connection to one remote host, and has the form:

System Times Caller Class Device/Phone_Number [Expect Send]....

Fields can be separated by any number of blanks or tabs. Lines beginning with a `#' character are comments; long lines can be continued by appending a `\' character to the end of the line.

The first five fields (System through Device/Phone_Number) specify the hardware mechanism that is necessary to make a connection to a remote host, such as a modem or network. Uucico searches from the top down through L.sys to find the desired System; it then opens the L-devices(5) file and searches for the first available device with the same Caller, Class, and (possibly) Device. (``Available'' means that the device is ready and not being used for something else.) Uucico attempts a connection using that device; if the connection cannot be made (for example, a dialer gets a busy signal), uucico tries the next available device. If this also fails, it returns to L.sys to look for another line for the same System. If none is found, uucico gives up.

System is the hostname of the remote system. Every machine with which this system communicates via UUCP should be listed, regardless of who calls whom. Systems not listed in L.sys will not be permitted a connection. The local hostname should not appear here for security reasons.

Times is a comma-separated list of the times of the day and week that calls are permitted to this System. Times is most commonly used to restrict long distance telephone calls to those times when rates are lower. List items are constructed as:

keywordhhmm-hhmm/grade;retry_time

Keyword is required, and must be one of:

Any Any time, any day of the week.

Wk Any weekday. In addition, Mo, Tu, We, Th, Fr, Sa, and Su can be used for Monday through Sunday, respectively.

Evening When evening telephone rates are in effect, from 1700 to 0800 Monday through Friday, and all day Saturday and Sunday. Evening is the same as Wk1700-0800,Sa,Su.

Night When nighttime telephone rates are in effect, from 2300 to 0800 Monday through Friday, all day Saturday, and from 2300 to 1700 Sunday. Night is the same as Any2300-0800,Sa,Su0800-1700.

NonPeak This is a slight modification of Evening. It matches when the USA X.25 carriers have their lower rate period. This is 1800 to 0700 Monday through Friday, and all day Saturday and Sunday. NonPeak is the same as Any1800-0700,Sa,Su.

Never Never call; calling into this System is forbidden or impossible. This is intended for polled connections, where the remote system calls into the local machine periodically. This is necessary when one of the machines is lacking either dial-in or dial-out modems.

The optional hhmm-hhmm subfield provides a time range that modifies the keyword. hhmm refers to hours and minutes in 24-hour time (from 0000 to 2359). The time range is permitted to "wrap" around midnight, and will behave in the obvious way. It is invalid to follow the Evening, NonPeak, and Night keywords with a time range.

The grade subfield is optional; if present, it is composed of a `/' (slash) and single character denoting the grade of the connection, from 0 to 9, A to Z, or a to z. This specifies that only requests of grade grade or better will be transferred during this time. (The grade of a request or job is specified when it is queued by uucp or uux.) By convention, mail is sent at grade C, news is sent at grade d, and uucp copies are sent at grade n. Unfortunately, some sites do not follow these conventions, so it is not 100% reliable.

The retry_time subfield is optional; it must be preceded by a `;' (semicolon) and specifies the time, in minutes, before a failed connection may be tried again. (This restriction is in addition to any constraints imposed by the rest of the Time field.) By default, the retry time starts at 10 minutes

and gradually increases at each failure, until after 26 tries uucico gives up completely (MAX RETRIES). If the retry time is too small, uucico may run into MAX RETRIES too soon.

Caller is the type of device used:

ACU Automatic call unit or auto-dialing modem such as the Hayes Smartmodem 1200 or Novation ``Smart Cat''. See L-devices for a list of supported modems.

DIR Direct connect; hardwired line (usually RS-232) to a remote system.

MICOM Micom Terminal Switch.

PAD X.25 PAD connection.

PCP GTE Telenet PC Pursuit. See L-devices for configuration details.

SYTEK Sytek high-speed dedicated modem port connection.

TCP Berkeley TCP/IP or 3Com UNET connection. These are mutually exclusive. TCP ports do not need entries in L-devices since all the necessary information is contained in L.sys. If several alternate ports or network connections should be tried, use multiple L.sys entries.

Class is usually the speed (baud) of the device, typically 300, 1200, or 2400 for ACU devices and 9600 for direct lines. Valid values are device dependent, and are specified in the L-devices file.

On some devices, the baud may be preceded by a non-numeric prefix. This is used in L-devices to distinguish among devices that have identical Caller and baud, but yet are distinctly different. For example, 1200 could refer to all Bell 212-compatible modems, V1200 to Racal-Vadic modems, and C1200 to CCITT modems, all at 1200 baud.

On TCP connections, Class is the port number (an integer number) or a port name from /etc/services that is used to make the connection. For standard Berkeley TCP/IP, UUCP normally uses port number 540.

Device/Phone_Number varies based on the Caller field. For ACU devices, this is the phone number to dial. The number may include: digits 0 through 9; # and * for dialing those symbols on tone telephone lines; - (hyphen) to pause for a moment, typically two to four seconds; = (equal sign) to wait for a second dial tone (implemented as a pause on many

modems). Other characters are modem dependent; generally standard telephone punctuation characters (such as the slash and parentheses) are ignored, although uucico does not guarantee this.

The phone number can be preceded by an alphabetic string; the string is indexed and converted through the L-dialcodes(5) file.

For DIR devices, the Device/Phone_Number field contains the name of the device in /dev that is used to make the connection. There must be a corresponding line in L-devices with identical Caller, Class, and Device fields.

For TCP and other network devices, Device/Phone_Number holds the true network name of the remote system, which may be different from its UUCP name (although one would hope not).

Expect and Send refer to an arbitrarily long set of strings that alternately specify what to expect and what to send to login to the remote system once a physical connection has been established. A complete set of expect/send strings is referred to as an expect/send script. The same syntax is used in the L-devices file to interact with the dialer prior to making a connection; there it is referred to as a chat script. The complete format for one expect/send pair is:

```
expect-timeout-send-expect-timeout   send
```

Expect and Send are character strings. Expect is compared against incoming text from the remote host; send is sent back when expect is matched. By default, the send is followed by a `\r' (carriage return). If the expect string is not matched within timeout seconds (default 45), then it is assumed that the match failed. The `expect-send-expect' notation provides a limited loop mechanism; if the first expect string fails to match, then the send string between the hyphens is transmitted, and uucico waits for the second expect string. This can be repeated indefinitely. When the last expect string fails, uucico hangs up and logs that the connection failed.

The timeout can (optionally) be specified by appending the parameter `~nn' to the expect string, when nn is the timeout time in seconds.

Backslash escapes that may be imbedded in the expect or send strings include:

```
\b    Generate a 3/10 second BREAK.
\bn   Where n is a single-digit number;
      generate an n/10 second BREAK.
```

\c	Suppress the \r at the end of a send string.
\d	Delay; pause for 1 second. (Send only.)
\r	Carriage Return.
\s	Space.
\n	Newline.
\xxx	Where xxx is an octal constant; denotes the corresponding ASCII character.

As a special case, an empty pair of double-quotes "" in the expect string is interpreted as ``expect nothing''; that is, transmit the send string regardless of what is received. Empty double-quotes in the send string cause a lone `\r' (carriage return) to be sent.

One of the following keywords may be substituted for the send string:

BREAK	Generate a 3/10 second BREAK
BREAKn	Generate an n/10 second BREAK
CR	Send a Carriage Return (same as "").
EOT	Send an End-Of-Transmission character, ASCII \004. Note that this will cause most hosts to hang up.
NL	Send a Newline.
PAUSE	Pause for 3 seconds.
PAUSEn	Pause for n seconds.
P_ODD	Use odd parity on future send strings.
P_ONE	Use parity one on future send strings.
P_EVEN	Use even parity on future send strings. (Default)
P_ZERO	Use parity zero on future send strings.

Finally, if the expect string consists of the keyword ABORT, then the string following is used to arm an abort trap. If that string is subsequently received any time prior to the completion of the entire expect/send script, then uucico will abort, just as if the script had timed out. This is useful for trapping error messages from port selectors or front-end processors such as ``Host Unavailable'' or ``System is Down.''

For example:

```
"" "" ogin:--ogin: nuucp ssword: ufeedme
```

This is executed as, ``When the remote system answers, expect nothing. Send a carriage return. Expect the remote to transmit the string `ogin:'. If it doesn't within 45 seconds, send another carriage return. When it finally does, send it the string `nuucp'. Then expect the string `ssword:'; when that is received, send `ufeedme'.''

FILES

/etc/uucp/L.sys

/etc/uucp/UUAIDS/L.sysL.sys example

SEE ALSO

uucp(1), uux(1), L-devices(5), services(5), uucico(8)

BUGS

``ABORT'' in the send/expect script is expressed ``backwards,'' that is, it should be written `` expect ABORT'' but instead it is `` ABORT expect''.

Several of the backslash escapes in the send/expect strings are confusing and/or different from those used by AT&T and Honey-Danber UUCP. For example, `\b' requests a BREAK, while practically everywhere else `\b' means backspace. `\t' for tab and `\f' for formfeed are not implemented. `\s' is a kludge; it would be more sensible to be able to delimit strings with quotation marks.

NAME

map3270 - database for mapping ascii keystrokes into IBM 3270 keys

SYNOPSIS

/usr/share/misc/map3270

DESCRIPTION

When emulating IBM-syle 3270 terminals under UNIX (see tn3270(1)), a mapping must be performed between sequences of keys hit on a user's (ascii) keyboard, and the keys that are available on a 3270. For example, a 3270 has a key labeled EEOF which erases the contents of the current field from the location of the cursor to the end. In order to accomplish this function, the terminal user and a program emulating a 3270 must agree on what keys will be typed to invoke the EEOF function.

The requirements for these sequences are:

1.) that the first character of the sequence be outside of the standard ascii printable characters;

2.) that no one sequence be an initial part of another (although sequences may share initial parts).

FORMAT

The file consists of entries for various terminals. The first part of an entry lists the names of the terminals which use that entry. These names should be the same as in /etc/termcap (see termcap(5)); note that often the terminals from various termcap entries will all use the same map3270 entry; for example, both 925 and 925vb (for 925 with visual bells) would probably use the same map3270 entry. After the names, separated by vertical bars (`|'), comes a left brace (`{'); the definitions; and, finally, a right brace (`}').

The definitions consist of a reserved keyword (see list below) which identifies the 3270 function (extended as defined below), followed by an equal sign (`='), followed by the various ways to generate this particular function, followed by a semi-colon (`;'). Each way is a sequence of strings of printable ascii characters enclosed inside single quotes (`''); various ways (options) are separated by vertical bars (`|').

Inside the single quotes, a few characters are special. A caret (`^') specifies that the next character is the ``control'' character of whatever the character is. So, `^a' represents control-a, ie: hexadecimal 1 (note that `^A' would generate the same code). To generate rubout, one

enters `^?'. To represent a control character inside a file requires using the caret to represent a control sequence; simply typing control-A will not work. Note: the ctrl-caret sequence (to generate a hexadecimal 1E) is represented as `^^' (not `^\^').

In addition to the caret, a letter may be preceeded by a backslash (`\'). Since this has little effect for most characters, its use is usually not recommended. For the case of a single quote (`''), the backslash prevents that single quote from terminating the string. To have the backslash be part of the string, it is necessary to place two backslashes ('\\') in the file.

In addition, the following characters are special:

`\E'	means an escape character;
`\n'	means newline;
`\t'	means tab;
`\r'	means carriage return.

It is not necessary for each character in a string to be enclosed within single quotes. `\E\E\E' means three escape characters.

Comments, which may appear anywhere on a line, begin with a hash mark (`#'), and terminate at the end of that line. However, comments cannot begin inside a quoted string; a hash mark inside a quoted string has no special meaning.

3270 KEYS SUPPORTED

The following is the list of 3270 key names that are supported in this file. Note that some of the keys don't really exist on a 3270. In particular, the developers of this file have relied extensively on the work at the Yale University Computer Center with their 3270 emulator which runs in an IBM Series/1 front end. The following list corresponds closely to the functions that the developers of the Yale code offer in their product.

In the following list, the starred ("*") functions are not supported by tn3270(1). An unsupported function will cause tn3270(1) to send a bell sequence to the user's terminal.

3270 Key Name	Functional description
(*)LPRT	local print
DP	dup character
FM	field mark character
(*)CURSEL	cursor select
RESHOW	redisplay the screen
EINP	erase input

```
   EEOF        erase end of field
   DELETE      delete character
   INSRT       toggle insert mode
   TAB         field tab
   BTAB        field back tab
   COLTAB      column tab
   COLBAK      column back tab
   INDENT      indent one tab stop
   UNDENT      undent one tab stop
   NL          new line
   HOME        home the cursor
   UP          up cursor
   DOWN        down cursor
   RIGHT       right cursor
   LEFT        left cursor
   SETTAB      set a column tab
   DELTAB      delete a columntab
   SETMRG      set left margin
   SETHOM      set home position
   CLRTAB      clear all column tabs
(*)APLON       apl on
(*)APLOFF      apl off
(*)APLEND      treat input as ascii
(*)PCON        xon/xoff on
(*)PCOFF       xon/xoff off
   DISC        disconnect (suspend)
(*)INIT        new terminal type
(*)ALTK        alternate keyboard dvorak
   FLINP       flush input
   ERASE       erase last character
   WERASE      erase last word
   FERASE      erase field
   SYNCH       we are in synch with the user
   RESET       reset key-unlock keyboard
   MASTER_RESET    reset, unlock and redisplay
(*)XOFF        please hold output
(*)XON         please give me output
   ESCAPE      enter telnet command mode
   WORDTAB     tab to beginning of next word
   WORDBACKTAB tab to beginning of current/last word
   WORDEND     tab to end of current/next word
   FIELDEND    tab to last non-blank of current/next
               unprotected (writable) field.

   PA1         program attention 1
   PA2         program attention 2
   PA3         program attention 3

   CLEAR       local clear of the 3270 screen
   TREQ        test request
   ENTER       enter key
```

```
PFK1        program function key 1
PFK2        program function key 2
etc.        etc.
PFK36       program function key 36
```

A SAMPLE ENTRY

The following entry is used by tn3270(1) when unable to locate a reasonable version in the user's environment and in /usr/share/misc/map3270:

```
name {      # actual name comes from TERM variable
clear = '^z';
flinp = '^x';
enter = '^m';
delete = '^d' | '^?';   # note that '^?' is delete (rubout)
synch = '^r';
reshow = '^v';
eeof = '^e';
tab = '^i';
btab = '^b';
nl = '^n';
left = '^h';
right = '^l';
up = '^k';
down = '^j';
einp = '^w';
reset = '^t';
xoff = '^s';
xon = '^q';
escape = '^c';
ferase = '^u';
insrt = 'E ';
# program attention keys
pa1 = '^p1'; pa2 = '^p2'; pa3 = '^p3';
# program function keys
pfk1 = 'E1'; pfk2 = 'E2'; pfk3 = 'E3'; pfk4 = 'E4';
pfk5 = 'E5'; pfk6 = 'E6'; pfk7 = 'E7'; pfk8 = 'E8';
pfk9 = 'E9'; pfk10 = 'E0'; pfk11 = 'E-'; pfk12 = 'E=';
pfk13 = 'E!'; pfk14 = 'E@'; pfk15 = 'E#'; pfk16 = 'E$';
pfk17 = 'E%'; pfk18 = 'E'; pfk19 = 'E&'; pfk20 = 'E*';
pfk21 = 'E('; pfk22 = 'E)'; pfk23 = 'E_'; pfk24 = 'E+';
}
```

IBM 3270 KEY DEFINITONS FOR AN ABOVE DEFINITION

The charts below show the proper keys to emulate each 3270 function when using the default key mapping supplied with tn3270(1) and mset(1).

Command Keys	IBM 3270 Key	Default Key(s)
	Enter	RETURN
	Clear	control-z
Cursor Movement Keys		

	New Line	control-n or Home
	Tab	control-i
	Back Tab	control-b
	Cursor Left	control-h
	Cursor Right	control-l
	Cursor Up	control-k
	Cursor Down	control-j or LINE FEED
Edit Control Keys		
	Delete Char	control-d or RUB
	Erase EOF	control-e
	Erase Input	control-w
	Insert Mode	ESC Space
	End Insert	ESC Space
Program Function Keys		
	PF1	ESC 1
	PF2	ESC 2
	...	...
	PF10	ESC 0
	PF11	ESC -
	PF12	ESC =
	PF13	ESC !
	PF14	ESC @
	...	...
	PF24	ESC +
Program Attention Keys		
	PA1	control-p 1
	PA2	control-p 2
	PA3	control-p 3
Local Control Keys		
	Reset After Error	control-r
	Purge Input Buffer	control-x
	Keyboard Unlock	control-t
	Redisplay Screen	control-v
Other Keys		
	Erase current field	control-u

FILES

/usr/share/misc/map3270

SEE ALSO

tn3270(1), mset(1), Yale ASCII Terminal Communication System II Program Description/Operator's Manual (IBM SB30-1911)

AUTHOR

Greg Minshall

BUGS

Tn3270 doesn't yet understand how to process all the functions available in map3270; when such a function is

requested tn3270 will beep at you.

The definition of "word" (for "word delete", "word tab") should be a run-time option. Currently it is defined as the kernel tty driver defines it (strings of non-blanks); more than one person would rather use the "vi" definition (strings of specials, strings of alphanumeric).

NAME

networks - network name data base

DESCRIPTION

The networks file contains information regarding the known networks which comprise the DARPA Internet. For each network a single line should be present with the following information:

official network name
network number
aliases

Items are separated by any number of blanks and/or tab characters. A ``#'' indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines which search the file. This file is normally created from the official network data base maintained at the Network Information Control Center (NIC), though local changes may be required to bring it up to date regarding unofficial aliases and/or unknown networks.

Network number may be specified in the conventional ``.'' notation using the inet_network() routine from the Internet address manipulation library, inet(3N). Network names may contain any printable character other than a field delimiter, newline, or comment character.

FILES

/etc/networks

SEE ALSO

getnetent(3N)

BUGS

A name server should be used instead of a static file.

NAME

passwd - password files

DESCRIPTION

Passwd files are files consisting of newline separated records, one per user, containing ten colon (``:'') separated fields. These fields are as follows:

name	user's login name
password	user's encrypted password
uid	user's id
gid	user's login group id
class	user's general classification (unused)
change	password change time
expire	account expiration time
gecos	general information about the user
home_dir	user's home directory
shell	user's login shell

The name field is the login used to access the computer account, and the uid field is the number associated with it. They should both be unique across the system (and often across a group of systems) since they control file access.

While it is possible to have multiple entries with identical login names and/or identical user id's, it is usually a mistake to do so. Routines that manipulate these files will often return only one of the multiple entries, and that one by random selection.

The login name must never begin with a hyphen (``-''); also, it is strongly suggested that neither upper-case characters or dots (``.'') be part of the name, as this tends to confuse mailers. No field may contain a colon (``:'') as this has been used historically to separate the fields in the user database.

The password field is the encrypted form of the password. If the password field is empty, no password will be required to gain access to the machine. This is almost invariably a mistake. Because these files contain the encrypted user passwords, they should not be readable by anyone without appropriate privileges.

The group field is the group that the user will be placed in upon login. Since this system supports multiple groups (see groups(1)) this field currently has little special meaning.

The class field is currently unused. In the near future it will be a key to a termcap(5) style database of user attributes.

The change field is the number in seconds, GMT, from the epoch, until the password for the account must be changed. This field may be left empty to turn off the password aging feature.

The expire field is the number in seconds, GMT, from the epoch, until the account expires. This field may be left empty to turn off the account aging feature.

The gecos field normally contains comma (``,'') separated subfields as follows:

```
name    user's full name
office  user's office number
wphone  user's work phone number
hphone  user's home phone number
```

This information is used by the finger(1) program.

The user's home directory is the full UNIX path name where the user will be placed on login.

The shell field is the command interpreter the user prefers. If the shell field is empty, the Bourne shell (/bin/sh) is assumed.

SEE ALSO

chpass(1), login(1), passwd(1), getpwent(3), mkpasswd(8), vipw(8) adduser(8)

BUGS

User information should (and eventually will) be stored elsewhere.

NAME

phones - remote host phone number data base

DESCRIPTION

The file /etc/phones contains the system-wide private phone numbers for the tip(1C) program. This file is normally unreadable, and so may contain privileged information. The format of the file is a series of lines of the form: <system-name>[\t]*<phone-number>. The system name is one of those defined in the remote(5) file and the phone number is constructed from any sequence of characters terminated only by ``,'' or the end of the line. The ``='' and ``*'' characters are indicators to the auto call units to pause and wait for a second dial tone (when going through an exchange). The ``='' is required by the DF02-AC and the ``*'' is required by the BIZCOMP 1030.

Only one phone number per line is permitted. However, if more than one line in the file contains the same system name tip(1C) will attempt to dial each one in turn, until it establishes a connection.

FILES

/etc/phones

SEE ALSO

tip(1C), remote(5)

NAME

plot - graphics interface

DESCRIPTION

Files of this format are produced by routines described in plot(3X) and plot(3F), and are interpreted for various devices by commands described in plot(1G). A graphics file is a stream of plotting instructions. Each instruction consists of an ASCII letter usually followed by bytes of binary information. The instructions are executed in order. A point is designated by four bytes representing the x and y values; each value is a signed integer. The last designated point in an l, m, n, a, or p instruction becomes the `current point' for the next instruction. The a and c instructions change the current point in a manner dependent upon the specific device.

Each of the following descriptions begins with the name of the corresponding routine in plot(3X).

m move: The next four bytes give a new current point.

n cont: Draw a line from the current point to the point given by the next four bytes.

p point: Plot the point given by the next four bytes.

l line: Draw a line from the point given by the next four bytes to the point given by the following four bytes.

t label: Place the following ASCII string so that its first character falls on the current point. The string is terminated by a newline.

a arc: The first four bytes give the center, the next four give the starting point, and the last four give the end point of a circular arc. The least significant coordinate of the end point is used only to determine the quadrant. The arc is drawn counter-clockwise.

c circle: The first four bytes give the center of the circle, the next two the radius.

e erase: Start another frame of output.

f linemod: Take the following string, up to a newline, as the style for drawing further lines. The styles are `dotted,' `solid,' `longdashed,' `shortdashed,' and `dotdashed.' Effective only in plot 4014 and plot ver.

s space: The next four bytes give the lower left corner of the plotting area; the following four give the upper

right corner. The plot will be magnified or reduced to fit the device as closely as possible.

Space settings that exactly fill the plotting area with unity scaling appear below for devices supported by the filters of plot(1G). The upper limit is just outside the plotting area. In every case the plotting area is taken to be square; points outside may be displayable on devices whose face isn't square.

```
4013        space(0, 0,  780,  780);
4014        space(0, 0, 3120, 3120);
ver         space(0, 0, 2048, 2048);
300, 300s   space(0, 0, 4096, 4096);
450         space(0, 0, 4096, 4096);
```

SEE ALSO

plot(1G), plot(3X), plot(3F), graph(1G)

NAME

printcap - printer capability data base

SYNOPSIS

/etc/printcap

DESCRIPTION

Printcap is a simplified version of the termcap(5) data base used to describe line printers. The spooling system accesses the printcap file every time it is used, allowing dynamic addition and deletion of printers. Each entry in the data base is used to describe one printer. This data base may not be substituted for, as is possible for termcap, because it may allow accounting to be bypassed.

The default printer is normally lp, though the environment variable PRINTER may be used to override this. Each spooling utility supports an option, -Pprinter, to allow explicit naming of a destination printer.

Refer to the 4.3BSD Line Printer Spooler Manual for a complete discussion on how setup the database for a given printer.

CAPABILITIES

Refer to termcap(5) for a description of the file layout.

Name	Type	Default	Description
af	str	NULL	name of accounting file
br	num	none	if lp is a tty, set the baud rate (ioctl call)
cf	str	NULL	cifplot data filter
df	str	NULL	tex data filter (DVI format)
fc	num	0	if lp is a tty, clear flag bits (sgtty.h)
ff	str	``\f''	string to send for a form feed
fo	bool	false	print a form feed when device is opened
fs	num	0	like `fc' but set bits
gf	str	NULL	graph data filter (plot (3X) format)
hl	bool	false	print the burst header page last
ic	bool	false	driver supports (non standard) ioctl to indent printout
if	str	NULL	name of text filter which does accounting
lf	str	/dev/console	error logging file name
lo	str	lock	name of lock file
lp	str	/dev/lp	device name to open for output
mx	num	1000	maximum file size (in BUFSIZ blocks), zero = unlimited
nd	str	NULL	next directory for list of queues (unimplemented)
nf	str	NULL	ditroff data filter (device independent troff)
of	str	NULL	name of output filtering program
pc	num	200	price per foot or page in hundredths of cents
pl	num	66	page length (in lines)
pw	num	132	page width (in characters)
px	num	0	page width in pixels (horizontal)
py	num	0	page length in pixels (vertical)

```
rf     str   NULL      filter for printing FORTRAN style text files
rg     str   NULL      restricted group. Only members of group allowed access
rm     str   NULL      machine name for remote printer
rp     str   lp        remote printer name argument
rs     bool  false     restrict remote users to those with local accounts
rw     bool  false     open the printer device for reading and writing
sb     bool  false     short banner (one line only)
sc     bool  false     suppress multiple copies
sd     str   /usr/spool/lpd  spool directory
sf     bool  false     suppress form feeds
sh     bool  false     suppress printing of burst page header
st     str   status    status file name
tf     str   NULL      troff data filter (cat phototypesetter)
tr     str   NULL      trailer string to print when queue empties
vf     str   NULL      raster image filter
xc     num   0         if lp is a tty, clear local mode bits (tty (4))
xs     num   0         like `xc' but set bits
```

If the local line printer driver supports indentation, the daemon must understand how to invoke it.

FILTERS

The lpd(8) daemon creates a pipeline of filters to process files for various printer types. The filters selected depend on the flags passed to lpr(1). The pipeline set up is:

```
-p   pr | if    regular text + pr(1)
none if   regular text
-c   cf   cifplot
-d   df   DVI (tex)
-g   gf   plot(3)
-n   nf   ditroff
-f   rf   Fortran
-t   tf   troff
-v   vf   raster image
```

The if filter is invoked with arguments:

```
if [ -c ] -wwidth -llength -iindent -n login -h host acct-file
```

The -c flag is passed only if the -l flag (pass control characters literally) is specified to lpr. Width and length specify the page width and length (from pw and pl respectively) in characters. The -n and -h parameters specify the login name and host name of the owner of the job respectively. Acct-file is passed from the af printcap entry.

If no if is specified, of is used instead, with the distinction that of is opened only once, while if is opened for every individual job. Thus, if is better suited to

performing accounting. The of is only given the width and length flags.

All other filters are called as:

```
filter -xwidth -ylength -n login -h host acct-file
```

where width and length are represented in pixels, specified by the px and py entries respectively.

All filters take stdin as the file, stdout as the printer, may log either to stderr or using syslog(3), and must not ignore SIGINT.

LOGGING

Error messages generated by the line printer programs themselves (that is, the lp* programs) are logged by syslog(3) using the LPR facility. Messages printed on stderr of one of the filters are sent to the corresponding lf file. The filters may, of course, use syslog themselves.

Error messages sent to the console have a carriage return and a line feed appended to them, rather than just a line feed.

SEE ALSO

termcap(5), lpc(8), lpd(8), pac(8), lpr(1), lpq(1), lprm(1)
4.3BSD Line Printer Spooler Manual

NAME

protocols - protocol name data base

DESCRIPTION

The protocols file contains information regarding the known protocols used in the DARPA Internet. For each protocol a single line should be present with the following information:

official protocol name
protocol number
aliases

Items are separated by any number of blanks and/or tab characters. A ``#'' indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines which search the file.

Protocol names may contain any printable character other than a field delimiter, newline, or comment character.

FILES

/etc/protocols

SEE ALSO

getprotoent(3N)

BUGS

A name server should be used instead of a static file.

NAME

remote - remote host description file

DESCRIPTION

The systems known by tip(1C) and their attributes are stored in an ASCII file which is structured somewhat like the termcap(5) file. Each line in the file provides a description for a single system. Fields are separated by a colon (``:''). Lines ending in a \ character with an immediately following newline are continued on the next line.

The first entry is the name(s) of the host system. If there is more than one name for a system, the names are separated by vertical bars. After the name of the system comes the fields of the description. A field name followed by an `=' sign indicates a string value follows. A field name followed by a `#' sign indicates a following numeric value.

Entries named ``tip*'' and ``cu*'' are used as default entries by tip, and the cu interface to tip, as follows. When tip is invoked with only a phone number, it looks for an entry of the form ``tip300'', where 300 is the baud rate with which the connection is to be made. When the cu interface is used, entries of the form ``cu300'' are used.

CAPABILITIES

Capabilities are either strings (str), numbers (num), or boolean flags (bool). A string capability is specified by capability=value; e.g. ``dv=/dev/harris''. A numeric capability is specified by capability#value; e.g. ``xa#99''. A boolean capability is specified by simply listing the capability.

at (str) Auto call unit type.

br (num) The baud rate used in establishing a connection to the remote host. This is a decimal number. The default baud rate is 300 baud.

cm (str) An initial connection message to be sent to the remote host. For example, if a host is reached through port selector, this might be set to the appropriate sequence required to switch to the host.

cu (str) Call unit if making a phone call. Default is the same as the `dv' field.

di (str) Disconnect message sent to the host when a disconnect is requested by the user.

du (bool) This host is on a dial-up line.

dv (str) UNIX device(s) to open to establish a connection. If this file refers to a terminal line, tip(1C) attempts to perform an exclusive open on the device to insure only one user at a time has access to the port.

el (str) Characters marking an end-of-line. The default is NULL. `~' escapes are only recognized by tip after one of the characters in `el', or after a carriage-return.

fs (str) Frame size for transfers. The default frame size is equal to BUFSIZ.

hd (bool) The host uses half-duplex communication, local echo should be performed.

ie (str) Input end-of-file marks. The default is NULL.

oe (str) Output end-of-file string. The default is NULL. When tip is transferring a file, this string is sent at end-of-file.

pa (str) The type of parity to use when sending data to the host. This may be one of ``even'', ``odd'', ``none'', ``zero'' (always set bit 8 to zero), ``one'' (always set bit 8 to 1). The default is even parity.

pn (str) Telephone number(s) for this host. If the telephone number field contains an @ sign, tip searches the file /etc/phones file for a list of telephone numbers; c.f. phones(5).

tc (str) Indicates that the list of capabilities is continued in the named description. This is used primarily to share common capability information.

Here is a short example showing the use of the capability continuation feature:

```
UNIX-1200:\
   :dv=/dev/cau0:el=^D^U^C^S^Q^O@:du:at=ventel:ie=#$%:oe=^D:br#1200:
arpavax|ax:\
   :pn=7654321%:tc=UNIX-1200
```

FILES

/etc/remote

SEE ALSO

tip(1C), phones(5)

NAME

resolver - resolver configuration file

SYNOPSIS

/etc/resolv.conf

DESCRIPTION

The resolver configuration file contains information that is read by the resolver routines the first time they are invoked by a process. The file is designed to be human readable and contains a list of name-value pairs that provide various types of resolver information.

On a normally configured system this file should not be necessary. The only name server to be queried will be on the local machine and the domain name is retrieved from the system.

The different configuration options are:

nameserver

followed by the Internet address (in dot notation) of a name server that the resolver should query. At least one name server should be listed. Up to MAXNS (currently 3) name servers may be listed, in that case the resolver library queries tries them in the order listed. If no nameserver entries are present, the default is to use the name server on the local machine. (The algorithm used is to try a name server, and if the query times out, try the next, until out of name servers, then repeat trying all the name servers until a maximum number of retries are made).

domain

followed by a domain name, that is the default domain to append to names that do not have a dot in them. If no domain entries are present, the domain returned by gethostname(2) is used (everything after the first `.'). Finally, if the host name does not contain a domain part, the root domain is assumed.

The name value pair must appear on a single line, and the keyword (e.g. nameserver) must start the line. The value follows the keyword, separated by white space.

FILES

/etc/resolv.conf

SEE ALSO

gethostbyname(3N), resolver(3), named(8)
Name Server Operations Guide for BIND

NAME

services - service name data base

DESCRIPTION

The services file contains information regarding the known services available in the DARPA Internet. For each service a single line should be present with the following information:

official service name
port number
protocol name
aliases

Items are separated by any number of blanks and/or tab characters. The port number and protocol name are considered a single item; a ``/'' is used to separate the port and protocol (e.g. ``512/tcp''). A ``#'' indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines which search the file.

Service names may contain any printable character other than a field delimiter, newline, or comment character.

FILES

/etc/services

SEE ALSO

getservent(3N)

BUGS

A name server should be used instead of a static file.

NAME

shells - shell database

DESCRIPTION

The shells file contains a list of the shells on the system. For each shell a single line should be present, consisting of the shell's path, relative to root.

A hash mark (``#'') indicates the beginning of a comment; subsequent characters up to the end of the line are not interpreted by the routines which search the file. Blank lines are also ignored.

FILES

/etc/shells

SEE ALSO

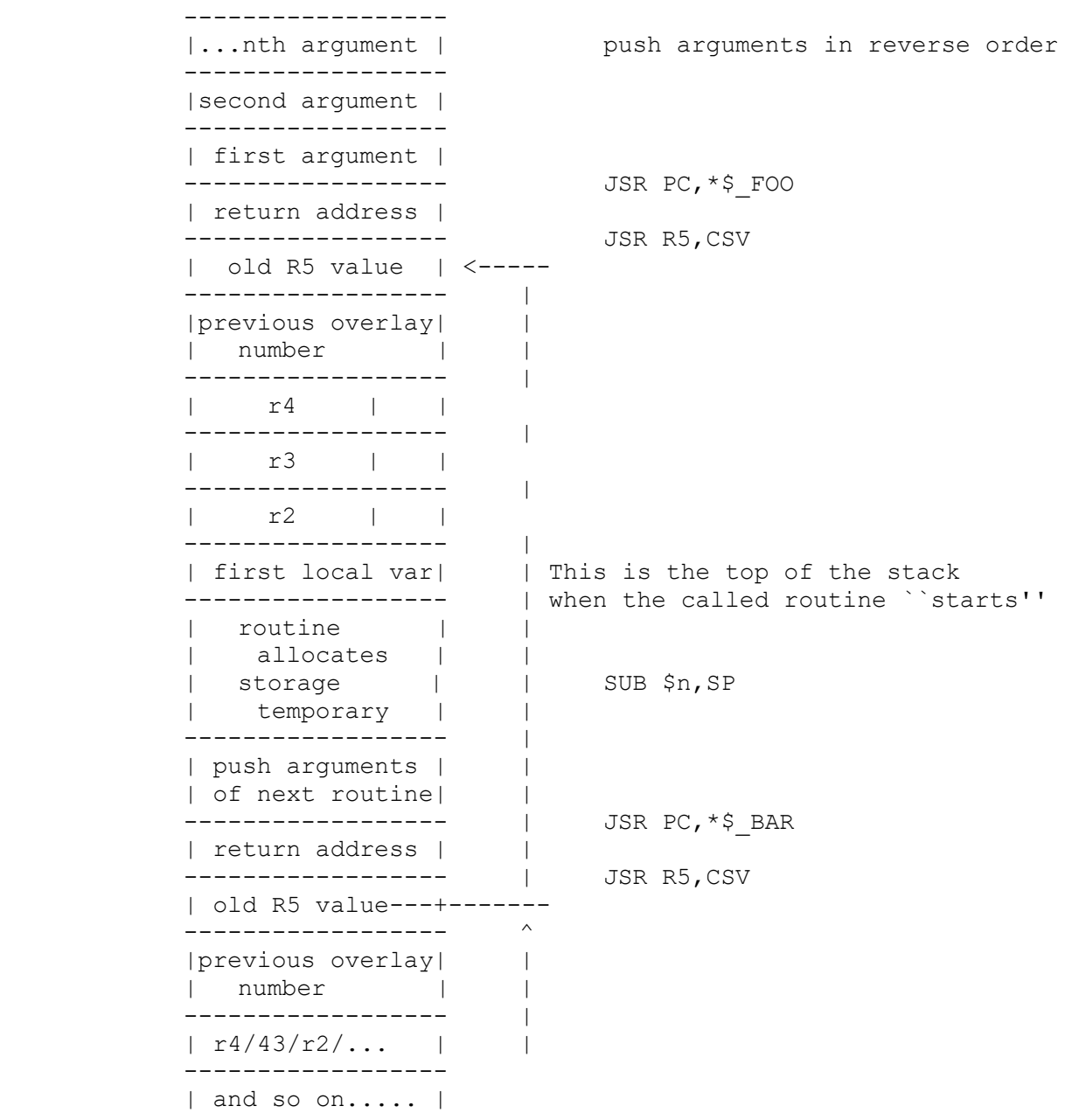
getusershell(3)

NAME

stack - 2.10BSD PDP-11 C stack frame conventions

DESCRIPTION

The standard C stack frame layout:



The stack pushes downward through memory addresses. Overlay numbers saved in non-overlaid objects are always zero, but the simplification of not having to maintain two different stack frame formats more than outweighs the extra few micro

seconds (less than four) necessary to save the zero ...

Functions returning integers leave their return value in R0; functions returning floating constants use FR0; functions returning longs leave return values in R1/R0 (R0 high word, R1 low); functions returning structures leave a pointer to bss storage (one chunk of which is allocated for each such routine) in R0, and the caller will copy from that bss storage to the local destination.

Local variables are allocated in such a way that they are referred to as ``-N(R5)'', arguments are referred to as ``+N(R5)''; arguments start at 4(R5), the first integer local declared will be at -10(R5).

The SP normally points at the first word available for parameter pushing. A function taking only single word as a parameter can be called simply by moving the parameter into (SP) and calling the function, without having to clean the parameter off the stack on return. Any parameters passed after the first (actually "Nth") must be pushed before the call and cleaned off afterwards. If the function has no local variables and calls no functions, it will allocate no stack and the word labelled ``first local var'' will be unused.

It is important to note that routines know how many arguments they pass to a function, and will adjust the stack accordingly after a function returns.

NOTE

This stack frame format is the same as that used by overlaid objects in 2.9BSD.

AUTHOR

John F. Woods, MIT Concouse Computer Center

NAME

syserrlst - error message file format

DESCRIPTION

mkerrlst(1), creates error message files in the format described below.

An ``error message file'' consists of a header, an array of structures specifying the offset and length of each message, and the array of message strings separated by newlines.

The message strings are separated by newlines but the newline characters are not included in the size of the message. These newline characters serve only to make the file editable or printable (after stripping off the header).

The file format is:

```
  /*
   * Definitions used by the 'mkerrlst' program which creates error message
   * files.
   *
   * The format of the file created is:
   *
   *   struct    ERRLSTHDR ehdr;
   *   struct    ERRLST  emsg[num_of_messages];
   *   struct    {
   *        char msg[] = "error message string";
   *        char lf = '0;
   *        } [num_of_messages];
   *
   * Note:  the newlines are NOT included in the message lengths, the newlines
   *        are present to make it easy to 'cat' or 'vi' the file.
   */

        struct    ERRLSTHDR
             {
             short      magic;
             short      maxmsgnum;
             short      maxmsglen;
             short      pad[5];    /* Reserved */
             };

        struct    ERRLST
             {
             off_t      offmsg;
             short      lenmsg;
             };

     #define   ERRMAGIC  012345
```

SEE ALSO

mkerrlst(1), syserrlst(3)

BUGS

Format of the file isn't necessarily portable between machines.

HISTORY

This file format is new with 2.11BSD.

NAME

tar - tape archive file format

DESCRIPTION

Tar, (the tape archive command) dumps several files into one, in a medium suitable for transportation.

A ``tar tape'' or file is a series of blocks. Each block is of size TBLOCK. A file on the tape is represented by a header block which describes the file, followed by zero or more blocks which give the contents of the file. At the end of the tape are two blocks filled with binary zeros, as an end-of-file indicator.

The blocks are grouped for physical I/O operations. Each group of n blocks (where n is set by the b keyletter on the tar(1) command line - default is 20 blocks) is written with a single system call; on nine-track tapes, the result of this write is a single tape record. The last group is always written at the full size, so blocks after the two zero blocks contain random data. On reading, the specified or default group size is used for the first read, but if that read returns less than a full tape block, the reduced block size is used for further reads.

The header block looks like:

```
#define TBLOCK 512
#define NAMSIZ 100

union hblock {
       char dummy[TBLOCK];
     struct header {
       char name[NAMSIZ];
       char mode[8];
       char uid[8];
       char gid[8];
       char size[12];
       char mtime[12];
       char chksum[8];
       char linkflag;
       char linkname[NAMSIZ];
     } dbuf;
};
```

Name is a null-terminated string. The other fields are zero-filled octal numbers in ASCII. Each field (of width w) contains w-2 digits, a space, and a null, except size and mtime, which do not contain the trailing null and chksum which has a null followed by a space. Name is the name of the file, as specified on the tar command line. Files dumped because they were in a directory which was named in

the command line have the directory name as prefix and /filename as suffix. Mode is the file mode, with the top bit masked off. Uid and gid are the user and group numbers which own the file. Size is the size of the file in bytes. Links and symbolic links are dumped with this field specified as zero. Mtime is the modification time of the file at the time it was dumped. Chksum is an octal ASCII value which represents the sum of all the bytes in the header block. When calculating the checksum, the chksum field is treated as if it were all blanks. Linkflag is NULL if the file is ``normal'' or a special file, ASCII `1' if it is an hard link, and ASCII `2' if it is a symbolic link. The name linked-to, if any, is in linkname, with a trailing null. Unused fields of the header are binary zeros (and are included in the checksum).

The first time a given i-node number is dumped, it is dumped as a regular file. The second and subsequent times, it is dumped as a link instead. Upon retrieval, if a link entry is retrieved, but not the file it was linked to, an error message is printed and the tape must be manually re-scanned to retrieve the linked-to file.

The encoding of the header is designed to be portable across machines.

SEE ALSO

tar(1)

BUGS

Names or linknames longer than NAMSIZ produce error reports and cannot be dumped.

NAME

termcap - terminal capability data base

SYNOPSIS

/etc/termcap

DESCRIPTION

Termcap is a data base describing terminals, used, e.g., by vi(1) and curses(3X). Terminals are described in termcap by giving a set of capabilities that they have and by describing how operations are performed. Padding requirements and initialization sequences are included in termcap.

Entries in termcap consist of a number of `:'-separated fields. The first entry for each terminal gives the names that are known for the terminal, separated by `|' characters. The first name is always two characters long and is used by older systems which store the terminal type in a 16-bit word in a system-wide data base. The second name given is the most common abbreviation for the terminal, the last name given should be a long name fully identifying the terminal, and all others are understood as synonyms for the terminal name. All names but the first and last should be in lower case and contain no blanks; the last name may well contain upper case and blanks for readability.

Terminal names (except for the last, verbose entry) should be chosen using the following conventions. The particular piece of hardware making up the terminal should have a root name chosen, thus "hp2621". This name should not contain hyphens. Modes that the hardware can be in or user preferences should be indicated by appending a hyphen and an indicator of the mode. Therefore, a "vt100" in 132-column mode would be "vt100-w". The following suffixes should be used where possible:

Suffix	Meaning	Example
-w	Wide mode (more than 80 columns)	vt100-w
-am	With automatic margins (usually default)	vt100-am
-nam	Without automatic margins	vt100-nam
-n	Number of lines on the screen	aaa-60
-na	No arrow keys (leave them in local)	concept100-na
-np	Number of pages of memory	concept100-4p
-rv	Reverse video	concept100-rv

CAPABILITIES

The characters in the Notes field in the table have the following meanings (more than one may apply to a capability):

N	indicates numeric parameter(s)
P	indicates that padding may be specified
*	indicates that padding may be based on the number of lines affected

o indicates capability is obsolete

"Obsolete" capabilities have no terminfo equivalents, since they were considered useless, or are subsumed by other capabilities. New software should not rely on them at all.

Name	Type	Notes	Description
ae	str	(P)	End alternate character set
AL	str	(NP*)	Add n new blank lines
al	str	(P*)	Add new blank line
am	bool		Terminal has automatic margins
as	str	(P)	Start alternate character set
bc	str	(o)	Backspace if not ^H
bl	str	(P)	Audible signal (bell)
bs	bool	(o)	Terminal can backspace with ^H
bt	str	(P)	Back tab
bw	bool		le (backspace) wraps from column 0 to last column
CC	str		Terminal settable command character in prototype
cd	str	(P*)	Clear to end of display
ce	str	(P)	Clear to end of line
ch	str	(NP)	Set cursor column (horizontal position)
cl	str	(P*)	Clear screen and home cursor
CM	str	(NP)	Memory-relative cursor addressing
cm	str	(NP)	Screen-relative cursor motion
co	num		Number of columns in a line (See BUGS section below)
cr	str	(P)	Carriage return
cs	str	(NP)	Change scrolling region (VT100)
ct	str	(P)	Clear all tab stops
cv	str	(NP)	Set cursor row (vertical position)
da	bool		Display may be retained above the screen
dB	num	(o)	Milliseconds of bs delay needed (default 0)
db	bool		Display may be retained below the screen
DC	str	(NP*)	Delete n characters
dC	num	(o)	Milliseconds of cr delay needed (default 0)
dc	str	(P*)	Delete character
dF	num	(o)	Milliseconds of ff delay needed (default 0)
DL	str	(NP*)	Delete n lines
dl	str	(P*)	Delete line
dm	str		Enter delete mode
dN	num	(o)	Milliseconds of nl delay needed (default 0)
DO	str	(NP*)	Move cursor down n lines
do	str		Down one line
ds	str		Disable status line
dT	num	(o)	Milliseconds of horizontal tab delay needed (default 0)
dV	num	(o)	Milliseconds of vertical tab delay needed (default 0)
ec	str	(NP)	Erase n characters
ed	str		End delete mode
ei	str		End insert mode
eo	bool		Can erase overstrikes with a blank
EP	bool	(o)	Even parity
es	bool		Escape can be used on the status line
ff	str	(P*)	Hardcopy terminal page eject

fs	str		Return from status line
gn	bool		Generic line type (e.g. dialup, switch)
hc	bool		Hardcopy terminal
HD	bool	(o)	Half-duplex
hd	str		Half-line down (forward 1/2 linefeed)
ho	str	(P)	Home cursor
hs	bool		Has extra "status line"
hu	str		Half-line up (reverse 1/2 linefeed)
hz	bool		Cannot print ~s (Hazeltine)
i1-i3	str		Terminal initialization strings (terminfo only)
IC	str	(NP*)	Insert n blank characters
ic	str	(P*)	Insert character
if	str		Name of file containing initialization string
im	str		Enter insert mode
in	bool		Insert mode distinguishes nulls
iP	str		Pathname of program for initialization (terminfo only)
ip	str	(P*)	Insert pad after character inserted
is	str		Terminal initialization string (termcap only)
it	num		Tabs initially every n positions
K1	str		Sent by keypad upper left
K2	str		Sent by keypad upper right
K3	str		Sent by keypad center
K4	str		Sent by keypad lower left
K5	str		Sent by keypad lower right
k0-k9	str		Sent by function keys 0-9
kA	str		Sent by insert-line key
ka	str		Sent by clear-all-tabs key
kb	str		Sent by backspace key
kC	str		Sent by clear-screen or erase key
kD	str		Sent by delete-character key
kd	str		Sent by down-arrow key
kE	str		Sent by clear-to-end-of-line key
ke	str		Out of "keypad transmit" mode
kF	str		Sent by scroll-forward/down key
kH	str		Sent by home-down key
kh	str		Sent by home key
kI	str		Sent by insert-character or enter-insert-mode key
kL	str		Sent by delete-line key
kl	str		Sent by left-arrow key
kM	str		Sent by insert key while in insert mode
km	bool		Has a "meta" key (shift, sets parity bit)
kN	str		Sent by next-page key
kn	num	(o)	Number of function (k0-k9) keys (default 0)
ko	str	(o)	Termcap entries for other non-function keys
kP	str		Sent by previous-page key
kR	str		Sent by scroll-backward/up key
kr	str		Sent by right-arrow key
kS	str		Sent by clear-to-end-of-screen key
ks	str		Put terminal in "keypad transmit" mode
kT	str		Sent by set-tab key
kt	str		Sent by clear-tab key
ku	str		Sent by up-arrow key

l0-l9	str		Labels on function keys if not "fn"
LC	bool	(o)	Lower-case only
LE	str	(NP)	Move cursor left n positions
le	str	(P)	Move cursor left one position
li	num		Number of lines on screen or page (See BUGS below)
ll	str		Last line, first column
lm	num		Lines of memory if > li (0 means varies)
ma	str	(o)	Arrow key map (used by vi version 2 only)
mb	str		Turn on blinking attribute
md	str		Turn on bold (extra bright) attribute
me	str		Turn off all attributes
mh	str		Turn on half-bright attribute
mi	bool		Safe to move while in insert mode
mk	str		Turn on blank attribute (characters invisible)
ml	str	(o)	Memory lock on above cursor
mm	str		Turn on "meta mode" (8th bit)
mo	str		Turn off "meta mode"
mp	str		Turn on protected attribute
mr	str		Turn on reverse-video attibute
ms	bool		Safe to move in standout modes
mu	str	(o)	Memory unlock (turn off memory lock)
nc	bool	(o)	No correctly-working cr (Datamedia 2500, Hazeltine 2000)
nd	str		Non-destructive space (cursor right)
NL	bool	(o)	\n is newline, not line feed
nl	str	(o)	Newline character if not \n
ns	bool	(o)	Terminal is a CRT but doesn't scroll
nw	str	(P)	Newline (behaves like cr followed by do)
OP	bool	(o)	Odd parity
os	bool		Terminal overstrikes
pb	num		Lowest baud where delays are required
pc	str		Pad character (default NUL)
pf	str		Turn off the printer
pk	str		Program function key n to type string s (terminfo only)
pl	str		Program function key n to execute string s (terminfo only)
pO	str	(N)	Turn on the printer for n bytes
po	str		Turn on the printer
ps	str		Print contents of the screen
pt	bool	(o)	Has hardware tabs (may need to be set with is)
px	str		Program function key n to transmit string s (terminfo only)
r1-r3	str		Reset terminal completely to sane modes (terminfo only)
rc	str	(P)	Restore cursor to position of last sc
rf	str		Name of file containing reset codes
RI	str	(NP)	Move cursor right n positions
rp	str	(NP*)	Repeat character c n times
rs	str		Reset terminal completely to sane modes (termcap only)
sa	str	(NP)	Define the video attributes
sc	str	(P)	Save cursor position
se	str		End standout mode
SF	str	(NP*)	Scroll forward n lines
sf	str	(P)	Scroll text up
sg	num		Number of garbage chars left by so or se (default 0)
so	str		Begin standout mode

SR	str	(NP*)	Scroll backward n lines
sr	str	(P)	Scroll text down
st	str		Set a tab in all rows, current column
ta	str	(P)	Tab to next 8-position hardware tab stop
tc	str		Entry of similar terminal - must be last
te	str		String to end programs that use termcap
ti	str		String to begin programs that use termcap
ts	str	(N)	Go to status line, column n
UC	bool	(o)	Upper-case only
uc	str		Underscore one character and move past it
ue	str		End underscore mode
ug	num		Number of garbage chars left by us or ue (default 0)
ul	bool		Underline character overstrikes
UP	str	(NP*)	Move cursor up n lines
up	str		Upline (cursor up)
us	str		Start underscore mode
vb	str		Visible bell (must not move cursor)
ve	str		Make cursor appear normal (undo vs/vi)
vi	str		Make cursor invisible
vs	str		Make cursor very visible
vt	num		Virtual terminal number (not supported on all systems)
wi	str	(N)	Set current window
ws	num		Number of columns in status line
xb	bool		Beehive (f1=ESC, f2=^C)
xn	bool		Newline ignored after 80 cols (Concept)
xo	bool		Terminal uses xoff/xon (DC3/DC1) handshaking
xr	bool	(o)	Return acts like ce cr nl (Delta Data)
xs	bool		Standout not erased by overwriting (Hewlett-Packard)
xt	bool		Tabs ruin, magic so char (Teleray 1061)
xx	bool	(o)	Tektronix 4025 insert-line

A Sample Entry

The following entry, which describes the Concept-100, is among the more complex entries in the termcap file as of this writing.

```
ca|concept100|c100|concept|c104|concept100-4p|HDS Concept-100:\
      :al=3*\E^R:am:bl=^G:cd=16*\E^C:ce=16\E^U:cl=2*^L:cm=\Ea%+ %+ :\
      :co#80:.cr=9^M:db:dc=16\E^A:dl=3*\E^B:do=^J:ei=\E\200:eo:im=\E^P:in:\
      :ip=16*:is=\EU\Ef\E7\E5\E8\El\ENH\EK\E\200\Eo&\200\Eo\47\E:k1=\E5:\
      :k2=\E6:k3=\E7:kb=^h:kd=\E<:ke=\Ex:kh=\E?:kl=\E>:kr=\E=:ks=\EX:\
      :ku=\E;:le=^H:li#24:mb=\EC:me=\EN\200:mh=\EE:mi:mk=\EH:mp=\EI:\
      :mr=\ED:nd=\E=:pb#9600:rp=0.2*\Er%.%+ :se=\Ed\Ee:sf=^J:so=\EE\ED:\
      :.ta=8\t:te=\Ev   \200\200\200\200\200\200\Ep\r\n:\
      :ti=\EU\Ev  8p\Ep\r:ue=\Eg:ul:up=\E;:us=\EG:\
      :vb=\Ek\200\200\200\200\200\200\200\200\200\200\200\200\200\200\EK:\
      :ve=\Ew:vs=\EW:vt#8:xn:\
      :bs:cr=^M:dC#9:dT#8:nl=^J:ta=^I:pt:
```

Entries may continue onto multiple lines by giving a \ as the last character of a line, and empty fields may be

included for readability (here between the last field on a line and the first field on the next). Comments may be included on lines beginning with "#".

Types of Capabilities

Capabilities in termcap are of three types: Boolean capabilities, which indicate particular features that the terminal has; numeric capabilities, giving the size of the display or the size of other attributes; and string capabilities, which give character sequences that can be used to perform particular terminal operations. All capabilities have two-letter codes. For instance, the fact that the Concept has automatic margins (i.e., an automatic return and linefeed when the end of a line is reached) is indicated by the Boolean capability am. Hence the description of the Concept includes am.

Numeric capabilities are followed by the character `#' then the value. In the example above co, which indicates the number of columns the display has, gives the value `80' for the Concept.

Finally, string-valued capabilities, such as ce (clear-to-end-of-line sequence) are given by the two-letter code, an `=', then a string ending at the next following `:'. A delay in milliseconds may appear after the `=' in such a capability, which causes padding characters to be supplied by tputs after the remainder of the string is sent to provide this delay. The delay can be either a number, e.g. `20', or a number followed by an `*', i.e., `3*'. An `*' indicates that the padding required is proportional to the number of lines affected by the operation, and the amount given is the per-affected-line padding required. (In the case of insert-character, the factor is still the number of lines affected; this is always 1 unless the terminal has in and the software uses it.) When an `*' is specified, it is sometimes useful to give a delay of the form `3.5' to specify a delay per line to tenths of milliseconds. (Only one decimal place is allowed.)

A number of escape sequences are provided in the string-valued capabilities for easy encoding of control characters there. \E maps to an ESC character, ^X maps to a control-X for any appropriate X, and the sequences \n \r \t \b \f map to linefeed, return, tab, backspace, and formfeed, respectively. Finally, characters may be given as three octal digits after a \, and the characters ^ and \ may be given as \^ and \\. If it is necessary to place a : in a capability it must be escaped in octal as \072. If it is necessary to place a NUL character in a string capability it must be encoded as \200. (The routines that deal with termcap use C

strings and strip the high bits of the output very late, so that a \200 comes out as a \000 would.)

Sometimes individual capabilities must be commented out. To do this, put a period before the capability name. For example, see the first cr and ta in the example above.

Preparing Descriptions

We now outline how to prepare descriptions of terminals. The most effective way to prepare a terminal description is by imitating the description of a similar terminal in termcap and to build up a description gradually, using partial descriptions with vi to check that they are correct. Be aware that a very unusual terminal may expose deficiencies in the ability of the termcap file to describe it or bugs in vi. To easily test a new terminal description you can set the environment variable TERMCAP to the absolute pathname of a file containing the description you are working on and programs will look there rather than in /etc/termcap. TERMCAP can also be set to the termcap entry itself to avoid reading the file when starting up a program.

To get the padding for insert-line right (if the terminal manufacturer did not document it), a severe test is to use vi to edit /etc/passwd at 9600 baud, delete roughly 16 lines from the middle of the screen, then hit the `u' key several times quickly. If the display messes up, more padding is usually needed. A similar test can be used for insert-character.

Basic Capabilities

The number of columns on each line of the display is given by the co numeric capability. If the display is a CRT, then the number of lines on the screen is given by the li capability. If the display wraps around to the beginning of the next line when the cursor reaches the right margin, then it should have the am capability. If the terminal can clear its screen, the code to do this is given by the cl string capability. If the terminal overstrikes (rather than clearing the position when a character is overwritten), it should have the os capability. If the terminal is a printing terminal, with no soft copy unit, give it both hc and os. (os applies to storage scope terminals, such as the Tektronix 4010 series, as well as to hard copy and APL terminals.) If there is a code to move the cursor to the left edge of the current row, give this as cr. (Normally this will be carriage-return, ^M.) If there is a code to produce an audible signal (bell, beep, etc.), give this as bl.

If there is a code (such as backspace) to move the cursor one position to the left, that capability should be given as le. Similarly, codes to move to the right, up, and down should be given as nd, up, and do, respectively. These local cursor motions should not alter the text they pass over; for example, you would not normally use "nd= " unless the terminal has the os capability, because the space would erase the character moved over.

A very important point here is that the local cursor motions encoded in termcap have undefined behavior at the left and top edges of a CRT display. Programs should never attempt to backspace around the left edge, unless bw is given, and never attempt to go up off the top using local cursor motions.

In order to scroll text up, a program goes to the bottom left corner of the screen and sends the sf (index) string. To scroll text down, a program goes to the top left corner of the screen and sends the sr (reverse index) string. The strings sf and sr have undefined behavior when not on their respective corners of the screen. Parameterized versions of the scrolling sequences are SF and SR, which have the same semantics as sf and sr except that they take one parameter and scroll that many lines. They also have undefined behavior except at the appropriate corner of the screen.

The am capability tells whether the cursor sticks at the right edge of the screen when text is output there, but this does not necessarily apply to nd from the last column. Leftward local motion is defined from the left edge only when bw is given; then an le from the left edge will move to the right edge of the previous row. This is useful for drawing a box around the edge of the screen, for example. If the terminal has switch-selectable automatic margins, the termcap description usually assumes that this feature is on, i.e., am. If the terminal has a command that moves to the first column of the next line, that command can be given as nw (newline). It is permissible for this to clear the remainder of the current line, so if the terminal has no correctly-working CR and LF it may still be possible to craft a working nw out of one or both of them.

These capabilities suffice to describe hardcopy and "glass-tty" terminals. Thus the Teletype model 33 is described as

```
T3|tty33|33|tty|Teletype model 33:\
     :bl=^G:co#72:cr=^M:do=^J:hc:os:
```

and the Lear Siegler ADM-3 is described as

```
l3|adm3|3|LSI ADM-3:\
     :am:bl=^G:cl=^Z:co#80:cr=^M:do=^J:le=^H:li#24:sf=^J:
```

Parameterized Strings

Cursor addressing and other strings requiring parameters are described by a parameterized string capability, with printf(3S)-like escapes %x in it, while other characters are passed through unchanged. For example, to address the cursor the cm capability is given, using two parameters: the row and column to move to. (Rows and columns are numbered from zero and refer to the physical screen visible to the user, not to any unseen memory. If the terminal has memory-relative cursor addressing, that can be indicated by an analogous CM capability.)

The % encodings have the following meanings:

```
%%   output `%'
%d   output value as in printf %d
%2   output value as in printf %2d
%3   output value as in printf %3d
%.   output value as in printf %c
%+x  add x to value, then do %.
%>xy if value > x then add y, no output
%r   reverse order of two parameters, no output
%i   increment by one, no output
%n   exclusive-or all parameters with 0140 (Datamedia 2500)
%B   BCD (16*(value/10)) + (value%10), no output
%D   Reverse coding (value - 2*(value%16)), no output (Delta Data)
```

Consider the Hewlett-Packard 2645, which, to get to row 3 and column 12, needs to be sent "\E&a12c03Y" padded for 6 milliseconds. Note that the order of the row and column coordinates is reversed here and that the row and column are sent as two-digit integers. Thus its cm capability is "cm=6\E&%r%2c%2Y".

The Microterm ACT-IV needs the current row and column sent simply encoded in binary preceded by a ^T, "cm=^T%.%.". Terminals that use "%." need to be able to backspace the cursor (le) and to move the cursor up one line on the screen (up). This is necessary because it is not always safe to transmit \n, ^D, and \r, as the system may change or discard them. (Programs using termcap must set terminal modes so that tabs are not expanded, so \t is safe to send. This turns out to be essential for the Ann Arbor 4080.)

A final example is the Lear Siegler ADM-3a, which offsets row and column by a blank character, thus "cm=\E=%+ %+ ".

Row or column absolute cursor addressing can be given as single parameter capabilities ch (horizontal position absolute) and cv (vertical position absolute). Sometimes these are shorter than the more general two-parameter sequence (as with the Hewlett-Packard 2645) and can be used in preference to cm. If there are parameterized local motions (e.g., move n positions to the right) these can be given as DO, LE, RI, and UP with a single parameter indicating how many positions to move. These are primarily useful if the terminal does not have cm, such as the Tektronix 4025.

Cursor Motions

If the terminal has a fast way to home the cursor (to the very upper left corner of the screen), this can be given as ho. Similarly, a fast way of getting to the lower left-hand corner can be given as ll; this may involve going up with up from the home position, but a program should never do this itself (unless ll does), because it can make no assumption about the effect of moving up from the home position. Note that the home position is the same as cursor address (0,0): to the top left corner of the screen, not of memory. (Therefore, the "\EH" sequence on Hewlett-Packard terminals cannot be used for ho.)

Area Clears

If the terminal can clear from the current position to the end of the line, leaving the cursor where it is, this should be given as ce. If the terminal can clear from the current position to the end of the display, this should be given as cd. cd must only be invoked from the first column of a line. (Therefore, it can be simulated by a request to delete a large number of lines, if a true cd is not available.)

Insert/Delete Line

If the terminal can open a new blank line before the line containing the cursor, this should be given as al; this must be invoked only from the first position of a line. The cursor must then appear at the left of the newly blank line. If the terminal can delete the line that the cursor is on, this should be given as dl; this must only be used from the first position on the line to be deleted. Versions of al and dl which take a single parameter and insert or delete that many lines can be given as AL and DL. If the terminal has a settable scrolling region (like the VT100), the command to set this can be described with the cs capability, which takes two parameters: the top and bottom lines of the scrolling region. The cursor position is, alas, undefined after using this command. It is possible to get the effect

of insert or delete line using this command - the sc and rc (save and restore cursor) commands are also useful. Inserting lines at the top or bottom of the screen can also be done using sr or sf on many terminals without a true insert/delete line, and is often faster even on terminals with those features.

If the terminal has the ability to define a window as part of memory which all commands affect, it should be given as the parameterized string wi. The four parameters are the starting and ending lines in memory and the starting and ending columns in memory, in that order. (This terminfo capability is described for completeness. It is unlikely that any termcap-using program will support it.)

If the terminal can retain display memory above the screen, then the da capability should be given; if display memory can be retained below, then db should be given. These indicate that deleting a line or scrolling may bring non-blank lines up from below or that scrolling back with sr may bring down non-blank lines.

Insert/Delete Character

There are two basic kinds of intelligent terminals with respect to insert/delete character that can be described using termcap. The most common insert/delete character operations affect only the characters on the current line and shift characters off the end of the line rigidly. Other terminals, such as the Concept-100 and the Perkin Elmer Owl, make a distinction between typed and untyped blanks on the screen, shifting upon an insert or delete only to an untyped blank on the screen which is either eliminated or expanded to two untyped blanks. You can determine the kind of terminal you have by clearing the screen then typing text separated by cursor motions. Type "abc def" using local cursor motions (not spaces) between the "abc" and the "def". Then position the cursor before the "abc" and put the terminal in insert mode. If typing characters causes the rest of the line to shift rigidly and characters to fall off the end, then your terminal does not distinguish between blanks and untyped positions. If the "abc" shifts over to the "def" which then move together around the end of the current line and onto the next as you insert, then you have the second type of terminal and should give the capability in, which stands for "insert null". While these are two logically separate attributes (one line vs. multi-line insert mode, and special treatment of untyped spaces), we have seen no terminals whose insert mode cannot be described with the single attribute.

Termcap can describe both terminals that have an insert mode and terminals that send a simple sequence to open a blank position on the current line. Give as im the sequence to get into insert mode. Give as ei the sequence to leave insert mode. Now give as ic any sequence that needs to be sent just before each character to be inserted. Most terminals with a true insert mode will not give ic; terminals that use a sequence to open a screen position should give it here. (If your terminal has both, insert mode is usually preferable to ic. Do not give both unless the terminal actually requires both to be used in combination.) If post-insert padding is needed, give this as a number of milliseconds in ip (a string option). Any other sequence that may need to be sent after insertion of a single character can also be given in ip. If your terminal needs to be placed into an `insert mode' and needs a special code preceding each inserted character, then both im/ei and ic can be given, and both will be used. The IC capability, with one parameter n, will repeat the effects of ic n times.

It is occasionally necessary to move around while in insert mode to delete characters on the same line (e.g., if there is a tab after the insertion position). If your terminal allows motion while in insert mode, you can give the capability mi to speed up inserting in this case. Omitting mi will affect only speed. Some terminals (notably Datamedia's) must not have mi because of the way their insert mode works.

Finally, you can specify dc to delete a single character, DC with one parameter n to delete n characters, and delete mode by giving dm and ed to enter and exit delete mode (which is any mode the terminal needs to be placed in for dc to work).

Highlighting, Underlining, and Visible Bells

If your terminal has one or more kinds of display attributes, these can be represented in a number of different ways. You should choose one display form as standout mode, representing a good high-contrast, easy-on-the-eyes format for highlighting error messages and other attention getters. (If you have a choice, reverse video plus half-bright is good, or reverse video alone.) The sequences to enter and exit standout mode are given as so and se, respectively. If the code to change into or out of standout mode leaves one or even two blank spaces or garbage characters on the screen, as the TVI 912 and Teleray 1061 do, then sg should be given to tell how many characters are left.

Codes to begin underlining and end underlining can be given as us and ue, respectively. Underline mode change garbage is specified by ug, similar to sg. If the terminal has a

code to underline the current character and move the cursor one position to the right, such as the Microterm Mime, this can be given as uc.

Other capabilities to enter various highlighting modes include mb (blinking), md (bold or extra bright), mh (dim or half-bright), mk (blanking or invisible text), mp (protected), mr (reverse video), me (turn off all attribute modes), as (enter alternate character set mode), and ae (exit alternate character set mode). Turning on any of these modes singly may or may not turn off other modes.

If there is a sequence to set arbitrary combinations of mode, this should be given as sa (set attributes), taking 9 parameters. Each parameter is either 0 or 1, as the corresponding attributes is on or off. The 9 parameters are, in order: standout, underline, reverse, blink, dim, bold, blank, protect, and alternate character set. Not all modes need be supported by sa, only those for which corresponding attribute commands exist. (It is unlikely that a termcap-using program will support this capability, which is defined for compatibility with terminfo.)

Terminals with the "magic cookie" glitches (sg and ug), rather than maintaining extra attribute bits for each character cell, instead deposit special "cookies", or "garbage characters", when they receive mode-setting sequences, which affect the display algorithm.

Some terminals, such as the Hewlett-Packard 2621, automatically leave standout mode when they move to a new line or when the cursor is addressed. Programs using standout mode should exit standout mode on such terminals before moving the cursor or sending a newline. On terminals where this is not a problem, the ms capability should be present to say that this overhead is unnecessary.

If the terminal has a way of flashing the screen to indicate an error quietly (a bell replacement), this can be given as vb; it must not move the cursor.

If the cursor needs to be made more visible than normal when it is not on the bottom line (to change, for example, a non-blinking underline into an easier-to-find block or blinking underline), give this sequence as vs. If there is a way to make the cursor completely invisible, give that as vi. The capability ve, which undoes the effects of both of these modes, should also be given.

If your terminal correctly displays underlined characters (with no special codes needed) even though it does not overstrike, then you should give the capability ul. If

overstrikes are erasable with a blank, this should be indicated by giving eo.

Keypad

If the terminal has a keypad that transmits codes when the keys are pressed, this information can be given. Note that it is not possible to handle terminals where the keypad only works in local mode (this applies, for example, to the unshifted Hewlett-Packard 2621 keys). If the keypad can be set to transmit or not transmit, give these codes as ks and ke. Otherwise the keypad is assumed to always transmit. The codes sent by the left-arrow, right-arrow, up-arrow, down-arrow, and home keys can be given as kl, kr, ku, kd, and kh, respectively. If there are function keys such as f0, f1, ..., f9, the codes they send can be given as k0, k1, k9. If these keys have labels other than the default f0 through f9, the labels can be given as l0, l1, l9. The codes transmitted by certain other special keys can be given: kH (home down), kb (backspace), ka (clear all tabs), kt (clear the tab stop in this column), kC (clear screen or erase), kD (delete character), kL (delete line), kM (exit insert mode), kE (clear to end of line), kS (clear to end of screen), kI (insert character or enter insert mode), kA (insert line), kN (next page), kP (previous page), kF (scroll forward/down), kR (scroll backward/up), and kT (set a tab stop in this column). In addition, if the keypad has a 3 by 3 array of keys including the four arrow keys, then the other five keys can be given as K1, K2, K3, K4, and K5. These keys are useful when the effects of a 3 by 3 directional pad are needed. The obsolete ko capability formerly used to describe "other" function keys has been completely supplanted by the above capabilities.

The ma entry is also used to indicate arrow keys on terminals that have single-character arrow keys. It is obsolete but still in use in version 2 of vi which must be run on some minicomputers due to memory limitations. This field is redundant with kl, kr, ku, kd, and kh. It consists of groups of two characters. In each group, the first character is what an arrow key sends, and the second character is the corresponding vi command. These commands are h for kl, j for kd, k for ku, l for kr, and H for kh. For example, the Mime would have "ma=^Hh^Kj^Zk^Xl" indicating arrow keys left (^H), down (^K), up (^Z), and right (^X). (There is no home key on the Mime.)

Tabs and Initialization

If the terminal needs to be in a special mode when running a program that uses these capabilities, the codes to enter and exit this mode can be given as ti and te. This arises, for

example, from terminals like the Concept with more than one page of memory. If the terminal has only memory-relative cursor addressing and not screen-relative cursor addressing, a screen-sized window must be fixed into the display for cursor addressing to work properly. This is also used for the Tektronix 4025, where ti sets the command character to be the one used by termcap.

Other capabilities include is, an initialization string for the terminal, and if, the name of a file containing long initialization strings. These strings are expected to set the terminal into modes consistent with the rest of the termcap description. They are normally sent to the terminal by the tset program each time the user logs in. They will be printed in the following order: is; setting tabs using ct and st; and finally if. (Terminfo uses i1-i2 instead of is and runs the program iP and prints i3 after the other initializations.) A pair of sequences that does a harder reset from a totally unknown state can be analogously given as rs and if. These strings are output by the reset program, which is used when the terminal gets into a wedged state. (Terminfo uses r1-r3 instead of rs.) Commands are normally placed in rs and rf only if they produce annoying effects on the screen and are not necessary when logging in. For example, the command to set the VT100 into 80-column mode would normally be part of is, but it causes an annoying glitch of the screen and is not normally needed since the terminal is usually already in 80-column mode.

If the terminal has hardware tabs, the command to advance to the next tab stop can be given as ta (usually ^I). A "backtab" command which moves leftward to the previous tab stop can be given as bt. By convention, if the terminal driver modes indicate that tab stops are being expanded by the computer rather than being sent to the terminal, programs should not use ta or bt even if they are present, since the user may not have the tab stops properly set. If the terminal has hardware tabs that are initially set every n positions when the terminal is powered up, then the numeric parameter it is given, showing the number of positions between tab stops. This is normally used by the tset command to determine whether to set the driver mode for hardware tab expansion, and whether to set the tab stops. If the terminal has tab stops that can be saved in nonvolatile memory, the termcap description can assume that they are properly set.

If there are commands to set and clear tab stops, they can be given as ct (clear all tab stops) and st (set a tab stop in the current column of every row). If a more complex sequence is needed to set the tabs than can be described by this, the sequence can be placed in is or if.

Delays

Certain capabilities control padding in the terminal driver. These are primarily needed by hardcopy terminals and are used by the tset program to set terminal driver modes appropriately. Delays embedded in the capabilities cr, sf, le, ff, and ta will cause the appropriate delay bits to be set in the terminal driver. If pb (padding baud rate) is given, these values can be ignored at baud rates below the value of pb. For 4.2BSD tset, the delays are given as numeric capabilities dC, dN, dB, dF, and dT instead.

Miscellaneous

If the terminal requires other than a NUL (zero) character as a pad, this can be given as pc. Only the first character of the pc string is used.

If the terminal has commands to save and restore the position of the cursor, give them as sc and rc.

If the terminal has an extra "status line" that is not normally used by software, this fact can be indicated. If the status line is viewed as an extra line below the bottom line, then the capability hs should be given. Special strings to go to a position in the status line and to return from the status line can be given as ts and fs. (fs must leave the cursor position in the same place that it was before ts. If necessary, the sc and rc strings can be included in ts and fs to get this effect.) The capability ts takes one parameter, which is the column number of the status line to which the cursor is to be moved. If escape sequences and other special commands such as tab work while in the status line, the flag es can be given. A string that turns off the status line (or otherwise erases its contents) should be given as ds. The status line is normally assumed to be the same width as the rest of the screen, i.e., co. If the status line is a different width (possibly because the terminal does not allow an entire line to be loaded), then its width in columns can be indicated with the numeric parameter ws.

If the terminal can move up or down half a line, this can be indicated with hu (half-line up) and hd (half-line down). This is primarily useful for superscripts and subscripts on hardcopy terminals. If a hardcopy terminal can eject to the next page (form feed), give this as ff (usually ^L).

If there is a command to repeat a given character a given number of times (to save time transmitting a large number of identical characters), this can be indicated with the parameterized string rp. The first parameter is the

character to be repeated and the second is the number of times to repeat it. (This is a terminfo feature that is unlikely to be supported by a program that uses termcap.)

If the terminal has a settable command character, such as the Tektronix 4025, this can be indicated with CC. A prototype command character is chosen which is used in all capabilities. This character is given in the CC capability to identify it. The following convention is supported on some UNIX systems: The environment is to be searched for a CC variable, and if found, all occurrences of the prototype character are replaced by the character in the environment variable. This use of the CC environment variable is a very bad idea, as it conflicts with make(1).

Terminal descriptions that do not represent a specific kind of known terminal, such as switch, dialup, patch, and network, should include the gn (generic) capability so that programs can complain that they do not know how to talk to the terminal. (This capability does not apply to virtual terminal descriptions for which the escape sequences are known.)

If the terminal uses xoff/xon (DC3/DC1) handshaking for flow control, give xo. Padding information should still be included so that routines can make better decisions about costs, but actual pad characters will not be transmitted.

If the terminal has a "meta key" which acts as a shift key, setting the 8th bit of any character transmitted, then this fact can be indicated with km. Otherwise, software will assume that the 8th bit is parity and it will usually be cleared. If strings exist to turn this "meta mode" on and off, they can be given as mm and mo.

If the terminal has more lines of memory than will fit on the screen at once, the number of lines of memory can be indicated with lm. An explicit value of 0 indicates that the number of lines is not fixed, but that there is still more memory than fits on the screen.

If the terminal is one of those supported by the UNIX system virtual terminal protocol, the terminal number can be given as vt.

Media copy strings which control an auxiliary printer connected to the terminal can be given as ps: print the contents of the screen; pf: turn off the printer; and po: turn on the printer. When the printer is on, all text sent to the terminal will be sent to the printer. It is undefined whether the text is also displayed on the terminal screen when the printer is on. A variation pO takes one parameter

and leaves the printer on for as many characters as the value of the parameter, then turns the printer off. The parameter should not exceed 255. All text, including pf, is transparently passed to the printer while pO is in effect.

Strings to program function keys can be given as pk, pl, and px. Each of these strings takes two parameters: the function key number to program (from 0 to 9) and the string to program it with. Function key numbers out of this range may program undefined keys in a terminal-dependent manner. The differences among the capabilities are that pk causes pressing the given key to be the same as the user typing the given string; pl causes the string to be executed by the terminal in local mode; and px causes the string to be transmitted to the computer. Unfortunately, due to lack of a definition for string parameters in termcap, only terminfo supports these capabilities.

Glitches and Braindamage

Hazeltine terminals, which do not allow `~' characters to be displayed, should indicate hz.

The nc capability, now obsolete, formerly indicated Datamedia terminals, which echo \r \n for carriage return then ignore a following linefeed.

Terminals that ignore a linefeed immediately after an am wrap, such as the Concept, should indicate xn.

If ce is required to get rid of standout (instead of merely writing normal text on top of it), xs should be given.

Teleray terminals, where tabs turn all characters moved over to blanks, should indicate xt (destructive tabs). This glitch is also taken to mean that it is not possible to position the cursor on top of a "magic cookie", and that to erase standout mode it is necessary to use delete and insert line.

The Beehive Superbee, which is unable to correctly transmit the ESC or ^C characters, has xb, indicating that the "f1" key is used for ESC and "f2" for ^C. (Only certain Superbees have this problem, depending on the ROM.)

Other specific terminal problems may be corrected by adding more capabilities of the form xx.

Similar Terminals

If there are two very similar terminals, one can be defined as being just like the other with certain exceptions. The

string capability tc can be given with the name of the similar terminal. This capability must be last, and the combined length of the entries must not exceed 1024. The capabilities given before tc override those in the terminal type invoked by tc. A capability can be canceled by placing xx@ to the left of the tc invocation, where xx is the capability. For example, the entry

```
hn|2621-nl:ks@:ke@:tc=2621:
```

defines a "2621-nl" that does not have the ks or ke capabilities, hence does not turn on the function key labels when in visual mode. This is useful for different modes for a terminal, or for different user preferences.

AUTHOR

William Joy
Mark Horton added underlining and keypad support

FILES

/etc/termcap file containing terminal descriptions

SEE ALSO

ex(1), more(1), tset(1), ul(1), vi(1), curses(3X), printf(3S), term(7).

CAVEATS AND BUGS

Note: termcap was replaced by terminfo in UNIX System V Release 2.0. The transition will be relatively painless if capabilities flagged as "obsolete" are avoided.

Lines and columns are now stored by the kernel as well as in the termcap entry. Most programs now use the kernel information primarily; the information in this file is used only if the kernel does not have any information.

Vi allows only 256 characters for string capabilities, and the routines in termlib(3) do not check for overflow of this buffer. The total length of a single entry (excluding only escaped newlines) may not exceed 1024.

Not all programs support all entries.

NAME

tp - DEC/mag tape formats

DESCRIPTION

Tp dumps files to and extracts files from DECtape and magtape. The formats of these tapes are the same except that magtapes have larger directories.

Block zero contains a copy of a stand-alone bootstrap program. See reboot(8).

Blocks 1 through 24 for DECtape (1 through 62 for magtape) contain a directory of the tape. There are 192 (resp. 496) entries in the directory; 8 entries per block; 64 bytes per entry. Each entry has the following format:

```
struct {
        char     pathname[32];
        unsigned short  mode;
        char     uid;
        char     gid;
        char     unused1;
        char     size[3];
        long     modtime;
        unsigned short  tapeaddr;
        char     unused2[16];
        unsigned short  checksum;
};
```

The path name entry is the path name of the file when put on the tape. If the pathname starts with a zero word, the entry is empty. It is at most 32 bytes long and ends in a null byte. Mode, uid, gid, size and time modified are the same as described under i-nodes (see file system fs(5)). The tape address is the tape block number of the start of the contents of the file. Every file starts on a block boundary. The file occupies (size+511)/512 blocks of continuous tape. The checksum entry has a value such that the sum of the 32 words of the directory entry is zero.

Blocks above 25 (resp. 63) are available for file storage.

A fake entry has a size of zero.

SEE ALSO

fs(5), tp(1)

BUGS

The pathname, uid, gid, and size fields are too small.

NAME

ttys - terminal initialization data

DESCRIPTION

The ttys file contains information that is used by various routines to initialize and control the use of terminal special files. This information is read with the getttyent(3) library routines. There is one line in the ttys file per special file. Fields are separated by tabs and/or spaces. Some fields may contain more than one word and should be enclosed in double quotes. Blank lines and comments can appear anywhere in the file; comments are delimited by `#' and new line. Unspecified fields default to null. The first field is the terminal's entry in the device directory, /dev. The second field of the file is the command to execute for the line, typically getty(8), which performs such tasks as baud-rate recognition, reading the login name, and calling login(1). It can be, however, any desired command, for example the start up for a window system terminal emulator or some other daemon process, and can contain multiple words if quoted. The third field is the type of terminal normally connected to that tty line, as found in the termcap(5) data base file. The remaining fields set flags in the ty_status entry (see getttyent(3)) or specify a window system process that init(8) will maintain for the terminal line. As flag values, the strings `on' and `off' specify whether init should execute the command given in the second field, while `secure' in addition to `on' allows root to login on this line. These flag fields should not be quoted. The string `window=' is followed by a quoted command string which init will execute before starting getty. If the line ends in a comment, the comment is included in the ty_comment field of the ttyent structure.

Some examples:

```
console "/usr/libexec/getty std.1200"   vt100    on secure
ttyd0   "/usr/libexec/getty d1200"      dialup   on   # 555-1234
ttyh0   "/usr/libexec/getty std.9600"   hp2621-nl      on   # 254MC
ttyh1   "/usr/libexec/getty std.9600"   plugboard     on   # John's office
ttyp0   none                  network
ttyp1   none                  network       off
ttyv0   "/usr/new/xterm -L :0"     vs100    on window="/usr/new/Xvs100 0"
```

The first example permits root login on the console at 1200 baud, the second allows dialup at 1200 baud without root login, the third and fourth allow login at 9600 baud with terminal types of "hp2621-nl" and "plugboard" respectively, the fifth and sixth line are examples of network pseudo ttys, which should not have getty enabled on them, and the last example shows a terminal emulator and window system startup entry.

FILES
/etc/ttys

SEE ALSO
login(1), getttyent(3), gettytab(5), init(8), getty(8)

NAME

types - primitive system data types

SYNOPSIS

```
#include <sys/types.h>
```

DESCRIPTION

The data types defined in the include file are used in UNIX system code; some data of these types are accessible to user code:

```
/*
 * Copyright (c) 1986 Regents of the University of California.
 * All rights reserved.  The Berkeley software License Agreement
 * specifies the terms and conditions for redistribution.
 *
 *       @(#)types.h    1.4.1 (2.11BSD) 2000/2/28
 */

#ifndef _TYPES_
#define  _TYPES_

/*
 * Basic system types and major/minor device constructing/busting macros.
 */

/* major part of a device */
#define  major(x) ((int)(((int)(x)>>8)&0377))

/* minor part of a device */
#define  minor(x) ((int)((x)&0377))

/* make a device number */
#define  makedev(x,y)   ((dev_t)(((x)<<8) | (y)))

typedef  unsigned char  u_char;
typedef  unsigned short u_short;
typedef  unsigned int   u_int;
typedef unsigned long   u_long;/* see this! unsigned longs at last! */
typedef  unsigned short ushort;/* sys III compat */

#ifdef pdp11
typedef  struct   _physadr { short r[1]; } *physadr;
typedef  struct   label_t {
       int  val[8]; /* regs 2-7, __ovno and super SP */
} label_t;
#endif
typedef  struct   _quad { long val[2]; } quad;
typedef  long     daddr_t;
typedef  char *   caddr_t;
typedef  u_short  ino_t;
typedef  long     swblk_t;
```

```
typedef  u_int    size_t;
typedef  intssize_t;
typedef  long     time_t;
typedef  short    dev_t;
typedef  long     off_t;
typedef  u_short  uid_t;
typedef  u_short  gid_t;
typedef  intpid_t;
typedef  u_short  mode_t;

#define  NBBY     8     /* number of bits in a byte */

#ifndef howmany
#define  howmany(x, y)  (((x)+((y)-1))/(y))
#endif

#include <sys/select.h>

typedef charbool_t; /* boolean */
typedef u_int     memaddr;/* core or swap address */
typedef longubadr_t;/* unibus address */

#endif
```

The form daddr_t is used for disk addresses except in an i-node on disk, see fs(5). Times are encoded in seconds since 00:00:00 GMT, January 1, 1970. The major and minor parts of a device code specify kind and unit number of a device and are installation-dependent. Offsets are measured in bytes from the beginning of a file. The label_t variables are used to save the processor state while another process is running.

SEE ALSO

fs(5), time(3), lseek(2), adb(1)

NAME

tzfile - time zone information

SYNOPSIS

```
#include <tzfile.h>
```

DESCRIPTION

The time zone information files used by tzset(3) begin with bytes reserved for future use, followed by three four-byte values of type long, written in a ``standard'' byte order (the high-order byte of the value is written first). These values are, in order:

tzh_timecnt
The number of "transition times" for which data is stored in the file.

tzh_typecnt
The number of "local time types" for which data is stored in the file (must not be zero).

tzh_charcnt
The number of characters of "time zone abbreviation strings" stored in the file.

The above header is followed by tzh_timecnt four-byte values of type long, sorted in ascending order. These values are written in ``standard'' byte order. Each is used as a transition time (as returned by time(2)) at which the rules for computing local time change. Next come tzh_timecnt one-byte values of type unsigned char; each one tells which of the different types of ``local time'' types described in the file is associated with the same-indexed transition time. These values serve as indices into an array of ttinfo structures that appears next in the file; these structures are defined as follows:

```
struct ttinfo {
     long     tt_gmtoff;
     int      tt_isdst;
     unsigned int  tt_abbrind;
};
```

Each structure is written as a four-byte value for tt_gmtoff of type long, in a standard byte order, followed by a one-byte value for tt_isdst and a one-byte value for tt_abbrind. In each structure, tt_gmtoff gives the number of seconds to be added to GMT, tt_isdst tells whether tm_isdst should be set by localtime (3) and tt_abbrind serves as an index into the array of time zone abbreviation characters that follow the ttinfo structure(s) in the file.

Localtime uses the first standard-time ttinfo structure in the file (or simply the first ttinfo structure in the absence of a standard-time structure) if either tzh_timecnt is zero or the time argument is less than the first transition time recorded in the file.

SEE ALSO

ctime(3)

NAME

USERFILE - UUCP pathname permissions file

DESCRIPTION

The USERFILE file specifies the file system directory trees that are accessible to local users and to remote systems via UUCP.

Each line in USERFILE is of the form:

[loginname],[system] [c] pathname [pathname] [pathname]

The first two items are separated by a comma; any number of spaces or tabs may separate the remaining items. Lines beginning with a `#' character are comments. A trailing `\' indicates that the next line is a continuation of the current line.

Loginname is a login (from /etc/passwd) on the local machine.

System is the name of a remote machine, the same name used in L.sys(5).

c denotes the optional callback field. If a c appears here, a remote machine that calls in will be told that callback is requested, and the conversation will be terminated. The local system will then immediately call the remote host back.

Pathname is a pathname prefix that is permissible for this login and/or system.

When uucico(8) runs in master role or uucp(1) or uux(1) are run by local users, the permitted pathnames are those on the first line with a loginname that matches the name of the user who executed the command. If no such line exists, then the first line with a null (missing) loginname field is used. (Beware: uucico is often run by the superuser or the UUCP administrator through cron(8).)

When uucico runs in slave role, the permitted pathnames are those on the first line with a system field that matches the hostname of the remote machine. If no such line exists, then the first line with a null (missing) system field is used.

Uuxqt(8) works differently; it knows neither a login name nor a hostname. It accepts the pathnames on the first line that has a null system field. (This is the same line that is used by uucico when it cannot match the remote machine's hostname.)

A line with both loginname and system null, for example

, /usr/spool/uucppublic

can be used to conveniently specify the paths for both "no match" cases if lines earlier in USERFILE did not define them. (This differs from older Berkeley and all USG versions, where each case must be individually specified. If neither case is defined earlier, a "null" line only defines the "unknown login" case.)

To correctly process loginname on systems that assign several logins per UID, the following strategy is used to determine the current loginname:

1) If the process is attached to a terminal, a login entry exists in /var/run/utmp, and the UID for the utmp name matches the current real UID, then loginname is set to the utmp name.

2) If the USER environment variable is defined and the UID for this name matches the current real UID, then loginname is set to the name in USER.

3) If both of the above fail, call getpwuid(3) to fetch the first name in /etc/passwd that matches the real UID.

4) If all of the above fail, the utility aborts.

FILES

/etc/uucp/USERFILE
/etc/uucp/UUAIDS/USERFILE USERFILE example

SEE ALSO

uucp(1), uux(1), L.cmds(5), L.sys(5), uucico(8), uuxqt(8)

NOTES

The UUCP utilities (uucico, uucp, uux, and uuxqt) always have access to the UUCP spool files in /usr/spool/uucp, regardless of pathnames in USERFILE.

If uucp is listed in L.cmds(5), then a remote system will execute uucp on the local system with the USERFILE privileges for its login, not its hostname.

Uucico freely switches between master and slave roles during the course of a conversation, regardless of the role it was started with. This affects how USERFILE is interpreted.

WARNING

USERFILE restricts access only on strings that the UUCP

utilities identify as being pathnames. If the wrong holes are left in other UUCP control files (notably L.cmds), it can be easy for an intruder to open files anywhere in the file system. Arguments to uucp(1) are safe, since it assumes all of its non-option arguments are files. Uux(1) cannot make such assumptions; hence, it is more dangerous.

BUGS

The UUCP Implementation Description explicitly states that all remote login names must be listed in USERFILE. This requirement is not enforced by Berkeley UUCP, although it is by USG UUCP.

Early versions of 4.2BSD uuxqt(8) erroneously check UUCP spool files against the USERFILE pathname permissions. Hence, on these systems it is necessary to specify /usr/spool/uucp as a valid path on the USERFILE line used by uuxqt. Otherwise, all uux(1) requests are rejected with a "PERMISSION DENIED" message.

NAME

utmp, wtmp - login records

SYNOPSIS

```
#include <utmp.h>
```

DESCRIPTION

The utmp file records information about who is currently using the system. The file is a sequence of entries with the following structure declared in the include file:

```
/*
 * Copyright (c) 1988 The Regents of the University of California.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms are permitted
 * provided that the above copyright notice and this paragraph are
 * duplicated in all such forms and that any documentation,
 * advertising materials, and other materials related to such
 * distribution and use acknowledge that the software was developed
 * by the University of California, Berkeley.  The name of the
 * University may not be used to endorse or promote products derived
 * from this software without specific prior written permission.
 * THIS SOFTWARE IS PROVIDED ``AS IS'' AND WITHOUT ANY EXPRESS OR
 * IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED
 * WARRANTIES OF MERCHANTIBILITY AND FITNESS FOR A PARTICULAR PURPOSE.
 *
 *   @(#)utmp.h     5.6.1 (2.11BSD) 1996/11/27
 */

#define   _PATH_UTMP    "/var/run/utmp"
#define   _PATH_WTMP    "/usr/adm/wtmp"

#define   UT_NAMESIZE    15
#define   UT_LINESIZE    8
#define   UT_HOSTSIZE    16
struct utmp {
     char ut_line[UT_LINESIZE];
     char ut_name[UT_NAMESIZE];
     char ut_host[UT_HOSTSIZE];
     long ut_time;
};
```

This structure gives the name of the special file associated with the user's terminal, the user's login name, and the time of the login in the form of time(3C).

The wtmp file records all logins and logouts. A null user name indicates a logout on the associated terminal. Furthermore, the terminal name `~' indicates that the system was rebooted at the indicated time; the adjacent pair of entries with terminal names `|' and `{' indicate the system-

maintained time just before and just after a date command has changed the system's idea of the time.

Wtmp is maintained by login(1) and init(8). Neither of these programs creates the file, so if it is removed record-keeping is turned off. It is summarized by ac(8).

FILES

/var/run/utmp
/usr/adm/wtmp

SEE ALSO

login(1), init(8), who(1), ac(8)

NAME

uuencode - format of an encoded uuencode file

DESCRIPTION

Files output by uuencode(1C) consist of a header line, followed by a number of body lines, and a trailer line. Uudecode(1C) will ignore any lines preceding the header or following the trailer. Lines preceding a header must not, of course, look like a header.

The header line is distinguished by having the first 6 characters "begin ". The word begin is followed by a mode (in octal), and a string which names the remote file. A space separates the three items in the header line.

The body consists of a number of lines, each at most 62 characters long (including the trailing newline). These consist of a character count, followed by encoded characters, followed by a newline. The character count is a single printing character, and represents an integer, the number of bytes the rest of the line represents. Such integers are always in the range from 0 to 63 and can be determined by subtracting the character space (octal 40) from the character.

Groups of 3 bytes are stored in 4 characters, 6 bits per character. All are offset by a space to make the characters printing. The last line may be shorter than the normal 45 bytes. If the size is not a multiple of 3, this fact can be determined by the value of the count on the last line. Extra garbage will be included to make the character count a multiple of 4. The body is terminated by a line with a count of zero. This line consists of one ASCII space.

The trailer line consists of "end" on a line by itself.

SEE ALSO

uuencode(1C), uudecode(1C), uusend(1C), uucp(1C), mail(1)

NAME

vfont - font formats for the Benson-Varian or Versatec

SYNOPSIS

/usr/share/vfont/*

DESCRIPTION

The fonts for the printer/plotters have the following format. Each file contains a header, an array of 256 character description structures, and then the bit maps for the characters themselves. The header has the following format:

```
struct header {
      short           magic;
      unsigned short  size;
      short           maxx;
      short           maxy;
      short           xtnd;
} header;
```

The magic number is 0436 (octal). The maxx, maxy, and xtnd fields are not used at the current time. Maxx and maxy are intended to be the maximum horizontal and vertical size of any glyph in the font, in raster lines. The size is the size of the bit maps for the characters in bytes. Before the maps for the characters is an array of 256 structures for each of the possible characters in the font. Each element of the array has the form:

```
struct dispatch {
      unsigned short  addr;
      short           nbytes;
      char            up;
      char            down;
      char            left;
      char            right;
      short           width;
};
```

The nbytes field is nonzero for characters which actually exist. For such characters, the addr field is an offset into the rest of the file where the data for that character begins. There are up+down rows of data for each character, each of which has left+right bits, rounded up to a number of bytes. The width field is not used by vcat, although it is to make width tables for troff. It represents the logical width of the glyph, in raster lines, and shows where the base point of the next glyph would be.

FILES

/usr/share/vfont/*

SEE ALSO
troff(1), pti(1), vfontinfo(1)

NAME

vgrindefs - vgrind's language definition data base

SYNOPSIS

/usr/share/misc/vgrindefs

DESCRIPTION

Vgrindefs contains all language definitions for vgrind. The data base is very similar to termcap(5).

FIELDS

The following table names and describes each field.

Name	Type	Description
pb	str	regular expression for start of a procedure
bb	str	regular expression for start of a lexical block
be	str	regular expression for the end of a lexical block
cb	str	regular expression for the start of a comment
ce	str	regular expression for the end of a comment
sb	str	regular expression for the start of a string
se	str	regular expression for the end of a string
lb	str	regular expression for the start of a character constant
le	str	regular expression for the end of a character constant
tl	bool	present means procedures are only defined at the top lexical level
oc	bool	present means upper and lower case are equivalent
kw	str	a list of keywords separated by spaces

Example

The following entry, which describes the C language, is typical of a language entry.

```
C|c:   :pb=^\d?*?\d?\p\d??):bb={:be=}:cb=/*:ce=*/:sb=":se=\e":\
     :lb=':le=\e':tl:\
     :kw=asm auto break case char continue default do double else enum\
     extern float for fortran goto if int long register return short\
     sizeof static struct switch typedef union unsigned while #define\
     #else #endif #if #ifdef #ifndef #include #undef # define else endif\
     if ifdef ifndef include undef:
```

Note that the first field is just the language name (and any variants of it). Thus the C language could be specified to vgrind(1) as "c" or "C".

Entries may continue onto multiple lines by giving a \ as the last character of a line. Capabilities in vgrindefs are of two types: Boolean capabilities which indicate that the language has some particular feature and string capabilities which give a regular expression or keyword list.

REGULAR EXPRESSIONS

Vgrindefs uses regular expression which are very similar to those of ex(1) and lex(1). The characters `^', `$', `:' and `\' are reserved characters and must be "quoted" with a preceding \ if they are to be included as normal characters. The metasymbols and their meanings are:

$ the end of a line

^ the beginning of a line

\d a delimiter (space, tab, newline, start of line)

\a matches any string of symbols (like .* in lex)

\p matches any alphanumeric name. In a procedure definition (pb) the string that matches this symbol is used as the procedure name.

() grouping

| alternation

? last item is optional

\e preceding any string means that the string will not match an input string if the input string is preceded by an escape character (\). This is typically used for languages (like C) which can include the string delimiter in a string b escaping it.

Unlike other regular expressions in the system, these match words and not characters. Hence something like "(tramp|steamer)flies?" would match "tramp", "steamer", "trampflies", or "steamerflies".

KEYWORD LIST

The keyword list is just a list of keywords in the language separated by spaces. If the "oc" boolean is specified, indicating that upper and lower case are equivalent, then all the keywords should be specified in lower case.

FILES

/usr/share/misc/vgrindefs file containing terminal descriptions

SEE ALSO

vgrind(1), troff(1)

AUTHOR
Dave Presotto

BUGS