NAME

intro — miscellaneous information pages

DESCRIPTION

This section contains miscellaneous documentation, including information on troff(1) macro packages.

ascii map of ASCII character set

environ user environment

hier file system hierarchy

hostname host name resolution description

mailaddr mail addressing description

mdoc macros for typesetting **-mdoc** style manual pages

mdoc.samples
tutorial for writing BSD manuals with **-mdoc**

operator C operator precedence and order of evaluation

release layout of NetBSD releases and snapshots

script how interpreter scripts are executed

signal available signals under NetBSD

sticky sticky bit (S_ISVTX) handling

symlink symbolic link handling

HISTORY

intro appeared in 4.2BSD.

NAME

ascii — octal, hexadecimal and decimal ASCII character sets

DESCRIPTION

The **octal** set:

000	nul	001	soh	002	stx	003	etx	004	eot	005	enq	006	ack	007	bel
010	bs	011	ht	012	nl	013	vt	014	np	015	cr	016	so	017	si
020	dle	021	dc1	022	dc2	023	dc3	024	dc4	025	nak	026	syn	027	etb
030	can	031	em	032	sub	033	esc	034	fs	035	gs	036	rs	037	us
040	sp	041	!	042	"	043	#	044	$	045	%	046	&	047	’
050	(	051	)	052	*	053	+	054	,	055	-	056	.	057	/
060	0	061	1	062	2	063	3	064	4	065	5	066	6	067	7
070	8	071	9	072	:	073	;	074	<	075	=	076	>	077	?
100	@	101	A	102	B	103	C	104	D	105	E	106	F	107	G
110	H	111	I	112	J	113	K	114	L	115	M	116	N	117	O
120	P	121	Q	122	R	123	S	124	T	125	U	126	V	127	W
130	X	131	Y	132	Z	133	[	134	\	135	]	136	^	137	_
140	‘	141	a	142	b	143	c	144	d	145	e	146	f	147	g
150	h	151	i	152	j	153	k	154	l	155	m	156	n	157	o
160	p	161	q	162	r	163	s	164	t	165	u	166	v	167	w
170	x	171	y	172	z	173	{	174	|	175	}	176	~	177	del

The **hexadecimal** set:

00	nul	01	soh	02	stx	03	etx	04	eot	05	enq	06	ack	07	bel
08	bs	09	ht	0a	nl	0b	vt	0c	np	0d	cr	0e	so	0f	si
10	dle	11	dc1	12	dc2	13	dc3	14	dc4	15	nak	16	syn	17	etb
18	can	19	em	1a	sub	1b	esc	1c	fs	1d	gs	1e	rs	1f	us
20	sp	21	!	22	"	23	#	24	$	25	%	26	&	27	’
28	(	29	)	2a	*	2b	+	2c	,	2d	-	2e	.	2f	/
30	0	31	1	32	2	33	3	34	4	35	5	36	6	37	7
38	8	39	9	3a	:	3b	;	3c	<	3d	=	3e	>	3f	?
40	@	41	A	42	B	43	C	44	D	45	E	46	F	47	G
48	H	49	I	4a	J	4b	K	4c	L	4d	M	4e	N	4f	O
50	P	51	Q	52	R	53	S	54	T	55	U	56	V	57	W
58	X	59	Y	5a	Z	5b	[	5c	\	5d	]	5e	^	5f	_
60	`	61	a	62	b	63	c	64	d	65	e	66	f	67	g
68	h	69	i	6a	j	6b	k	6c	l	6d	m	6e	n	6f	o
70	p	71	q	72	r	73	s	74	t	75	u	76	v	77	w
78	x	79	y	7a	z	7b	{	7c	|	7d	}	7e	~	7f	del

The **decimal** set:

0	nul	1	soh	2	stx	3	etx	4	eot	5	enq	6	ack	7	bel
8	bs	9	ht	10	nl	11	vt	12	np	13	cr	14	so	15	si
16	dle	17	dc1	18	dc2	19	dc3	20	dc4	21	nak	22	syn	23	etb
24	can	25	em	26	sub	27	esc	28	fs	29	gs	30	rs	31	us
32	sp	33	!	34	"	35	#	36	$	37	%	38	&	39	’
40	(	41	)	42	*	43	+	44	,	45	-	46	.	47	/
48	0	49	1	50	2	51	3	52	4	53	5	54	6	55	7
56	8	57	9	58	:	59	;	60	<	61	=	62	>	63	?
64	@	65	A	66	B	67	C	68	D	69	E	70	F	71	G
72	H	73	I	74	J	75	K	76	L	77	M	78	N	79	O

```
 80  P    81  Q    82  R    83  S    84  T    85  U    86  V    87  W
 88  X    89  Y    90  Z    91  [    92  \    93  ]    94  ^    95  _
 96  ‘    97  a    98  b    99  c   100  d   101  e   102  f   103  g
104  h   105  i   106  j   107  k   108  l   109  m   110  n   111  o
112  p   113  q   114  r   115  s   116  t   117  u   118  v   119  w
120  x   121  y   122  z   123  {   124  |   125  }   126  ~   127 del
```

FILES

`/usr/share/misc/ascii`

HISTORY

An **ascii** manual page appeared in Version 7 AT&T UNIX.

NAME

des_modes − the variants of DES and other crypto algorithms of OpenSSL

DESCRIPTION

Several crypto algorithms for OpenSSL can be used in a number of modes. Those are used for using block ciphers in a way similar to stream ciphers, among other things.

OVERVIEW

Electronic Codebook Mode (ECB)

Normally, this is found as the function *algorithm_ecb_encrypt()*.

- 64 bits are enciphered at a time.
- The order of the blocks can be rearranged without detection.
- The same plaintext block always produces the same ciphertext block (for the same key) making it vulnerable to a 'dictionary attack'.
- An error will only affect one ciphertext block.

Cipher Block Chaining Mode (CBC)

Normally, this is found as the function *algorithm_cbc_encrypt()*. Be aware that *des_cbc_encrypt()* is not really DES CBC (it does not update the IV); use *des_ncbc_encrypt()* instead.

- a multiple of 64 bits are enciphered at a time.
- The CBC mode produces the same ciphertext whenever the same plaintext is encrypted using the same key and starting variable.
- The chaining operation makes the ciphertext blocks dependent on the current and all preceding plaintext blocks and therefore blocks can not be rearranged.
- The use of different starting variables prevents the same plaintext enciphering to the same ciphertext.
- An error will affect the current and the following ciphertext blocks.

Cipher Feedback Mode (CFB)

Normally, this is found as the function *algorithm_cfb_encrypt()*.

- a number of bits (j) <= 64 are enciphered at a time.
- The CFB mode produces the same ciphertext whenever the same plaintext is encrypted using the same key and starting variable.
- The chaining operation makes the ciphertext variables dependent on the current and all preceding variables and therefore j−bit variables are chained together and can not be rearranged.
- The use of different starting variables prevents the same plaintext enciphering to the same ciphertext.
- The strength of the CFB mode depends on the size of k (maximal if j == k). In my implementation this is always the case.
- Selection of a small value for j will require more cycles through the encipherment algorithm per unit of plaintext and thus cause greater processing overheads.
- Only multiples of j bits can be enciphered.
- An error will affect the current and the following ciphertext variables.

Output Feedback Mode (OFB)

Normally, this is found as the function *algorithm_ofb_encrypt()*.

- a number of bits (j) <= 64 are enciphered at a time.
- The OFB mode produces the same ciphertext whenever the same plaintext enciphered using the same key and starting variable. More over, in the OFB mode the same key stream is produced when the same key

and start variable are used. Consequently, for security reasons a specific start variable should be used only once for a given key.

- The absence of chaining makes the OFB more vulnerable to specific attacks.
- The use of different start variables values prevents the same plaintext enciphering to the same ciphertext, by producing different key streams.
- Selection of a small value for j will require more cycles through the encipherment algorithm per unit of plaintext and thus cause greater processing overheads.
- Only multiples of j bits can be enciphered.
- OFB mode of operation does not extend ciphertext errors in the resultant plaintext output. Every bit error in the ciphertext causes only one bit to be in error in the deciphered plaintext.
- OFB mode is not self−synchronizing. If the two operation of encipherment and decipherment get out of synchronism, the system needs to be re−initialized.
- Each re-initialization should use a value of the start variable different from the start variable values used before with the same key. The reason for this is that an identical bit stream would be produced each time from the same parameters. This would be susceptible to a 'known plaintext' attack.

Triple ECB Mode

Normally, this is found as the function *algorithm_ecb3_encrypt()*.

- Encrypt with key1, decrypt with key2 and encrypt with key3 again.
- As for ECB encryption but increases the key length to 168 bits. There are theoretic attacks that can be used that make the effective key length 112 bits, but this attack also requires 2^56 blocks of memory, not very likely, even for the NSA.
- If both keys are the same it is equivalent to encrypting once with just one key.
- If the first and last key are the same, the key length is 112 bits. There are attacks that could reduce the effective key strength to only slightly more than 56 bits, but these require a lot of memory.
- If all 3 keys are the same, this is effectively the same as normal ecb mode.

Triple CBC Mode

Normally, this is found as the function *algorithm_ede3_cbc_encrypt()*.

- Encrypt with key1, decrypt with key2 and then encrypt with key3.
- As for CBC encryption but increases the key length to 168 bits with the same restrictions as for triple ecb mode.

NOTES

This text was been written in large parts by Eric Young in his original documentation for SSLeay, the predecessor of OpenSSL. In turn, he attributed it to:

```
AS 2805.5.2
Australian Standard
Electronic funds transfer - Requirements for interfaces,
Part 5.2: Modes of operation for an n-bit block cipher algorithm
Appendix A
```

SEE ALSO

blowfish (3), *des* (3), *idea* (3), *rc2* (3)

NAME

Modes of DES − the variants of DES and other crypto algorithms of OpenSSL

DESCRIPTION

Several crypto algorithms for OpenSSL can be used in a number of modes. Those are used for using block ciphers in a way similar to stream ciphers, among other things.

OVERVIEW

Electronic Codebook Mode (ECB)

Normally, this is found as the function *algorithm_ecb_encrypt()*.

- 64 bits are enciphered at a time.
- The order of the blocks can be rearranged without detection.
- The same plaintext block always produces the same ciphertext block (for the same key) making it vulnerable to a 'dictionary attack'.
- An error will only affect one ciphertext block.

Cipher Block Chaining Mode (CBC)

Normally, this is found as the function *algorithm_cbc_encrypt()*. Be aware that *des_cbc_encrypt()* is not really DES CBC (it does not update the IV); use *des_ncbc_encrypt()* instead.

- a multiple of 64 bits are enciphered at a time.
- The CBC mode produces the same ciphertext whenever the same plaintext is encrypted using the same key and starting variable.
- The chaining operation makes the ciphertext blocks dependent on the current and all preceding plaintext blocks and therefore blocks can not be rearranged.
- The use of different starting variables prevents the same plaintext enciphering to the same ciphertext.
- An error will affect the current and the following ciphertext blocks.

Cipher Feedback Mode (CFB)

Normally, this is found as the function *algorithm_cfb_encrypt()*.

- a number of bits (j) <= 64 are enciphered at a time.
- The CFB mode produces the same ciphertext whenever the same plaintext is encrypted using the same key and starting variable.
- The chaining operation makes the ciphertext variables dependent on the current and all preceding variables and therefore j-bit variables are chained together and can not be rearranged.
- The use of different starting variables prevents the same plaintext enciphering to the same ciphertext.
- The strength of the CFB mode depends on the size of k (maximal if j == k). In my implementation this is always the case.
- Selection of a small value for j will require more cycles through the encipherment algorithm per unit of plaintext and thus cause greater processing overheads.
- Only multiples of j bits can be enciphered.
- An error will affect the current and the following ciphertext variables.

Output Feedback Mode (OFB)

Normally, this is found as the function *algorithm_ofb_encrypt()*.

- a number of bits (j) <= 64 are enciphered at a time.

- The OFB mode produces the same ciphertext whenever the same plaintext enciphered using the same key and starting variable. More over, in the OFB mode the same key stream is produced when the same key and start variable are used. Consequently, for security reasons a specific start variable should be used only once for a given key.
- The absence of chaining makes the OFB more vulnerable to specific attacks.
- The use of different start variables values prevents the same plaintext enciphering to the same ciphertext, by producing different key streams.
- Selection of a small value for j will require more cycles through the encipherment algorithm per unit of plaintext and thus cause greater processing overheads.
- Only multiples of j bits can be enciphered.
- OFB mode of operation does not extend ciphertext errors in the resultant plaintext output. Every bit error in the ciphertext causes only one bit to be in error in the deciphered plaintext.
- OFB mode is not self-synchronizing. If the two operation of encipherment and decipherment get out of synchronism, the system needs to be re-initialized.
- Each re-initialization should use a value of the start variable different from the start variable values used before with the same key. The reason for this is that an identical bit stream would be produced each time from the same parameters. This would be susceptible to a 'known plaintext' attack.

Triple ECB Mode

Normally, this is found as the function *algorithm_ecb3_encrypt()*.

- Encrypt with key1, decrypt with key2 and encrypt with key3 again.
- As for ECB encryption but increases the key length to 168 bits. There are theoretic attacks that can be used that make the effective key length 112 bits, but this attack also requires 2^56 blocks of memory, not very likely, even for the NSA.
- If both keys are the same it is equivalent to encrypting once with just one key.
- If the first and last key are the same, the key length is 112 bits. There are attacks that could reduce the effective key strength to only slightly more than 56 bits, but these require a lot of memory.
- If all 3 keys are the same, this is effectively the same as normal ecb mode.

Triple CBC Mode

Normally, this is found as the function *algorithm_ede3_cbc_encrypt()*.

- Encrypt with key1, decrypt with key2 and then encrypt with key3.
- As for CBC encryption but increases the key length to 168 bits with the same restrictions as for triple ecb mode.

NOTES

This text was been written in large parts by Eric Young in his original documentation for SSLeay, the predecessor of OpenSSL. In turn, he attributed it to:

```
AS 2805.5.2
Australian Standard
Electronic funds transfer - Requirements for interfaces,
Part 5.2: Modes of operation for an n-bit block cipher algorithm
Appendix A
```

SEE ALSO

blowfish(3), des(3), idea(3), rc2(3)

NAME

environ — user process environment

SYNOPSIS

```
extern char **environ;
```

DESCRIPTION

An array of strings called the *environment* is made available by execve(2) when a process begins. By convention these strings have the form "*name=value*". The following names are used by various commands:

AUDIOCTLDEVICE — The name of the audio control device to be used by audioctl(1), audioplay(1) and audiorecord(1).

AUDIODEVICE — The name of the audio device to be used by audioplay(1) and audiorecord(1).

BLOCKSIZE — The size of the block units used by several commands, most notably df(1), du(1) and ls(1). BLOCKSIZE may be specified in units of a byte by specifying a number, in units of a kilobyte by specifying a number followed by "K" or "k", in units of a megabyte by specifying a number followed by "M" or "m" and in units of a gigabyte by specifying a number followed by "G" or "g". Sizes less than 512 bytes or greater than a gigabyte are ignored.

EXINIT — A startup list of commands read by ex(1) and vi(1).

HOME — A user's login directory, set by login(1) from the password file passwd(5).

LANG — Default for all NLS categories. Only used if LC_ALL or the environment variable for a particular NLS category is not provided (LC_COLLATE, LC_CTYPE, LC_MESSAGES, LC_MONETARY, LC_NUMERIC, or LC_TIME).

LC_ALL — Override for all NLS categories. If set, overrides the values of LC_COLLATE, LC_CTYPE, LC_MESSAGES, LC_MONETARY, LC_NUMERIC, and LC_TIME.

LC_COLLATE — NLS string-collation order information.

LC_CTYPE — NLS character classification, case conversion, and other character attributes.

LC_MESSAGES — NLS format for affirmative and negative responses.

LC_MONETARY — NLS rules and symbols for formatting monetary numeric information.

LC_NUMERIC — NLS rules and symbols for formatting nonmonetary numeric information.

LC_TIME — NLS rules and symbols for formatting time and date information.

LIBC_DIAGASSERT — Control how the **_DIAGASSERT**() macro (from ⟨assert.h⟩) behaves once the assertion is raised. Refer to _DIAGASSERT(3) for more information.

LOGNAME — The login name of the user.

MALLOC_OPTIONS — Control the behaviour of the **malloc**() function. Refer to malloc(3) for more information.

MIXERDEVICE — The name of the audio mixer device to be used by mixerctl(1).

PAGER — The program used for paginating the output of several commands such as man(1). If null or not set, the standard pagination program more(1) will be used.

PATH — The sequence of directories, separated by colons, searched by csh(1), sh(1), system(3), execvp(3), etc, when looking for an executable file. PATH is set to "/usr/bin:/bin:/usr/pkg/bin:/usr/local/bin" initially by login(1).

PRINTER — The name of the default printer to be used by lpr(1), lpq(1), and lprm(1).

RCMD_CMD — When using the rcmd(3) function, this variable is used as the program to run instead of rcmd(1).

SHELL — The full pathname of the user's login shell.

TERM — The kind of terminal for which output is to be prepared. This information is used by commands, such as nroff(1) which may exploit special terminal capabilities. See /usr/share/misc/termcap (termcap(5)) for a list of terminal types.

TERMCAP — The string describing the terminal in TERM, or, if it begins with a '/', the name of the termcap file. See TERMPATH below, termcap(5), and termcap(3).

TERMPATH — A sequence of pathnames of termcap files, separated by colons or spaces, which are searched for terminal descriptions in the order listed. Having no TERMPATH is equivalent to a TERMPATH of "$HOME/.termcap:/usr/share/misc/termcap". TERMPATH is ignored if TERMCAP contains a full pathname.

TIMEFORMAT — A strftime(3) format string that may be used by programs such as dump(8) for formatting timestamps.

TMPDIR — The directory in which to store temporary files. Most applications use either /tmp or /var/tmp. Setting this variable will make them use another directory.

TZ — The timezone to use when displaying dates. The normal format is a pathname relative to /usr/share/zoneinfo. For example, the command **env TZ=US/Pacific date** displays the current time in California. See tzset(3) for more information.

USER — The login name of the user. It is recommended that portable applications use LOGNAME instead.

Further names may be placed in the environment by the **export** command and *name=value* arguments in sh(1), or by the **setenv** command if you use csh(1). It is unwise to change certain sh(1) variables that are frequently exported by .profile files, such as MAIL, PS1, PS2, and IFS, unless you know what you are doing.

SEE ALSO

audioctl(1), audioplay(1), audiorecord(1), csh(1), ex(1), login(1), man(1), more(1), sh(1), execve(2), _DIAGASSERT(3), execle(3), malloc(3), rcmd(3), system(3), termcap(3), audio(4), termcap(5), nls(7), dump(8)

HISTORY

The **environ** manual page appeared in 4.2BSD.

NAME

hier — layout of filesystems

DESCRIPTION

An outline of the filesystem hierarchy.

Naming is very important. The UNIX System relies on filename conventions for much of its power as a system. The following file system layout describes generally where things are and what they are, with references to other man pages for more detailed documentation.

Not all files will be in every system.

/ root directory of the system

/COPYRIGHT
system copyright notice, most often put on CD-ROM distributions.

/[a-z]/ user filesystems

/altroot/ alternate root filesystem, in case of disaster

/bin/ utilities used in both single and multi-user environments

/boot* second-stage boot loader(s) for some platforms; see `installboot`(8)

/dev/ block, character and other special device files

MAKEDEV
script for creating device files; see `makedev`(8)

console the computer's console device
drum system swap space; see `drum`(4)
fd/ file descriptor files; see `fd`(4)
klog kernel logging device; see `syslog`(3)
kmem kernel virtual memory device; see `mem`(4)
log UNIX domain datagram log socket; see `syslogd`(8)
mem kernel physical memory device; see `mem`(4)
null the null device; see `null`(4)
stderr
stdin
stdout file descriptor files; see `fd`(4)
tty process' controlling terminal device; see `tty`(4)
zero the zero device; see `zero`(4)

/etc/ system configuration files and scripts

amd* configuration files for `amd`(8)
changelist files backed up by the security script
crontab schedule used by the `cron`(8) daemon
csh.cshrc
csh.login
csh.logout system-wide scripts for `csh`(1)
daily script run each day by `cron`(8)
daily.conf configuration file for **daily**; see `daily.conf`(5)
defaults/ default configuration files read by various `/etc/*.conf` files
disktab disk description file, see `disktab`(5)

dm.conf dungeon master configuration; see `dm.conf`(5)
dumpdates dump history; see `dump`(8)
exports filesystem export information; see `mountd`(8)
fstab filesystem information; see `fstab`(5) and `mount`(8)
ftpusers users denied `ftp`(1) access; see `ftpd`(8)
ftpwelcome `ftp`(1) initial message; see `ftpd`(8)
gettytab terminal configuration database; see `gettytab`(5)
group group permissions file; see `group`(5)
hosts host name database backup for `named`(8); see `hosts`(5)
hosts.equiv trusted machines with equivalent user ID's
hosts.lpd trusted machines with printing privileges
inetd.conf Internet server configuration file; see `inetd`(8)
kerberosV/ configuration files for the kerberos version V; see `kerberos`(8)
localtime local timezone information; see `ctime`(3)
mail/ configuration files for `sendmail`(8)

- **aliases**∗ name alias files
- **sendmail.**∗ `sendmail`(8) configuration information

mail.rc system-wide initialization script for `mail`(1)
man.conf configuration file for `man`(1); see `man.conf`(5)
master.passwd Main password file, readable only by root; see `passwd`(5)
mk.conf optional file containing `make`(1) variables, read by pkgsrc and the system sources.
monthly script run each month by `cron`(8)
monthly.conf configuration file for **monthly**; see `monthly.conf`(5)
motd system message of the day
mtree/ mtree configuration files; see `mtree`(8)
named.∗
namedb/ named configuration files and databases; see `named`(8)
netgroup network groups; see `netgroup`(5)
netstart network startup script
networks network name data base; see `networks`(5)
passwd World readable password file generated from master.passwd; see `passwd`(5), `pwd_mkdb`(8)
phones remote host phone number data base; see `phones`(5)
printcap system printer configuration; see `printcap`(5)
protocols protocol name database; see `protocols`(5)
pwd.db database form of passwd file; see `pwd_mkdb`(8)
rc master system startup script invoked by `init`(8); see `rc`(8)
rc.conf configuration file for system startup and shutdown scripts; see `rc.conf`(5)
rc.d/ directory containing per-subsystem startup and shutdown scripts; see `rc`(8)
rc.local locally editable system startup script
rc.shutdown master system shutdown script invoked by `shutdown`(8); see `rc`(8)
remote remote host description file; see `remote`(5)
security daily (in)security script run by `cron`(8)
security.conf configuration file for **security**; see `security.conf`(5)
services service name data base; see `services`(5)
shells list of permitted shells; see `shells`(5)

sliphome/ SLIP login/logout scripts; see `sliplogin`(8)
spwd.db database form of master.passwd file; see `pwd_mkdb`(8)
syslog.conf `syslogd`(8) configuration file; see `syslog.conf`(5)
termcap terminal type database; see `termcap`(3)
ttys terminal initialization information; see `ttys`(5)
uucp/ UUCP configuration files; see `uucp`(1) and `uucico`(8).
weekly script run each week by `cron`(8)
weekly.conf configuration file for **weekly**; see `weekly.conf`(5)

/home/ mount point for the automounter; see `amd`(8)

/lib/ dynamic linked libraries used by dynamic linked programs (such as those in `/bin/` and `/sbin/`) that cannot rely upon `/usr/lib/` being available.

/libexec/ system utilities (such as the dynamic linker) required by programs and libraries that cannot rely upon `/usr/libexec/` being available.

/mnt/ empty directory commonly used by system administrators as a temporary mount point

/netbsd pure kernel executable (the operating system loaded into memory at boot time).

/rescue/ statically linked rescue tools, for use in system recovery

/root/ home directory for the super-user

.cshrc super-user start-up file
.login super-user start-up file
.profile super-user start-up file
.rhosts super-user id mapping between machines

/sbin/ system programs and administration utilities used in both single-user and multi-user environments

/stand/ programs used in a standalone environment

/tmp/ temporary files, usually a `mfs`(8) memory-based filesystem (the contents of `/tmp` are usually *not* preserved across a system reboot)

/usr/ contains the majority of the system utilities and files

X11R6/ X11 files

bin/ X11 binaries
include/ X11 include files
lib/ X11 libraries

bin/ common utilities, programming tools, and applications
games/ the important stuff
include/ standard C include files

arpa/ include files for Internet service protocols
atf/ include files for the Automated Testing Framework; see `atf`(1)
g++/ include files for the C++ compiler
machine/ machine specific include files
net/
netatalk/ C include files for AppleTalk protocols miscellaneous network include files; see `atalk`(4)

netinet/ include files for Internet standard protocols; see `inet`(4)
netinet6/ include files for Internet protocol version 6; see `inet6`(4)
netiso/ include files for ISO standard protocols; see `iso`(4)
netkey/ include files for secret key management, used for security protocols; see `ipsec`(4)
netnatm/ C include files for native mode ATM
nfs/ C include files for NFS (Network File System)
protocols/ C include files for Berkeley service protocols
sys/ system C include files (kernel data structures)
ufs/ C include files for UFS (The U-word File System)

lib/ archive, profiled, position independent archive, and shared libraries
libdata/ miscellaneous utility data files
libexec/ system daemons & system utilities (executed by other programs)

uucp/ UUCP binaries and scripts (historically placed; to be moved)

lkm/ loadable kernel modules
mdec/ boot blocks, etc.
obj/ architecture-specific target tree produced by building the `/usr/src` tree; normally a symbolic link or mounted filesystem
pkg/ packages maintained by groups other than the NetBSD Project.

bin/ contributed binaries
include/ contributed include files
lib/ contributed libraries
libdata/ contributed data files
libexec/ contributed daemons
sbin/ contributed system utilities

pkgsrc/ build descriptions ("packages") for the NetBSD packages system.

distfiles/ Where unchanged source archives are fetched to/stored
packages/ Where compiled binary packages are stored

There are also several other subdirectories which contain packages of a certain category, e.g., archivers, graphics, ...

sbin/ system daemons and system utilities (normally executed by the super-user)
share/ architecture-independent text files

calendar/ a variety of calendar files; see `calendar`(1)
dict/ word lists; see `look`(1) and `spell`(1)

words common words
web2 words of Webster's 2nd International
papers/ reference databases; see `refer`(1)
special/ custom word lists; see `spell`(1)

doc/ miscellaneous documentation; source for most of the printed 4.3BSD manuals (available from the USENIX association)
games/ text files used by various games
i18n/ internationalization databases; see `iconv`(3)
lkm/ documentation on the loadable kernel modules interface
locale/ locale databases and gettext message catalogs; see `setlocale`(3) and `gettext`(3)

man/ formatted manual pages
me/ macros for use with the `me`(7) macro package
misc/ miscellaneous system-wide text files

termcap terminal characteristics database; see `termcap`(5)

mk/ include files for `make`(1)
ms/ macros for use with the `ms`(7) macro package
nls/ message catalogs; see `catgets`(3)
skel/ sample initialization files for new user accounts
tabset/ tab description files for a variety of terminals, used in the termcap file; see `termcap`(5)
tmac/ text processing macros; see `nroff`(1) and `troff`(1)
zoneinfo/ timezone configuration information; see `tzfile`(5)

tests/ test programs; see `atf-run`(1) for information on how to run them

/usr/src/ NetBSD and local source files

bin/ source for utilities/files in `/bin`
common/ sources shared between kernel and userland
crypto/ cryptographic source, which may have import or export restrictions
dist/ third-party 'virgin' source code, referenced by other parts of the source tree
distrib/ tools and data-files for making distributions
doc/ documentation about the source tree (i.e., about the tree, not about how to use the built software.)
etc/ source (usually example files) for files in `/etc`
external/ source for programs from external third parties (where NetBSD is the not the primary maintainer), grouped by license, and then products per license

bsd/ BSD (or equivalent) licensed software, possibly with the "advertising clause"

games/ source for utilities/files in `/usr/games`
gnu/ source for programs covered by the GNU license (or similar)
include/ source for files in `/usr/include`
lib/ source for libraries in `/usr/lib`
libexec/ source for utilities/files in `/usr/libexec`
regress/ various regression tests
rescue/ source/makefiles for /rescue
sbin/ source for utilities/files in `/sbin`
share/ source for files in `/usr/share`

doc/

papers/ source for various Berkeley technical papers
psd/ source for Programmer's Supplementary Documents
smm/ source for System Manager's Manual
usd/ source for User's Supplementary Documents

sys/ kernel source files

arch/ architecture-specific support

acorn26/ Acorn Archimedes, A-series and R-series systems

acorn32/	Acorn RiscPC/A7000 and VLSI RC7500
algor/	Algorithmics Ltd. MIPS evaluations boards
alpha/	Digital/Compaq Alpha
amd64/	Computers with x86_64 capable CPUs
amiga/	Commodore Amiga and MacroSystem DraCo
amigappc/	PowerPC based Amiga boards
arc/	MIPS-based machines following the Advanced RISC Computing spec
arm/	ARM processor general support
atari/	Atari TT030, Falcon and Hades
bebox/	Be Inc. BeBox
cats/	Chalice Technology's CATS and Intel's EBSA-285 evaluation boards
cesfic/	CES FIC8234 VME processor board
cobalt/	Cobalt Networks' MIPS-based Microserver
dreamcast/	Sega Dreamcast game console
evbarm/	ARM based evaluation boards
evbmips/	MIPS based evaluation boards
evbppc/	PowerPC based evaluation boards and appliances
evbsh3/	SH3/SH4 based evaluation boards
ews4800mips/	NEC's MIPS based EWS4800 workstations
hp300/	Hewlett-Packard 9000/300 and 400 680x0-based workstations
hp700/	Hewlett-Packard 9000/700 HPPA based workstations
hpcarm/	StrongARM based WinCE PDA machines
hpcmips/	MIPS based WinCE PDA machines
hpcsh/	Hitachi SH3/4 based WinCE PDA machines
hppa/	HPPA processor general support
i386/	80x86-based IBM PCs and clones
ibmnws/	IBM Network Station 1000
iyonix/	Castle Technology's Iyonix ARM based PCs
luna68k/	Omron Tateishi Electric's 680x0-based LUNA workstations
m68k/	680x0 processor general support
mac68k/	Apple Macintosh with 68k CPU
macppc/	Apple Power Macintosh and clones
mips/	MIPS processor general support
mipsco/	MIPS Computer Systems Inc. family of workstations and servers
mmeye/	Brains Inc. SH3 based mmEye multimedia server

mvme68k/	Motorola MVME 680x0-based SBCs
mvmeppc/	Motorola PowerPC VME SBCs
netwinder/	StrongARM based NetWinder machines
news68k/	Sony’s 680x0-based NEWS workstations
newsmips/	Sony’s MIPS-based NEWS workstations
next68k/	NeXT 68k "black" hardware
ofppc/	Open Firmware PowerPC workstations
playstation2/	SONY PlayStation 2
pmax/	Digital MIPS-based DECstations and DECsystems
powerpc/	PowerPC processor support
prep/	PReP (PowerPC Reference Platform) and CHRP machines
sandpoint/	Motorola Sandpoint reference platform
sbmips/	Broadcom/SiByte evaluation boards
sgimips/	Silicon Graphics’ MIPS-based workstations
sh3/	SH3/SH4 processor general support
shark/	Digital DNARD ("Shark")
sparc/	Sun Microsystems SPARC (32-bit) and UltraSPARC (in 32-bit mode)
sparc64/	Sun Microsystems UltraSPARC (in native 64-bit mode)
sun2/	Sun Microsystems 68010-based Sun 2 architecture
sun3/	Sun Microsystems 68020/68030-based Sun 3/3x architecture
sun68k/	680x0-based Sun architecture general support
vax/	Digital VAX
x68k/	Sharp X680x0 680x0-based workstations
x86/	General support for PC/AT compatibles with ia32 or x86_64 CPUs
xen/	The Xen virtual machine monitor
zaurus/	Sharp C3x00 Arm based PDA

compat/ kernel compatibility modules directory

common/	common compatibility routines, old 4BSD and NetBSD routines.
freebsd/	support for FreeBSD binaries; see `compat_freebsd`(8)
hpux/	support for 68000 HP-UX binaries
ibcs2/	support for Intel Binary binaries
linux/	support for Linux binaries; see `compat_linux`(8)
m68k4k/	support for 4KB page 68000 binaries
netbsd32/	support for NetBSD 32-bit binaries on 64 bit platforms with compatible CPU families
osf1/	support for Digital UNIX (formerly OSF/1) binaries

ossaudio/ support for OSS audio
pecoff/ support for Win32 binaries; see `compat_pecoff`(8)
sunos/ support for SunOS 4.x binaries; see `compat_sunos`(8)
svr4/ support for System V Release 4 binaries; see `compat_svr4`(8)
ultrix/ support for ULTRIX binaries
vax1k/ support for older VAX binaries that started on a 1 KB boundary

conf/ architecture independent configuration directory
crypto/ cryptographic kernel source, which may have import or export restrictions
ddb/ in kernel debugger
dev/ architecture independent device support
fs/ miscellaneous file systems
adosfs/ AmigaDOS file-system support; see `mount_ados`(8)
cd9660/ support for the ISO-9660 filesystem; see `mount_cd9660`(8)
filecorefs/ support for the Acorn RISC OS filecore filesystem; see `mount_filecore`(8)
msdosfs/ MS-DOS file system; see `mount_msdos`(8)
ntfs/ NTFS filesystem support; see `mount_ntfs`(8)
ptyfs/ pseudo-terminal device filesystem; see `mount_ptyfs`(8)
smbfs/ SMB/CIFS filesystem support; see `mount_smbfs`(8)
union/ union file system; see `mount_union`(8)
gdbscripts/ support for accessing kernel structures from within the debugger `gdb`(1).
ipkdb/ support for kernel debugging over the network
kern/ support for the high kernel (system calls)
lib/ kernel libraries
libkern/ C library routines used in the kernel
libsa/ machine independent stand alone kernel library
libz/ compression library
lkm/ loadable kernel modules
compat/ LKM support compatibility modules; currently unsupported.
netinet/ LKM support networking modules
if_ipl/ LKM for IP-Filter
vfs/ LKM support for file systems.
miscfs/ miscellaneous file systems

deadfs/ kernel only dead file system
fdesc/ file descriptor file system; see `mount_fdesc`(8)
fifofs/ POSIX FIFO support
genfs/ kernel only generic file system
kernfs/ kernel namespace file system; see `mount_kernfs`(8)
nullfs/ loop back file system; see `mount_null`(8)
overlay/ overlay file system; see `mount_overlay`(8)
portal/ portal file system; see `mount_portal`(8)
procfs/ process file system; see `mount_procfs`(8)
specfs/ kernel only special file system
syncfs/ kernel trickle sync algorithm
umapfs/ user and group re-mapping file system; see `mount_umap`(8)

net/ miscellaneous networking support
netatalk/ AppleTalk networking support
netinet/ IP networking support
netinet6/ IPv6 networking support
netiso/ ISO networking support
netkey/ Key database for IPsec networking support
netnatm/ ATM networking support
nfs/ NFS support
stand/ kernel standalone support
sys/ kernel (and system) include files
ufs/ local filesystem support

ffs/ the Berkeley Fast File System
lfs/ the log-structured file system
mfs/ the in-memory file system
ufs/ shared UNIX file system support

uvm/ UVM virtual memory system

tests/ source for test programs in `/usr/tests`
usr.bin/ source for utilities/files in `/usr/bin`
usr.sbin/ source for utilities/files in `/usr/sbin`

/var/ multi-purpose log, temporary, transient, and spool files

account/ system accounting files

acct execution accounting file; see `acct`(5)

at/ timed command scheduling files; see `at`(1)
backups/ miscellaneous backup files, largely of files found in `/etc`
chroot/ home directories of applications which are run in a `chroot`(8) “cage”.
crash/ system crash dumps; see `savecore`(8)
cron/ scheduled commands configuration files; see `cron`(8)
db/ miscellaneous automatically generated system-specific database files, and persistent files used in the maintenance of third party software.

pkg default location for metadata related to third party software packages. See `pkg_install`(1) for more details of the NetBSD Packages Collection, or pkgsrc.

games/ miscellaneous game status and log files
heimdal/ Kerberos 5 KDC database; see `kdc`(8)
log/ miscellaneous system log files

amd.∗	`amd`(8) logs
daily.out	output of the last run of the `/etc/daily` script
ftp.∗	`ftp`(1) logs
kerberos.∗	`kerberos`(8) logs
lastlog	system last time logged in log; see `utmp`(5)
lpd-errs.∗	printer daemon error logs; see `lpd`(8)
maillog.∗	`sendmail`(8) log files
messages.∗	general system information log
monthly.out	output of the last run of the `/etc/monthly` script
secure	sensitive security information log
sendmail.st	`sendmail`(8) statistics
timed.∗	`timed`(8) logs
weekly.out	output of the last run of the `/etc/weekly` script
wtmp	login/logout log; see `utmp`(5)

mail/ user system mailboxes
msgs/ system messages; see `msgs`(1)
preserve/ temporary home of files preserved after an accidental death of `ex`(1) or `vi`(1)
quotas/ filesystem quota information
run/ system information files, rebuilt after each reboot

utmp	database of current users; see `utmp`(5)

rwho/ rwho data files; see `rwhod`(8), `rwho`(1), and `ruptime`(1)
spool/ miscellaneous printer and mail system spooling directories

ftp/	commonly "~ftp", the anonymous ftp root directory; see `ftpd`(8)
mqueue/	sendmail mail queue; see `sendmail`(8)
news/	Network news archival and spooling directories
output/	printer spooling directories
postfix/	postfix mail queue; see `postfix`(1)
uucp/	uucp spool directory
uucppublic/	commonly "~uucp", the uucp public temporary directory; see `uucp`(1)

tmp/ temporary files that are not discarded between system reboots

vi.recover/	recovery directory for new `vi`(1)

yp/ Databases and configuration for the NIS (YP) system; see `nis`(8).

SEE ALSO

`apropos`(1), `ls`(1), `whatis`(1), `whereis`(1), `which`(1)

HISTORY

A **`hier`** manual page appeared in Version 7 AT&T UNIX.

NAME

hostname — host name resolution description

DESCRIPTION

Hostnames are domains, where a domain is a hierarchical, dot-separated list of subdomains; for example, the machine monet, in the Berkeley subdomain of the EDU subdomain of the Internet would be represented as

```
monet.Berkeley.EDU
```

(with no trailing dot).

Hostnames are often used with network client and server programs, which must generally translate the name to an address for use. (This function is generally performed by the library routine `gethostbyname`(3).) Hostnames are resolved by the Internet name resolver in the following fashion.

If the name consists of a single component, i.e. contains no dot, and if the environment variable "`HOSTALIASES`" is set to the name of a file, that file is searched for any string matching the input hostname. The file should consist of lines made up of two white-space separated strings, the first of which is the hostname alias, and the second of which is the complete hostname to be substituted for that alias. If a case-insensitive match is found between the hostname to be resolved and the first field of a line in the file, the substituted name is looked up with no further processing.

If the input name ends with a trailing dot, the trailing dot is removed, and the remaining name is looked up with no further processing.

If the input name does not end with a trailing dot, it is looked up by searching through a list of domains until a match is found. The default search list includes first the local domain, then its parent domains with at least 2 name components (longest first). For example, in the domain CS.Berkeley.EDU, the name lithium.CChem will be checked first as lithium.CChem.CS.Berkeley.EDU and then as lithium.CChem.Berkeley.EDU. Lithium.CChem.EDU will not be tried, as there is only one component remaining from the local domain. The search path can be changed from the default by a system-wide configuration file (see `resolv.conf`(5)).

SEE ALSO

`gethostbyname`(3), `resolv.conf`(5), `mailaddr`(7), `named`(8)

HISTORY

hostname appeared in 4.2BSD.

NAME

mailaddr — mail addressing description

DESCRIPTION

Mail addresses are based on the Internet protocol listed at the end of this manual page. These addresses are in the general format

```
user@domain
```

where a domain is a hierarchical dot separated list of subdomains. For example, a valid address is:

```
eric@CS.Berkeley.EDU
```

Unlike some other (now obsolete) forms of addressing, domains do not imply any routing, or the existence of a particular host. Simply because mail may be sent to ‘‘user@somedomain.com’’ does not imply that there is any actual host named ‘‘somedomain.com’’, and does not imply a particular routing of the message. Routing is performed by Mail Transport Agents, such as sendmail(8), based on policies set in the MTA's configuration.

Abbreviation

Under certain circumstances it may not be necessary to type the entire domain name. In general, anything following the first dot may be omitted if it is the same as the domain from which you are sending the message. For example, a user on ‘‘calder.berkeley.edu’’ could send to ‘‘eric@CS’’ without adding the ‘‘berkeley.edu’’ since it is the same on both sending and receiving hosts. Whether abbreviation is permitted depends on how your site is configured.

Case Distinctions

Domain names (i.e., anything after the ‘‘@’’ sign) may be given in any mixture of upper and lower case. Most hosts accept any combination of case in user names, although there are exceptions.

Postmaster

Every site is required to have a user or user alias designated ‘‘postmaster’’ to which problems with the mail system may be addressed, for example:

```
postmaster@CS.Berkeley.EDU
```

Obsolete Formats

Certain old address formats, such as UUCP ‘‘bang path’’ addresses, explicitly routed internet addresses (so-called ‘‘route-addrs’’ and the ‘‘percent hack’’) and others have been used historically. All these addressing formats are now considered obsolete, and should no longer be used.

To some extent, sendmail(8) (when running with normal configuration files) attempts to provide backward compatibility for these addressing forms, but in practice many of them no longer work. Users should always use standard Internet style addresses.

SEE ALSO

mail(1), sendmail(8)

D. H. Crocker, *Standard for the Format of Arpa Internet Text Messages*, RFC, 822, August 1982.

HISTORY

mailaddr appeared in 4.2BSD.

BUGS

The RFC 822 group syntax (‘‘group:user1,user2,user3;’’) is not supported except in the special case of ‘‘group:;’’ because of a conflict with old berknet-style addresses, not that anyone cares about either berknet or group syntax style addresses any longer.

NAME

mdoc — quick reference guide for the **-mdoc** macro package

SYNOPSIS

groff -m*doc files . . .*

DESCRIPTION

The **-mdoc** package is a set of content-based and domain-based macros used to format the BSD man pages. The macro names and their meanings are listed below for quick reference; for a detailed explanation on using the package, see the tutorial sampler `mdoc.samples`(7).

The macros are described in two groups, the first includes the structural and physical page layout macros. The second contains the manual and general text domain macros which differentiate the **-oc** package from other `troff`(1) formatting packages.

PAGE STRUCTURE DOMAIN

Title Macros

To create a valid manual page, these three macros, in this order, are required:

`.Dd`	*Month day, year*	Document date.
`.Dt`	*DOCUMENT_TITLE [section] [volume]*	Title, in upper case.
`.Os`	*OPERATING_SYSTEM [version/release]*	Operating system (BSD).

Page Layout Macros

Section headers, paragraph breaks, lists and displays.

`.Sh` Section Headers. Valid headers, in the order of presentation:

NAME	Name section, should include the `.Nm` or `.Fn` and the `.Nd` macros.
LIBRARY	Sections two and three function calls.
SYNOPSIS	Usage.
DESCRIPTION	General description, should include options and parameters.
EXIT STATUS	Sections one and eight commands and utilities.
RETURN VALUES	Sections two and three function calls.
ENVIRONMENT	Describe environment variables.
FILES	Files associated with the subject.
EXAMPLES	Examples and suggestions.
DIAGNOSTICS	Normally used for section four device interface diagnostics.
ERRORS	Sections two and three error and signal handling.
SEE ALSO	Cross references and citations.
STANDARDS	Conformance to standards if applicable.
HISTORY	If a standard is not applicable, the history of the subject should be given.
BUGS	Gotchas and caveats.
SECURITY CONSIDERATIONS	Security issues to be aware of.
other	Customized headers may be added at the author's discretion.

`.Ss` Subsection Headers.

`.Pp` Paragraph Break. Vertical space (one line).

`.D1` (D-one) Display-one Indent and display one text line.

`.Dl` (D-ell) Display-one literal. Indent and display one line of literal text.

`.Bd` Begin-display block. Display options:

-ragged	Unjustified (ragged edges).
-filled	Filled, and if `troff`(1), also justified.

-unfilled Unfilled, unjustified.
-literal Literal text or code.
-file *name* Read in named *file* and display.
-offset *string* Offset display. Acceptable *string* values:
left Align block on left (default).
center Approximate center margin.
indent Six constant width spaces (a tab).
indent-two Two tabs.
right Left aligns block 2 inches from right.
*xx***n** Where *xx* is a number from 4**n** to 99**n**.
Aa Where *Aa* is a callable macro name.
string The width of *string* is used.

.Ed End-display (matches .Bd).
.Bl Begin-list. Create lists or columns. Options:
List-types
-bullet Bullet Item List
-dash Dash Item List
-hyphen (as per **-dash**)
-item Unlabeled List
-enum Enumerated List
-tag Tag Labeled List
-diag Diagnostic List
-hang Hanging Labeled List
-ohang Overhanging Labeled List
-inset Inset or Run-on Labeled List
-column Multiple Columns
List-parameters
-offset (All lists.) See .Bd begin-display above.
-width (**-tag** and **-hang** lists only.) See .Bd. This parameter is effectively required for **-tag** lists.
-compact (All lists.) Suppresses blank lines.
.El End-list.
.It List item.

MANUAL AND GENERAL TEXT DOMAIN MACROS

The manual and general text domain macros are special in that most of them are parsed for callable macros for example:

```
.Op Fl s Ar file
```

Produces:

[**-s** *file*]

In this example, the option enclosure macro .Op is parsed, and calls the callable content macro 'Fl' which operates on the argument 's' and then calls the callable content macro 'Ar' which operates on the argument file. Some macros may be callable, but are not parsed and vice versa. These macros are indicated in the *parsed* and *callable* columns below.

Unless stated, manual domain macros share a common syntax:

```
.Va argument [ . , ; : ( ) [ ] argument ... ]
```

Note: Opening and closing punctuation characters are only recognized as such if they are presented one at a time. The string '),' is not recognized as punctuation and will be output with a leading white space and in

whatever font the calling macro uses. The argument list `] ) ,` is recognized as three sequential closing punctuation characters and a leading white space is not output between the characters and the previous argument (if any).

The special meaning of a punctuation character may be escaped with the string '`\&`'. For example the following string:

```
.Ar file1 \&, file2 , file3 ) .
```

Produces:

file1 , file2, *file3*).

Manual Domain Macros

Name	*Parsed*	*Callable*	*Description*
Ad	Yes	Yes	Address. (This macro may be deprecated.)
Ar	Yes	Yes	Command line argument.
Cd	No	No	Configuration declaration (section four only).
Cm	Yes	Yes	Command line argument modifier.
Dv	Yes	Yes	Defined variable (source code).
Er	Yes	Yes	Error number (source code).
Ev	Yes	Yes	Environment variable.
Fa	Yes	Yes	Function argument.
Fd	No	No	Function declaration.
Fl	Yes	Yes	Command line flag.
Fn	Yes	Yes	Function call (also .Fo and .Fc).
Ft	Yes	Yes	Function type.
Ic	Yes	Yes	Interactive command.
In	No	No	Include header.
Lb	Yes	Yes	Library name.
Li	Yes	Yes	Literal text.
Nd	No	No	Name description.
Nm	Yes	Yes	Command name.
Op	Yes	Yes	Option (also .Oo and .Oc).
Ot	Yes	Yes	Old style function type (Fortran only).
Pa	Yes	Yes	Pathname or file name.
St	Yes	Yes	Standards (-p1003.2, -p1003.1 or -ansiC)
Va	Yes	Yes	Variable name.
Vt	Yes	Yes	Variable type.
Xr	Yes	Yes	Manual Page Cross Reference.

General Text Domain Macros

Name	*Parsed*	*Callable*	*Description*
%A	Yes	No	Reference author.
%B	Yes	Yes	Reference book title.
%C	No	No	Reference place of publishing (city).
%D	No	No	Reference date.
%J	Yes	Yes	Reference journal title.
%N	No	No	Reference issue number.
%O	No	No	Reference optional information.
%P	No	No	Reference page number(s).

%R	No	No	Reference report Name.
%T	Yes	Yes	Reference article title.
%V	No	No	Reference volume.
Ac	Yes	Yes	Angle close quote.
Ao	Yes	Yes	Angle open quote.
Ap	Yes	Yes	Insert apostrophe; switch to .No mode
Aq	Yes	Yes	Angle quote.
At	No	No	AT&T UNIX.
Bc	Yes	Yes	Bracket close quote.
Bf	No	No	Begin font mode.
Bo	Yes	Yes	Bracket open quote.
Bq	Yes	Yes	Bracket quote.
Bsx	Yes	Yes	BSD/OS.
Bx	Yes	Yes	BSD.
Db	No	No	Debug (default is "off").
Dc	Yes	Yes	Double close quote.
Do	Yes	Yes	Double open quote.
Dq	Yes	Yes	Double quote.
Ec	Yes	Yes	Enclose string close quote.
Ef	No	No	End font mode.
Em	Yes	Yes	Emphasis (traditional English).
Eo	Yes	Yes	Enclose string open quote.
Fx	Yes	Yes	FreeBSD.
No	Yes	Yes	Normal text (no-op).
Ns	Yes	Yes	No space.
Nx	Yes	Yes	NetBSD.
Ox	Yes	Yes	OpenBSD.
Pc	Yes	Yes	Parenthesis close quote.
Pf	Yes	No	Prefix string.
Po	Yes	Yes	Parenthesis open quote.
Pq	Yes	Yes	Parentheses quote.
Qc	Yes	Yes	Straight Double close quote.
Ql	Yes	Yes	Quoted literal.
Qo	Yes	Yes	Straight Double open quote.
Qq	Yes	Yes	Straight Double quote.
Re	No	No	Reference end.
Rs	No	No	Reference start.
Sc	Yes	Yes	Single close quote.
So	Yes	Yes	Single open quote.
Sm	No	No	Space mode (default is "on").
Sq	Yes	Yes	Single quote.
Sx	Yes	Yes	Section Cross Reference.
Sy	Yes	Yes	Symbolic (traditional English).
Tn	Yes	Yes	Trade or type name (small Caps).
Ux	Yes	Yes	UNIX.
Xc	Yes	Yes	Extend argument list close.
Xo	Yes	Yes	Extend argument list open.

Macro names ending in ‘q’ quote remaining items on the argument list. Macro names ending in ‘o’ begin a quote which may span more than one line of input and are close quoted with the matching macro name ending in ‘c’. Enclosure macros may be nested and are limited to eight arguments.

Note: the extended argument list macros (`.Xo`, `.Xc`) and the function enclosure macros (`.Fo`, `.Fc`) are irregular. The extended list macros are used when the number of macro arguments would exceed the `troff`(1) limitation of nine arguments.

FILES

`tmac.doc`	Manual and general text domain macros.
`tmac.doc-common`	Common structural macros and definitions.
`tmac.doc-nroff`	Site dependent `nroff`(1) style file.
`tmac.doc-ditroff`	Site dependent `troff`(1) style file.
`tmac.doc-syms`	Special defines (such as the standards macro).

SEE ALSO

`mdoc.samples`(7)

NAME

mdoc.samples — tutorial sampler for writing BSD manuals with **-mdoc**

SYNOPSIS

man mdoc.samples

DESCRIPTION

A tutorial sampler for writing BSD manual pages with the **-mdoc** macro package, a *content*−based and *domain*−based formatting package for troff(1). Its predecessor, the **-man** package (see groff_man(7)), addressed page layout leaving the manipulation of fonts and other typesetting details to the individual author. In **-mdoc**, page layout macros make up the *page structure domain* which consists of macros for titles, section headers, displays and lists. Essentially items which affect the physical position of text on a formatted page. In addition to the page structure domain, there are two more domains, the manual domain and the general text domain. The general text domain is defined as macros which perform tasks such as quoting or emphasizing pieces of text. The manual domain is defined as macros that are a subset of the day to day informal language used to describe commands, routines and related BSD files. Macros in the manual domain handle command names, command line arguments and options, function names, function parameters, pathnames, variables, cross references to other manual pages, and so on. These domain items have value for both the author and the future user of the manual page. It is hoped the consistency gained across the manual set will provide easier translation to future documentation tools.

Throughout the UNIX manual pages, a manual entry is simply referred to as a man page, regardless of actual length and without sexist intention.

GETTING STARTED

Since a tutorial document is normally read when a person desires to use the material immediately, the assumption has been made that the user of this document may be impatient. The material presented in the remainder of this document is outlined as follows:

1. TROFF IDIOSYNCRASIES
   - Macro Usage.
   - Passing Space Characters in an Argument.
   - Trailing Blank Space Characters (a warning).
   - Escaping Special Characters.
2. THE ANATOMY OF A MAN PAGE
   - A manual page template.
3. INTRODUCTION OF TITLE MACROS.
4. INTRODUCTION OF MANUAL AND GENERAL TEXT DOMAINS.
   - What's in a name....
   - General Syntax.
5. MANUAL DOMAIN
   - Addresses.
   - Arguments.
   - Configuration Declarations (section four only).
   - Command Modifier.
   - Defined Variables.
   - Errno's (Section two only).
   - Environment Variables.

Function Argument.
Function Declaration.
Flags.
Functions (library routines).
Function Types.
Interactive Commands.
Literals.
Names.
Options.
Pathnames.
Variables.
Cross References.

6. GENERAL TEXT DOMAIN
AT&T Macro.
BSD Macro.
BSD/OS Macro.
FreeBSD Macro.
NetBSD Macro.
OpenBSD Macro.
UNIX Macro.
Emphasis Macro.
Enclosure/Quoting Macros
Angle Bracket Quote/Enclosure.
Bracket Quotes/Enclosure.
Double Quote macro/Enclosure.
Parenthesis Quote/Enclosure.
Single Quotes/Enclosure.
Prefix Macro.
Extended Arguments.
No−Op or Normal Text Macro.
No Space Macro.
Section Cross References.
Symbolic Macro.
References and Citations.
Trade Names (Acronyms and Type Names).

7. PAGE STRUCTURE DOMAIN
Section Headers.
Paragraphs and Line Spacing.
Keeps.
Displays.
Lists and Columns.

8. PREDEFINED STRINGS

9. DIAGNOSTICS

10. FORMATTING WITH GROFF, TROFF AND NROFF

11. BUGS

TROFF IDIOSYNCRASIES

The **-mdoc** package attempts to simplify the process of writing a man page. Theoretically, one should not have to learn the dirty details of troff(1) to use **-mdoc**; however, there are a few limitations which are unavoidable and best gotten out of the way. And, too, be forewarned, this package is *not* fast.

Macro Usage

As in troff(1), a macro is called by placing a '.' (dot character) at the beginning of a line followed by the two character name for the macro. Arguments may follow the macro separated by spaces. It is the dot character at the beginning of the line which causes troff(1) to interpret the next two characters as a macro name. To place a '.' (dot character) at the beginning of a line in some context other than a macro invocation, precede the '.' (dot) with the '\&' escape sequence. The '\&' translates literally to a zero width space, and is never displayed in the output.

In general, troff(1) macros accept up to nine arguments, any extra arguments are ignored. Most macros in **-mdoc** accept nine arguments and, in limited cases, arguments may be continued or extended on the next line (See **Extended Arguments**). A few macros handle quoted arguments (see **Passing Space Characters in an Argument** below).

Most of the **-mdoc** general text domain and manual domain macros are special in that their argument lists are *parsed* for callable macro names. This means an argument on the argument list which matches a general text or manual domain macro name and is determined to be callable will be executed or called when it is processed. In this case the argument, although the name of a macro, is not preceded by a '.' (dot). It is in this manner that many macros are nested; for example the option macro, .Op, may *call* the flag and argument macros, 'Fl' and 'Ar', to specify an optional flag with an argument:

[**-s** *bytes*] is produced by `.Op Fl s Ar bytes`

To prevent a two character string from being interpreted as a macro name, precede the string with the escape sequence '\&':

[Fl s Ar bytes] is produced by `.Op \&Fl s \&Ar bytes`

Here the strings 'Fl' and 'Ar' are not interpreted as macros. Macros whose argument lists are parsed for callable arguments are referred to as parsed and macros which may be called from an argument list are referred to as callable throughout this document and in the companion quick reference manual mdoc(7). This is a technical *faux pas* as almost all of the macros in **-mdoc** are parsed, but as it was cumbersome to constantly refer to macros as being callable and being able to call other macros, the term parsed has been used.

Passing Space Characters in an Argument

Sometimes it is desirable to give as one argument a string containing one or more blank space characters. This may be necessary to defeat the nine argument limit or to specify arguments to macros which expect particular arrangement of items in the argument list. For example, the function macro .Fn expects the first argument to be the name of a function and any remaining arguments to be function parameters. As ANSI C stipulates the declaration of function parameters in the parenthesized parameter list, each parameter is guaranteed to be at minimum a two word string. For example, *int foo*.

There are two possible ways to pass an argument which contains an embedded space. *Implementation note*: Unfortunately, the most convenient way of passing spaces in between quotes by reassigning individual arguments before parsing was fairly expensive speed wise and space wise to implement in all the macros for AT&T troff(1). It is not expensive for groff(1) but for the sake of portability, has been limited to the following macros which need it the most:

`Cd` Configuration declaration (section 4 **SYNOPSIS**)
`Bl` Begin list (for the width specifier).
`Em` Emphasized text.
`Fn` Functions (sections two and four).
`It` List items.
`Li` Literal text.
`Sy` Symbolic text.
`%B` Book titles.
`%J` Journal names.
`%O` Optional notes for a reference.
`%R` Report title (in a reference).
`%T` Title of article in a book or journal.

One way of passing a string containing blank spaces is to use the hard or unpaddable space character '`\ `', that is, a blank space preceded by the escape character '`\`'. This method may be used with any macro but has the side effect of interfering with the adjustment of text over the length of a line. `troff`(1) sees the hard space as if it were any other printable character and cannot split the string into blank or newline separated pieces as one would expect. The method is useful for strings which are not expected to overlap a line boundary. For example:

fetch(*char *str*) is created by `.Fn fetch char\ *str`

fetch(*char *str*) can also be created by `.Fn fetch "*char *str"`

If the '`\`' or quotes were omitted, `.Fn` would see three arguments and the result would be:

fetch(*char*, **str*)

For an example of what happens when the parameter list overlaps a newline boundary, see the **BUGS** section.

Trailing Blank Space Characters

`troff`(1) can be confused by blank space characters at the end of a line. It is a wise preventive measure to globally remove all blank spaces from <blank-space><end-of-line> character sequences. Should the need arise to force a blank character at the end of a line, it may be forced with an unpaddable space and the '`\&`' escape character. For example, `string\ \&`.

Sentences

To recognize the end of a sentence, `troff`(1) needs two spaces or a newline character. Since it is easy to forget about the second space, policy is to begin new sentences on a new line.

Escaping Special Characters

Special characters like the newline character '`\n`', are handled by replacing the '`\`' with '`\e`' (e.g. `\en`) to preserve the backslash.

THE ANATOMY OF A MAN PAGE

The body of a man page is easily constructed from a basic template found in the file:

```
.\" /usr/share/misc/mdoc.template:
.\" The following six lines are required.
.Dd Month day, year
.Os
.Dt DOCUMENT_TITLE SECTION_NUMBER [MACHINE]
.Sh NAME
.\" This next request is for sections 2 and 3 only; see next comment.
```

```
.Sh LIBRARY
.Sh SYNOPSIS
.Sh DESCRIPTION
.\" The following requests should be uncommented and
.\" used where appropriate.
.\" This next request is for
.\" sections 1 and 8 exit statuses only.
.\" .Sh EXIT STATUS
.\" This next request is for sections 2 and 3 function return
.\"     values only.
.\" .Sh RETURN VALUES
.\" This next request is for sections 1, 6, 7 & 8 only
.\" .Sh ENVIRONMENT
.\" .Sh FILES
.\" .Sh EXAMPLES
.\" This next request is for sections 1, 6, 7 & 8 only
.\"     (command return values (to shell) and
.\"       fprintf/stderr type diagnostics)
.\" .Sh DIAGNOSTICS
.\" The next request is for sections 2 and 3 error
.\" and signal handling only.
.\" .Sh ERRORS
.\" .Sh SEE ALSO
.\" .Sh STANDARDS
.\" .Sh HISTORY
.\" .Sh AUTHORS
.\" .Sh BUGS
.\" .Sh SECURITY CONSIDERATIONS
```

The first items in the template are the macros (`.Dd`, `.Os`, `.Dt`); the document date, the operating system the man page or subject source is developed or modified for (should have no argument by default), and the man page title (*in upper case*) along with the section of the manual the page belongs in, and optionally the machine if it is machine specific. These macros identify the page, and are discussed below in **TITLE MACROS**.

The remaining items in the template are section headers (`.Sh`); of which **NAME**, **SYNOPSIS** and **DESCRIPTION** are mandatory. The headers are discussed in **PAGE STRUCTURE DOMAIN**, after presentation of **MANUAL DOMAIN**. Several content macros are used to demonstrate page layout macros; reading about content macros before page layout macros is recommended.

TITLE MACROS

The title macros are the first portion of the page structure domain, but are presented first and separate for someone who wishes to start writing a man page yesterday. Three header macros designate the document title or manual page title, the operating system, and the date of authorship. These macros are called once at the very beginning of the document and are used to construct the headers and footers only.

`.Dt DOCUMENT_TITLE SECTION_NUMBER [MACHINE]`

The document title is the subject of the man page and must be in CAPITALS due to troff limitations. The section number may be 1, ..., 9, and the machine should be the machine the man page is for (that is, the NetBSD port to which it applies).

`.Os operating_system release#`

This should have no argument on NetBSD man pages by default. Otherwise, the name of the operating system should be the common acronym, e.g. BSD or ATT. The release should be the standard

release nomenclature for the system specified, e.g. 4.3, 4.3+Tahoe, V.3, V.4. Unrecognized arguments are displayed as given in the page footer. For instance, a typical footer might be:

```
.Os BSD 4.3
```

or for a locally produced set

```
.Os CS Department
```

The Berkeley default, `.Os` without an argument, has been defined as the current NetBSD version, see `/usr/share/tmac/tmac.doc-common`. Note, if the `.Os` macro is not present, the bottom left corner of the page will be ugly.

`.Dd month day, year`

The date of the last significant revision to the manual page; the date should be written formally:

```
January 25, 1989
```

Note that the date must not be placed in quotes!

MANUAL DOMAIN

What's in a name...

The manual domain macro names are derived from the day to day informal language used to describe commands, subroutines and related files. Slightly different variations of this language are used to describe the three different aspects of writing a man page. First, there is the description of **-mdoc** macro request usage. Second is the description of a UNIX command *with* **-mdoc** macros and third, the description of a command to a user in the verbal sense; that is, discussion of a command in the text of a man page.

In the first case, `troff`(1) macros are themselves a type of command; the general syntax for a troff command is:

```
.Va argument1 argument2 ... argument9
```

The `.Va` is a macro command or request, and anything following it is an argument to be processed. In the second case, the description of a UNIX command using the content macros is a bit more involved; a typical **SYNOPSIS** command line might be displayed as:

filter [**-flag**] *infile outfile*

Here, **filter** is the command name and the bracketed string **-flag** is a *flag* argument designated as optional by the option brackets. In **-mdoc** terms, *infile* and *outfile* are called *arguments*. The macros which formatted the above example:

```
.Nm filter
.Op Fl flag
.Ar infile outfile
```

In the third case, discussion of commands and command syntax includes both examples above, but may add more detail. The arguments *infile* and *outfile* from the example above might be referred to as *operands* or *file arguments*. Some command line argument lists are quite long:

make [**-eiknqrstv**] [**-D** *variable*] [**-d** *flags*] [**-f** *makefile*] [**-I** *directory*] [**-j** *max_jobs*] [*variable=value*] [*target ...*]

Here one might talk about the command **make** and qualify the argument *makefile*, as an argument to the flag, **-f**, or discuss the optional file operand *target*. In the verbal context, such detail can prevent confusion, however the **-mdoc** package does not have a macro for an argument *to* a flag. Instead the 'Ar' argument macro is used for an operand or file argument like *target* as well as an argument to a flag like *variable*. The make command line was produced from:

```
.Nm make
.Op Fl eiknqrstv
.Op Fl D Ar variable
.Op Fl d Ar flags
.Op Fl f Ar makefile
.Op Fl I Ar directory
.Op Fl j Ar max_jobs
.Op Ar variable=value
.Bk -words
.Op Ar target ...
.Ek
```

The `.Bk` and `.Ek` macros are explained in **Keeps**.

General Syntax

The manual domain and general text domain macros share a similar syntax with a few minor deviations: `.Ar`, `.Fl`, `.Nm`, and `.Pa` differ only when called without arguments; `.Fn` and `.Xr` impose an order on their argument lists and the `.Op` and `.Fn` macros have nesting limitations. All content macros are capable of recognizing and properly handling punctuation, provided each punctuation character is separated by a leading space. If a request is given:

```
.Li sptr, ptr),
```

The result is:

```
sptr, ptr),
```

The punctuation is not recognized and all is output in the literal font. If the punctuation is separated by a leading white space:

```
.Li sptr , ptr ) ,
```

The result is:

```
sptr , ptr ) ,
```

The punctuation is now recognized and is output in the default font distinguishing it from the strings in literal font.

To remove the special meaning from a punctuation character escape it with ‘\&’. `troff`(1) is limited as a macro language, and has difficulty when presented with a string containing a member of the mathematical, logical or quotation set:

```
{+,−,/,*,%,<,>,≤,≥,=,==,&,‘,’,"}
```

The problem is that `troff`(1) may assume it is supposed to actually perform the operation or evaluation suggested by the characters. To prevent the accidental evaluation of these characters, escape them with ‘\&’. Typical syntax is shown in the first content macro displayed below, `.Ad`.

Address Macro

The address macro identifies an address construct of the form addr1[,addr2[,addr3]].

```
Usage: .Ad address ...
       .Ad addr1
               addr1
       .Ad addr1 .
               addr1.
```

```
.Ad addr1 , file2
        addr1, file2
.Ad f1 , f2 , f3 :
        f1, f2, f3:
.Ad addr ) ) ,
        addr)),
```

It is an error to call `.Ad` without arguments. `.Ad` is callable by other macros and is parsed.

Argument Macro

The `.Ar` argument macro may be used whenever a command line argument is referenced.

```
Usage: .Ar argument ...
       .Ar          file ...
       .Ar file1    file1
       .Ar file1 . file1.
       .Ar file1 file2
                    file1 file2
       .Ar f1 f2 f3 :
                    f1 f2 f3:
       .Ar file ) ) ,
                    file)),
```

If `.Ar` is called without arguments '*`file ...`*' is assumed. The `.Ar` macro is parsed and is callable.

Configuration Declaration (section four only)

The `.Cd` macro is used to demonstrate a `config`(1) declaration for a device interface in a section four manual. This macro accepts quoted arguments (double quotes only).

`device le0 at scode?` produced by: `.Cd device le0 at scode?`.

Command Modifier

The command modifier is identical to the `.Fl` (flag) command with the exception the `.Cm` macro does not assert a dash in front of every argument. Traditionally flags are marked by the preceding dash, some commands or subsets of commands do not use them. Command modifiers may also be specified in conjunction with interactive commands such as editor commands. See **Flags**.

Defined Variables

A variable which is defined in an include file is specified by the macro `.Dv`.

```
Usage: .Dv defined_variable ...
       .Dv MAXHOSTNAMELEN
                     MAXHOSTNAMELEN
       .Dv TIOCGPGRP )
                     TIOCGPGRP)
```

It is an error to call `.Dv` without arguments. `.Dv` is parsed and is callable.

Errno's (Section two only)

The `.Er` errno macro specifies the error return value for section two library routines. The second example below shows `.Er` used with the `.Bq` general text domain macro, as it would be used in a section two manual page.

```
Usage: .Er ERRNOTYPE ...
```

```
.Er ENOENT
          ENOENT
.Er ENOENT ) ;
          ENOENT);
.Bq Er ENOTDIR
          [ENOTDIR]
```

It is an error to call `.Er` without arguments. The `.Er` macro is parsed and is callable.

Environment Variables

The `.Ev` macro specifies an environment variable.

```
Usage: .Ev argument ...
       .Ev DISPLAY
                 DISPLAY
       .Ev PATH .
                 PATH.
       .Ev PRINTER ) ) ,
                 PRINTER)),
```

It is an error to call `.Ev` without arguments. The `.Ev` macro is parsed and is callable.

Function Argument

The `.Fa` macro is used to refer to function arguments (parameters) outside of the **SYNOPSIS** section of the manual or inside the **SYNOPSIS** section should a parameter list be too long for the `.Fn` macro and the enclosure macros `.Fo` and `.Fc` must be used. `.Fa` may also be used to refer to structure members.

```
Usage: .Fa function_argument ...
       .Fa d_namlen ) ) ,
                    d_namlen)),
       .Fa iov_len    iov_len
```

It is an error to call `.Fa` without arguments. `.Fa` is parsed and is callable.

Function Declaration

The `.Fd` macro is used in the **SYNOPSIS** section with section two, three or nine functions. The `.Fd` macro does not call other macros and is not callable by other macros.

```
Usage: .Fd include_file (or defined variable)
```

In the **SYNOPSIS** section a `.Fd` request causes a line break if a function has already been presented and a break has not occurred. This leaves a nice vertical space in between the previous function call and the declaration for the next function.

Flags

The `.Fl` macro handles command line flags. It prepends a dash, ‘−’, to the flag. For interactive command flags, which are not prepended with a dash, the `.Cm` (command modifier) macro is identical, but without the dash.

```
Usage: .Fl argument ...
       .Fl          -
       .Fl cfv      -cfv
       .Fl cfv .    -cfv.
```

```
.Fl s v t    -s -v -t
.Fl - ,      --,
.Fl xyz ) ,  -xyz),
```

The `.Fl` macro without any arguments results in a dash representing stdin/stdout. Note that giving `.Fl` a single dash, will result in two dashes. The `.Fl` macro is parsed and is callable.

Functions (library routines)

The `.Fn` macro is modeled on ANSI C conventions.

```
Usage: .Fn [type] function [[type] parameters ... ]
.Fn getchar                          getchar()
.Fn strlen ) ,                       strlen()),
.Fn "int align" "const * char *sptrs",
                                     int align(const * char *sptrs),
```

It is an error to call `.Fn` without any arguments. The `.Fn` macro is parsed and is callable, note that any call to another macro signals the end of the `.Fn` call (it will close-parenthesis at that point).

For functions that have more than eight parameters (and this is rare), the macros `.Fo` (function open) and `.Fc` (function close) may be used with `.Fa` (function argument) to get around the limitation. For example:

```
.Ft "int"
.Fo "res_mkquery"
.Fa "int op"
.Fa "char *dname"
.Fa "int class"
.Fa "int type"
.Fa "char *data"
.Fa "int datalen"
.Fa "struct rrec *newrr"
.Fa "char *buf"
.Fa "int buflen"
.Fc
```

Produces:

int **res_mkquery**(*int op*, *char *dname*, *int class*, *int type*, *char *data*, *int datalen*, *struct rrec *newrr*, *char *buf*, *int buflen*)

The `.Fo` and `.Fc` macros are parsed and are callable. In the **SYNOPSIS** section, the function will always begin at the beginning of line. If there is more than one function presented in the **SYNOPSIS** section and a function type has not been given, a line break will occur, leaving a nice vertical space between the current function name and the one prior. At the moment, `.Fn` does not check its word boundaries against troff line lengths and may split across a newline ungracefully. This will be fixed in the near future.

Function Type

This macro is intended for the **SYNOPSIS** section. It may be used anywhere else in the man page without problems, but its main purpose is to present the function type in kernel normal form for the **SYNOPSIS** of sections two and three (it causes a page break allowing the function name to appear on the next line).

```
Usage: .Ft type ...
       .Ft struct stat struct stat
```

The `.Ft` request is not callable by other macros.

The .In (#include statement) macro is the short form for
.Ft #include <header.h>.
It specifies the C header file as being included in a C program. It also causes a line break, and is neither callable nor parsed.

```
Usage: .In ⟨header file⟩

       .In stdio.h <stdio.h>
```

Interactive Commands

The .Ic macro designates an interactive or internal command.

```
Usage: .Ic command ...
       .Ic :wq            :wq
       .Ic do while {...} do while {...}
       .Ic setenv , unsetenv
                          setenv, unsetenv
```

It is an error to call .Ic without arguments. The .Ic macro is parsed and is callable.

Literals

The .Li literal macro may be used for special characters, variable constants, anything which should be displayed as it would be typed.

```
Usage: .Li argument ...
       .Li \en  \n
       .Li M1 M2 M3 ;
                M1 M2 M3;
       .Li cntrl-D ) ,
                cntrl-D),
       .Li 1024 ...
                1024 ...
```

The .Li macro is parsed and is callable.

Name Macro

The .Nm macro is used for the document title or subject name. It has the peculiarity of remembering the first argument it was called with, which should always be the subject name of the page. When called without arguments, .Nm regurgitates this initial name for the sole purpose of making less work for the author. If trailing punctuation is required with this feature, use "" as a first argument to .Nm. Note: a section two, three or nine document function name is addressed with the .Nm in the **NAME** section, and with .Fn in the **SYNOPSIS** and remaining sections. For interactive commands, such as the while command keyword in csh(1), the .Ic macro should be used. While the .Ic is nearly identical to .Nm, it can not recall the first argument it was invoked with.

```
Usage: .Nm argument ...
       .Nm mdoc.samples
                   mdoc.samples
       .Nm \-mdoc  −mdoc
       .Nm foo ) ) ,
                   foo)),
       .Nm         mdoc.samples
       .Nm "" :    mdoc.samples:
```

The .Nm macro is parsed and is callable.

Options

The .Op macro places option brackets around the any remaining arguments on the command line, and places any trailing punctuation outside the brackets. The macros .Oc and .Oo may be used across one or more lines.

```
Usage: .Op options ...
.Op                     [ ]
.Op Fl k                [ -k ]
.Op Fl k ) .            [ -k ]).
.Op Fl k Ar kookfile    [ -k kookfile ]
.Op Fl k Ar kookfile ,
                        [ -k kookfile ],
.Op Ar objfil Op Ar corfil
                        [ objfil [ corfil ]]
.Op Fl c Ar objfil Op Ar corfil ,
                        [ -c objfil [ corfil ]],
.Op word1 word2         [ word1 word2 ]
```

The .Oc and .Oo macros:

```
.Oo
.Op Fl k Ar kilobytes
.Op Fl i Ar interval
.Op Fl c Ar count
.Oc
```

Produce: [[**-k** *kilobytes*] [**-i** *interval*] [**-c** *count*]]

The macros .Op, .Oc and .Oo are parsed and are callable.

Pathnames

The .Pa macro formats path or file names.

```
Usage: .Pa pathname
       .Pa /usr/share /usr/share
       .Pa /tmp/fooXXXXX ) .
                         /tmp/fooXXXXX).
```

The .Pa macro is parsed and is callable.

Variables

Generic variable reference:

```
Usage: .Va variable ...
       .Va count
                count
       .Va settimer,
                settimer,
       .Va int *prt ) :
                int *prt):
       .Va char s ] ) ) ,
                char s])),
```

It is an error to call .Va without any arguments. The .Va macro is parsed and is callable.

Manual Page Cross References

The .Xr macro expects the first argument to be a manual page name, and the second argument, if it exists, to be either a section page number or punctuation. Any remaining arguments are assumed to be punctuation.

```
Usage: .Xr man_page [1,...,9]
       .Xr mdoc mdoc
       .Xr mdoc ,
                mdoc,
       .Xr mdoc 7
                mdoc(7)
       .Xr mdoc 7 ) ) ,
                mdoc(7))),
```

The .Xr macro is parsed and is callable. It is an error to call .Xr without any arguments.

GENERAL TEXT DOMAIN

AT&T Macro

```
Usage: .At [v1 .. v7 | 32v | V.1 | V.4] ...
       .At              AT&T UNIX
       .At v6 .         Version 6 AT&T UNIX.
```

The .At macro is *not* parsed and *not* callable. It accepts at most two arguments.

BSD Macro

```
Usage: .Bx [Version/release] ...
       .Bx     BSD
       .Bx 4.3 .
               4.3BSD.
```

The .Bx macro is parsed and is callable.

BSD/OS Macro

```
Usage: .Bsx [Version/release] ...
       .Bsx      BSD/OS
       .Bsx 4.1 .
                 BSD/OS 4.1.
```

The .Bsx macro is parsed and is callable.

FreeBSD Macro

```
Usage: .Fx [Version/release] ...
       .Fx        FreeBSD
       .Fx 2.2 .  FreeBSD 2.2.
```

The .Fx macro is parsed and is callable.

NetBSD Macro

```
Usage: .Nx [Version/release] ...
       .Nx        NetBSD
       .Nx 1.4 .  NetBSD 1.4.
```

The .Nx macro is parsed and is callable.

OpenBSD Macro

```
Usage: .Ox [Version/release] ...
       .Ox          OpenBSD
       .Ox 2.7 .    OpenBSD 2.7.
```

The `.Ox` macro is parsed and is callable.

UNIX Macro

```
Usage: .Ux ...
       .Ux        UNIX
```

The `.Ux` macro is parsed and is callable.

Emphasis Macro

Text may be stressed or emphasized with the `.Em` macro. The usual font for emphasis is italic.

```
Usage: .Em argument ...
       .Em does not
                 does not
       .Em exceed 1024 .
                 exceed 1024.
       .Em vide infra ) ) ,
                 vide infra)),
```

The `.Em` macro is parsed and is callable. It is an error to call `.Em` without arguments.

Enclosure and Quoting Macros

The concept of enclosure is similar to quoting. The object being to enclose one or more strings between a pair of characters like quotes or parentheses. The terms quoting and enclosure are used interchangeably throughout this document. Most of the one line enclosure macros end in small letter '`q`' to give a hint of quoting, but there are a few irregularities. For each enclosure macro there is also a pair of open and close macros which end in small letters '`o`' and '`c`' respectively. These can be used across one or more lines of text and while they have nesting limitations, the one line quote macros can be used inside of them.

Quote	*Close*	*Open*	*Function*	*Result*
`.Aq`	`.Ac`	`.Ao`	Angle Bracket Enclosure	<string>
`.Bq`	`.Bc`	`.Bo`	Bracket Enclosure	[string]
`.Dq`	`.Dc`	`.Do`	Double Quote	‘‘string’’
`.Ec`	`.Eo`		Enclose String (in XX)	XXstringXX
`.Pq`	`.Pc`	`.Po`	Parenthesis Enclosure	(string)
`.Ql`			Quoted Literal	‘st’ or string
`.Qq`	`.Qc`	`.Qo`	Straight Double Quote	"string"
`.Sq`	`.Sc`	`.So`	Single Quote	‘string’

Except for the irregular macros noted below, all of the quoting macros are parsed and callable. All handle punctuation properly, as long as it is presented one character at a time and separated by spaces. The quoting macros examine opening and closing punctuation to determine whether it comes before or after the enclosing string. This makes some nesting possible.

`.Ec`, `.Eo` These macros expect the first argument to be the opening and closing strings respectively.

`.Ql` The quoted literal macro behaves differently for `troff`(1) than `nroff`(1). If formatted with `nroff`(1), a quoted literal is always quoted. If formatted with troff, an item is only quoted if the width of the item is less than three constant width characters. This is to make short strings more visible where the font change to literal (constant width) is less noticeable.

`.Pf` The prefix macro is not callable, but it is parsed:

```
.Pf ( Fa name2
```
becomes (*name2*.

`.Ns` The `.Ns` (no space) macro, which *is* callable, performs the analogous suffix function.

`.Ap` The `.Ap` macro inserts an apostrophe and exits any special text modes, continuing in `.No` mode.

Examples of quoting:

`.Aq`	⟨⟩
`.Aq Ar ctype.h ) ,`	⟨*ctype.h*⟩),
`.Bq`	[ ]
`.Bq Em Greek , French .`	[*Greek*, *French*].
`.Dq`	“”
`.Dq string abc .`	“string abc”.
`.Dq ´^[A-Z]´`	“´^[A-Z]´”
`.Ql man mdoc`	`man mdoc`
`.Qq`	""
`.Qq string ) ,`	"string"),
`.Qq string Ns ),`	"string),"
`.Sq`	‘’
`.Sq string`	‘string’
`.Em or Ap ing`	*or*’ing

For a good example of nested enclosure macros, see the `.Op` option macro. It was created from the same underlying enclosure macros as those presented in the list above. The `.Xo` and `.Xc` extended argument list macros were also built from the same underlying routines and are a good example of **`-mdoc`** macro usage at its worst.

No−Op or Normal Text Macro

The macro `.No` is a hack for words in a macro command line which should *not* be formatted and follows the conventional syntax for content macros.

Space Macro

The `.Ns` macro eliminates unwanted spaces in between macro requests. It is useful for old style argument lists where there is no space between the flag and argument:

```
.Op Fl I Ns Ar directory
```
produces [**-I***directory*]

Note: the `.Ns` macro always invokes the `.No` macro after eliminating the space unless another macro name follows it. The macro `.Ns` is parsed and is callable.

Section Cross References

The `.Sx` macro designates a reference to a section header within the same document. It is parsed and is callable.

```
.Sx FILES
```
FILES

Symbolic

The symbolic emphasis macro is generally a boldface macro in either the symbolic sense or the traditional English usage.

```
Usage: .Sy symbol ...
       .Sy Important Notice
```

Important Notice

The `.Sy` macro is parsed and is callable. Arguments to `.Sy` may be quoted.

References and Citations

The following macros make a modest attempt to handle references. At best, the macros make it convenient to manually drop in a subset of refer style references.

`.Rs`	Reference Start. Causes a line break and begins collection of reference information until the reference end macro is read.
`.Re`	Reference End. The reference is printed.
`.%A`	Reference author name, one name per invocation.
`.%B`	Book title.
`.%C`	City/place.
`.%D`	Date.
`.%J`	Journal name.
`.%N`	Issue number.
`.%O`	Optional information.
`.%P`	Page number.
`.%R`	Report name.
`.%T`	Title of article.
`.%V`	Volume(s).

The macros beginning with '%' are not callable, and are parsed only for the trade name macro which returns to its caller. (And not very predictably at the moment either.) The purpose is to allow trade names to be pretty printed in `troff`(1)/**`ditroff`** output.

Trade Names (or Acronyms and Type Names)

The trade name macro is generally a small caps macro for all upper case words longer than two characters.

```
Usage: .Tn symbol ...
       .Tn DEC
            DEC
       .Tn ASCII
            ASCII
```

The `.Tn` macro is parsed and is callable by other macros.

Extended Arguments

The `.Xo` and `.Xc` macros allow one to extend an argument list on a macro boundary. Argument lists cannot be extended within a macro which expects all of its arguments on one line such as `.Op`.

Here is an example of `.Xo` using the space mode macro to turn spacing off:

```
.Sm off
.It Xo Sy I Ar operation
.No \en Ar count No \en
.Xc
.Sm on
```

Produces

I*operation*\n*count*\n

Another one:

```
.Sm off
.It Cm S No / Ar old_pattern Xo
.No / Ar new_pattern
.No / Op Cm g
.Xc
.Sm on
```

Produces

S/*old_pattern*/*new_pattern*/[**g**]

Another example of `.Xo` and using enclosure macros: Test the value of an variable.

```
.It Xo
.Ic .ifndef
.Oo \&! Oc Ns Ar variable
.Op Ar operator variable ...
.Xc
```

Produces

.ifndef [!]*variable* [*operator variable ...*]

All of the above examples have used the `.Xo` macro on the argument list of the `.It` (list-item) macro. The extend macros are not used very often, and when they are it is usually to extend the list-item argument list. Unfortunately, this is also where the extend macros are the most finicky. In the first two examples, spacing was turned off; in the third, spacing was desired in part of the output but not all of it. To make these macros work in this situation make sure the `.Xo` and `.Xc` macros are placed as shown in the third example. If the `.Xo` macro is not alone on the `.It` argument list, spacing will be unpredictable. The `.Ns` (no space macro) must not occur as the first or last macro on a line in this situation. Out of 900 manual pages (about 1500 actual pages) currently released with BSD only fifteen use the `.Xo` macro.

PAGE STRUCTURE DOMAIN

Section Headers

The first three `.Sh` section header macros listed below are required in every man page. The remaining section headers are recommended at the discretion of the author writing the manual page. The `.Sh` macro can take up to nine arguments. It is parsed but is not callable.

`.Sh NAME` The `.Sh NAME` macro is mandatory. If not specified, the headers, footers and page layout defaults will not be set and things will be rather unpleasant. The **NAME** section consists of at least three items. The first is the `.Nm` name macro naming the subject of the man page. The second is the Name Description macro, `.Nd`, which separates the subject name from the third item, which is the description. The description should be the most terse and lucid possible, as the space available is small.

`.Sh SYNOPSIS`

The **SYNOPSIS** section describes the typical usage of the subject of a man page. The macros required are either `.Nm`, `.Cd`, `.Fn`, (and possibly `.Fo`, `.Fc`, `.Fd`, `.Ft` macros). The function name macro `.Fn` is required for manual page sections 2 and 3, the command and general name macro `.Nm` is required for sections 1, 5, 6, 7, 8. Section 4 manuals require a `.Nm`, `.Fd` or a `.Cd` configuration device usage macro. Several other macros may be necessary to produce the synopsis line as shown below:

cat [**-benstuv**] [**-**] *file ...*

The following macros were used:

```
.Nm cat
.Op Fl benstuv
.Op Fl
.Ar
```

Note: The macros `.Op`, `.Fl`, and `.Ar` recognize the pipe bar character '|', so a command line such as:

```
.Op Fl a | Fl b
```

will not go orbital. `troff`(1) normally interprets a | as a special operator. See **PREDEFINED STRINGS** for a usable | character in other situations.

`.Sh DESCRIPTION`

In most cases the first text in the **DESCRIPTION** section is a brief paragraph on the command, function or file, followed by a lexical list of options and respective explanations. To create such a list, the `.Bl` begin-list, `.It` list-item and `.El` end-list macros are used (see **Lists and Columns** below).

The following `.Sh` section headers are part of the preferred manual page layout and must be used appropriately to maintain consistency. They are listed in the order in which they would be used.

`.Sh ENVIRONMENT`

The **ENVIRONMENT** section should reveal any related environment variables and clues to their behavior and/or usage.

`.Sh EXAMPLES`

There are several ways to create examples. See the **EXAMPLES** section below for details.

`.Sh FILES`

Files which are used or created by the man page subject should be listed via the `.Pa` macro in the **FILES** section.

`.Sh SEE ALSO`

References to other material on the man page topic and cross references to other relevant man pages should be placed in the **SEE ALSO** section. Cross references are specified using the `.Xr` macro. At this time `refer`(1) style references are not accommodated.

It is recommended that the cross references are sorted on the section number, and then alphabetically on the names within a section.

`.Sh STANDARDS`

If the command, library function or file adheres to a specific implementation such as IEEE Std 1003.2 ("POSIX.2") or ANSI X3.159-1989 ("ANSI C89") this should be noted here. If the command does not adhere to any standard, its history should be noted in the **HISTORY** section.

`.Sh HISTORY`

Any command which does not adhere to any specific standards should be outlined historically in this section.

`.Sh AUTHORS`

Credits, if need be, should be placed here.

`.Sh DIAGNOSTICS`

Diagnostics from a command should be placed in this section.

`.Sh ERRORS`
Specific error handling, especially from library functions (man page sections 2 and 3) should go here. The `.Er` macro is used to specify an errno.

`.Sh BUGS` Blatant problems with the topic go here...

User specified `.Sh` sections may be added, for example, this section was set with:

```
.Sh PAGE STRUCTURE DOMAIN
```

Paragraphs and Line Spacing.

`.Pp` The `.Pp` paragraph command may be used to specify a line space where necessary. The macro is not necessary after a `.Sh` or `.Ss` macro or before a `.Bl` macro. (The `.Bl` macro asserts a vertical distance unless the -compact flag is given).

Keeps

The only keep that is implemented at this time is for words. The macros are `.Bk` (begin-keep) and `.Ek` (end-keep). The only option that `.Bk` accepts is **`-words`** and is useful for preventing line breaks in the middle of options. In the example for the make command line arguments (see **What's in a name**), the keep prevented `nroff`(1) from placing the flag and the argument on separate lines. (Actually, the option macro formerly prevented this from occurring, but was dropped when the decision (religious) was made to force right justified margins in `troff`(1) as options in general look atrocious when spread across a sparse line. More work needs to be done with the keep macros, a **`-line`** option needs to be added.)

Examples and Displays

There are six types of displays: a quickie, one-line indented display `.D1`; a quickie, one-line literal display `.Dl`; and block-literal, block-filled, block-unfilled, and block-ragged which use the `.Bd` begin-display and `.Ed` end-display macros.

`.D1` (D-one) Display one line of indented text. This macro is parsed, but it is not callable.

`-ldghfstru`

The above was produced by: `.D1 Fl ldghfstru`.

`.Dl` (D-ell) Display one line of indented *literal* text. The `.Dl` example macro has been used throughout this file. It allows the indent (display) of one line of text. Its default font is set to constant width (literal) however it is parsed and will recognize other macros. It is however not callable.

```
% ls -ldg /usr/local/bin
```

The above was produced by: `.Dl % ls -ldg /usr/local/bin`.

`.Bd` Begin-display. The `.Bd` display must be ended with the `.Ed` macro. Displays may be nested within lists, but may *not* contain other displays; this also prohibits nesting of `.D1` and `.Dl` one-line displays. `.Bd` has the following syntax:

```
.Bd display-type [-offset offset_value] [-compact]
```

The display-type must be one of the four types (**`-ragged`**, **`-unfilled`**, **`-filled`**, **`-literal`**) and may have an offset specifier for indentation: `.Bd`.

`-ragged`	Fill, but do not adjust the right margin.
`-unfilled`	Do not fill: display a block of text as typed, the right (and left) margin edges are left ragged.
`-filled`	Display a filled (formatted) block. The block of text is formatted (the edges are filled − not left unjustified).

-literal Display a literal block, useful for source code or simple tabbed or spaced text.

-file *file_name* The file name following the **-file** flag is read and displayed. Literal mode is asserted and tabs are set at 8 constant width character intervals, however any troff(1)/**-mdoc** commands in file will be processed.

-offset *string* If **-offset** is specified with one of the following strings, the string is interpreted to indicate the level of indentation for the forthcoming block of text:

- *left* Align block on the current left margin, this is the default mode of .Bd.
- *center* Supposedly center the block. At this time unfortunately, the block merely gets left aligned about an imaginary center margin.
- *indent* Indents by one default indent value or tab. The default indent value is also used for the .D1 display so one is guaranteed the two types of displays will line up. This indent is normally set to 6n or about two thirds of an inch (six constant width characters).
- *indent-two* Indents two times the default indent value.
- *right* This *left* aligns the block about two inches from the right side of the page. This macro needs work and perhaps may never do the right thing by troff(1).

.Ed End-display.

Tagged Lists and Columns

There are several types of lists which may be initiated with the .Bl begin-list macro. Items within the list are specified with the .It item macro and each list must end with the .El macro. Lists other than -enum may be nested within themselves and within displays. The use of columns inside of lists or lists inside of columns is unproven.

In addition, several list attributes may be specified such as the width of a tag, the list offset, and compactness (blank lines between items allowed or disallowed). Most of this document has been formatted with a tag style list (**-tag**). For a change of pace, the list-type used to present the list-types is an over-hanging list (**-ohang**). This type of list is quite popular with TeX users, but might look a bit funny after having read many pages of tagged lists. The following list types are accepted by .Bl:

-bullet
-dash
-enum
-hyphen
-item

These five are the simplest types of lists. Once the .Bl macro has been given, items in the list are merely indicated by a line consisting solely of the .It macro. For example, the source text for a simple enumerated list would look like:

```
.Bl -enum -compact
.It
Item one goes here.
.It
And item two here.
.It
Lastly item three goes here.
```

```
.El
```

The results:

1. Item one goes here.
2. And item two here.
3. Lastly item three goes here.

A simple bullet list construction:

```
.Bl -bullet -compact
.It
Bullet one goes here.
.It
Bullet two here.
.El
```

Produces:

- Bullet one goes here.
- Bullet two here.

-inset
-diag
-hang
-ohang
-tag

These list-types collect arguments specified with the `.It` macro and create a label which may be *inset* into the forthcoming text, *hanged* from the forthcoming text, *overhanged* from above and not indented or *tagged*. This list was constructed with the **-ohang** list-type. The `.It` macro is parsed only for the inset, hang and tag list-types and is not callable. Here is an example of inset labels:

Tag The tagged list (also called a tagged paragraph) is the most common type of list used in the Berkeley manuals. Use a **-width** attribute as described below.

Diag Diag lists create section four diagnostic lists and are similar to inset lists except callable macros are ignored.

Hang Hanged labels are a matter of taste.

Ohang Overhanging labels are nice when space is constrained.

Inset Inset labels are useful for controlling blocks of paragraphs and are valuable for converting **-mdoc** manuals to other formats.

Here is the source text which produced the above example:

```
.Bl -inset -offset indent
.It Em Tag
The tagged list (also called a tagged paragraph) is the
most common type of list used in the Berkeley manuals.
Use a
.Fl width
attribute as described below.
.It Em Diag
Diag lists create section four diagnostic lists
and are similar to inset lists except callable
macros are ignored.
.It Em Hang
```

```
Hanged labels are a matter of taste.
.It Em Ohang
Overhanging labels are nice when space is constrained.
.It Em Inset
Inset labels are useful for controlling blocks of
paragraphs and are valuable for converting
.Nm \-mdoc
manuals to other formats.
.El
```

Here is a hanged list with just two items:

Hanged labels appear similar to tagged lists when the label is smaller than the label width.

Longer hanged list labels blend in to the paragraph unlike tagged paragraph labels.

And the unformatted text which created it:

```
.Bl -hang -offset indent
.It Em Hanged
labels appear similar to tagged lists when the
label is smaller than the label width.
.It Em Longer hanged list labels
blend in to the paragraph unlike
tagged paragraph labels.
.El
```

The tagged list which follows uses a width specifier to control the width of the tag.

SL sleep time of the process (seconds blocked)
PAGEIN number of disk I/O's resulting from references by the process to pages not loaded in core.
UID numerical user-id of process owner
PPID numerical id of parent of process priority (non-positive when in non-interruptible wait)

The raw text:

```
.Bl -tag -width "PAGEIN" -compact -offset indent
.It SL
sleep time of the process (seconds blocked)
.It PAGEIN
number of disk
.Tn I/O Ns 's
resulting from references
by the process to pages not loaded in core.
.It UID
numerical user-id of process owner
.It PPID
numerical id of parent of process priority
(non-positive when in non-interruptible wait)
.El
```

Acceptable width specifiers:

-width *Fl* sets the width to the default width for a flag. All callable macros have a default width value. The `.Fl`, value is presently set to ten constant width characters or about five sixth of an inch.

-width *24n*
sets the width to 24 constant width characters or about two inches. The 'n' is absolutely necessary for the scaling to work correctly.

-width *ENAMETOOLONG*
sets width to the constant width length of the string given.

-width *"int mkfifo"*
again, the width is set to the constant width of the string given.

If a width is not specified for the tag list type, the first time .It is invoked, an attempt is made to determine an appropriate width. If the first argument to .It is a callable macro, the default width for that macro will be used as if the macro name had been supplied as the width. However, if another item in the list is given with a different callable macro name, a new and nested list is assumed. This effectively means that **-width** is required for the tag list type.

-column
This list type generates multiple columns. The number of columns and the width of each column is determined by the arguments to the **-column** list. Each .It argument is parsed to make a row, each column within the row is a separate argument separated by a tab or the .Ta macro.
The table:

String	Nroff	Troff
≤	<=	≤
≥	>=	≥

was produced by:

```
.Bl -column "String" "Nroff" "Troff" -offset indent
.It Sy "String" Ta Sy "Nroff" Ta Sy "Troff"
.It Li "≤" Ta \&<\&= Ta \*(≤
.It Li "≥" Ta \&>\&= Ta \*(≥
.El
```

PREDEFINED STRINGS

The following strings are predefined and may be used by preceding with the troff string interpreting sequence *(xx where *xx* is the name of the defined string or as *x where *x* is the name of the string. The interpreting sequence may be used any where in the text.

String	Nroff	Troff
≤	<=	≤
≥	>=	≥
Rq	”	”
Lq	“	“
ua	^	↑
aa	’	´
ga	`	`
q	"	"
Pi	pi	π
Ne	!=	≠
Le	≤	≤
Ge	≥	≥
Lt	<	>

Gt	>	<
Pm	+-	±
If	infinity	∞
Na	*NaN*	*NaN*
Ba	|	|

Note: The string named ‘q’ should be written as *q since it is only one char.

DIAGNOSTICS

The debugging facilities for **-mdoc** are limited, but can help detect subtle errors such as the collision of an argument name with an internal register or macro name. (A what?) A register is an arithmetic storage class for troff(1) with a one or two character name. All registers internal to **-mdoc** for troff(1) and **ditroff** are two characters and of the form <upper_case><lower_case> such as ‘Ar’, <lower_case><upper_case> as ‘aR’ or <upper or lower letter><digit> as ‘C1’. And adding to the muddle, troff(1) has its own internal registers all of which are either two lower case characters or a dot plus a letter or meta-character character. In one of the introduction examples, it was shown how to prevent the interpretation of a macro name with the escape sequence ‘\&’. This is sufficient for the internal register names also.

If a non-escaped register name is given in the argument list of a request unpredictable behavior will occur. In general, any time huge portions of text do not appear where expected in the output, or small strings such as list tags disappear, chances are there is a misunderstanding about an argument type in the argument list. Your mother never intended for you to remember this evil stuff - so here is a way to find out whether or not your arguments are valid: The .Db (debug) macro displays the interpretation of the argument list for most macros. Macros such as the .Pp (paragraph) macro do not contain debugging information. All of the callable macros do, and it is strongly advised whenever in doubt, turn on the .Db macro.

```
Usage: .Db [on | off]
```

An example of a portion of text with the debug macro placed above and below an artificially created problem (a flag argument ‘aC’ which should be \&aC in order to work):

```
.Db on
.Op Fl aC Ar file )
.Db off
```

The resulting output:

```
DEBUGGING ON
DEBUG(argv) MACRO: ‘.Op’  Line #: 2
       Argc: 1  Argv: ‘Fl’  Length: 2
       Space: ‘’  Class: Executable
       Argc: 2  Argv: ‘aC’  Length: 2
       Space: ‘’  Class: Executable
       Argc: 3  Argv: ‘Ar’  Length: 2
       Space: ‘’  Class: Executable
       Argc: 4  Argv: ‘file’  Length: 4
       Space: ‘ ’  Class: String
       Argc: 5  Argv: ‘)’  Length: 1
       Space: ‘ ’  Class: Closing Punctuation or suffix
       MACRO REQUEST: .Op Fl aC Ar file )
DEBUGGING OFF
```

The first line of information tells the name of the calling macro, here .Op, and the line number it appears on. If one or more files are involved (especially if text from another file is included) the line number may be bogus. If there is only one file, it should be accurate. The second line gives the argument count, the argument (Fl) and its length. If the length of an argument is two characters, the argument is tested to see if it is

executable (unfortunately, any register which contains a non-zero value appears executable). The third line gives the space allotted for a class, and the class type. The problem here is the argument ‘aC’ should not be executable. The four types of classes are string, executable, closing punctuation and opening punctuation. The last line shows the entire argument list as it was read. In this next example, the offending ‘aC’ is escaped:

```
.Db on
.Em An escaped \&aC
.Db off
```

```
DEBUGGING ON
DEBUG(fargv) MACRO: ‘.Em’  Line #: 2
       Argc: 1  Argv: ‘An’  Length: 2
       Space: ‘ ’  Class: String
       Argc: 2  Argv: ‘escaped’  Length: 7
       Space: ‘ ’  Class: String
       Argc: 3  Argv: ‘aC’  Length: 2
       Space: ‘ ’  Class: String
       MACRO REQUEST: .Em An escaped &aC
DEBUGGING OFF
```

The argument \&aC shows up with the same length of 2 as the ‘\&’ sequence produces a zero width, but a register named \&aC was not found and the type classified as string.

Other diagnostics consist of usage statements and are self explanatory.

GROFF, TROFF AND NROFF

The **−mdoc** package does not need compatibility mode with groff(1).

The package inhibits page breaks, and the headers and footers which normally occur at those breaks with nroff(1), to make the manual more efficient for viewing on-line. At the moment, groff(1) with **−T***ascii* does eject the imaginary remainder of the page at end of file. The inhibiting of the page breaks makes nroff(1)’d files unsuitable for hardcopy. There is a register named ‘cR’ which can be set to zero in the site dependent style file /usr/src/share/tmac/doc-nroff to restore the old style behavior.

FILES

/usr/share/tmac/tmac.doc	manual macro package
/usr/share/misc/mdoc.template	template for writing a man page

SEE ALSO

man(1), troff(1), mdoc(7)

BUGS

Undesirable hyphenation on the dash of a flag argument is not yet resolved, and causes occasional mishaps in the **DESCRIPTION** section. (line break on the hyphen).

Predefined strings are not declared in documentation.

Section 3f has not been added to the header routines.

.Nm font should be changed in **NAME** section.

.Fn needs to have a check to prevent splitting up if the line length is too short. Occasionally it separates the last parenthesis, and sometimes looks ridiculous if a line is in fill mode.

The method used to prevent header and footer page breaks (other than the initial header and footer) when using `nroff`(1) occasionally places an unsightly partially filled line (blank) at the would be bottom of the page.

If the outer-most list definition doesn't have a **`-width`** argument, the `.It` elements of inner lists may not work (producing a list where each successive element 'walks' to the right).

The list and display macros to not do any keeps and certainly should be able to.

NAME

me − macros for formatting papers

SYNOPSIS

nroff −me [options] file ...
troff −me [options] file ...

DESCRIPTION

This package of *nroff* and *troff* macro definitions provides a canned formatting facility for technical papers in various formats. When producing 2-column output on a terminal, filter the output through *col*(1).

The macro requests are defined below. Many *nroff* and *troff* requests are unsafe in conjunction with this package, however, these requests may be used with impunity after the first .pp:

.bp	begin new page
.br	break output line here
.sp n	insert n spacing lines
.ls n	(line spacing) n=1 single, n=2 double space
.na	no alignment of right margin
.ce n	center next n lines
.ul n	underline next n lines
.sz +n	add n to point size

Output of the *eqn*, *neqn*, *refer*, and *tbl*(1) preprocessors for equations and tables is acceptable as input.

FILES

/usr/share/tmac/tmac.e
/usr/share/me/*

SEE ALSO

eqn(1), troff(1), refer(1), tbl(1)
−me Reference Manual, Eric P. Allman
Writing Papers with Nroff Using −me

REQUESTS

In the following list, “initialization” refers to the first .pp, .lp, .ip, .np, .sh, or .uh macro. This list is incomplete; see *The −me Reference Manual* for interesting details.

Request	Initial Value	Cause Break	Explanation
.(c	-	yes	Begin centered block
.(d	-	no	Begin delayed text
.(f	-	no	Begin footnote
.(l	-	yes	Begin list
.(q	-	yes	Begin major quote
.(x *x*	-	no	Begin indexed item in index *x*
.(z	-	no	Begin floating keep
.)c	-	yes	End centered block
.)d	-	yes	End delayed text
.)f	-	yes	End footnote
.)l	-	yes	End list
.)q	-	yes	End major quote
.)x	-	yes	End index item
.)z	-	yes	End floating keep
.++ *m H*	-	no	Define paper section. *m* defines the part of the paper, and can be **C** (chapter), **A** (appendix), **P** (preliminary, e.g., abstract, table of contents, etc.), **B** (bibliography), **RC** (chapters renumbered from page one each chapter), or **RA** (appendix renumbered from page one).
.+c *T*	-	yes	Begin chapter (or appendix, etc., as set by .++). *T* is the chapter title.
.1c	1	yes	One column format on a new page.
.2c	1	yes	Two column format.

.EN	-	yes	Space after equation produced by *eqn* or *neqn*.
.EQ *x y*	-	yes	Precede equation; break out and add space. Equation number is *y*. The optional argument *x* may be *I* to indent equation (default), *L* to left-adjust the equation, or *C* to center the equation.
.GE	-	yes	End *gremlin* picture.
.GS	-	yes	Begin *gremlin* picture.
.PE	-	yes	End *pic* picture.
.PS	-	yes	Begin *pic* picture.
.TE	-	yes	End table.
.TH	-	yes	End heading section of table.
.TS *x*	-	yes	Begin table; if *x* is *H* table has repeated heading.
.ac *A N*	-	no	Set up for ACM style output. *A* is the Author's name(s), *N* is the total number of pages. Must be given before the first initialization.
.b *x*	no	no	Print *x* in boldface; if no argument switch to boldface.
.ba +*n*	0	yes	Augments the base indent by *n.* This indent is used to set the indent on regular text (like paragraphs).
.bc	no	yes	Begin new column
.bi *x*	no	no	Print *x* in bold italics (nofill only)
.bu	-	yes	Begin bulleted paragraph
.bx *x*	no	no	Print *x* in a box (nofill only).
.ef ´*x*´*y*´*z*´	´´´´	no	Set even footer to x y z
.eh ´*x*´*y*´*z*´	´´´´	no	Set even header to x y z
.fo ´*x*´*y*´*z*´	´´´´	no	Set footer to x y z
.hx	-	no	Suppress headers and footers on next page.
.he ´*x*´*y*´*z*´	´´´´	no	Set header to x y z
.hl	-	yes	Draw a horizontal line
.i *x*	no	no	Italicize *x;* if *x* missing, italic text follows.
.ip *x y*	no	yes	Start indented paragraph, with hanging tag *x*. Indentation is *y* ens (default 5).
.lp	yes	yes	Start left-blocked paragraph.
.lo	-	no	Read in a file of local macros of the form **.***x*. Must be given before initialization.
.np	1	yes	Start numbered paragraph.
.of ´*x*´*y*´*z*´	´´´´	no	Set odd footer to x y z
.oh ´*x*´*y*´*z*´	´´´´	no	Set odd header to x y z
.pd	-	yes	Print delayed text.
.pp	no	yes	Begin paragraph. First line indented.
.r	yes	no	Roman text follows.
.re	-	no	Reset tabs to default values.
.sc	no	no	Read in a file of special characters and diacritical marks. Must be given before initialization.
.sh *n x*	-	yes	Section head follows, font automatically bold. *n* is level of section, *x* is title of section.
.sk	no	no	Leave the next page blank. Only one page is remembered ahead.
.sm *x*	-	no	Set *x* in a smaller pointsize.
.sz +*n*	10p	no	Augment the point size by *n* points.
.th	no	no	Produce the paper in thesis format. Must be given before initialization.
.tp	no	yes	Begin title page.
.u *x*	-	no	Underline argument (even in *troff*). (Nofill only).
.uh	-	yes	Like .sh but unnumbered.
.xp *x*	-	no	Print index *x.*

NAME

NLS — Native Language Support Overview

DESCRIPTION

Native Language Support (NLS) provides commands for a single worldwide operating system base. An internationalized system has no built-in assumptions or dependencies on language-specific or cultural-specific conventions such as:

- Character classifications
- Character comparison rules
- Character collation order
- Numeric and monetary formatting
- Date and time formatting
- Message-text language
- Character sets

All information pertaining to cultural conventions and language is obtained at program run time.

"Internationalization" (often abbreviated "i18n") refers to the operation by which system software is developed to support multiple cultural-specific and language-specific conventions. This is a generalization process by which the system is untied from calling only English strings or other English-specific conventions. "Localization" (often abbreviated "l10n") refers to the operations by which the user environment is customized to handle its input and output appropriate for specific language and cultural conventions. This is a specialization process, by which generic methods already implemented in an internationalized system are used in specific ways. The formal description of cultural conventions for some country, together with all associated translations targeted to the native language, is called the "locale".

NetBSD provides extensive support to programmers and system developers to enable internationalized software to be developed. NetBSD also supplies a large variety of locales for system localization.

Localization of Information

All locale information is accessible to programs at run time so that data is processed and displayed correctly for specific cultural conventions and language.

A locale is divided into categories. A category is a group of language-specific and culture-specific conventions as outlined in the list above. ISO C specifies the following six standard categories supported by NetBSD:

`LC_COLLATE`	string-collation order information
`LC_CTYPE`	character classification, case conversion, and other character attributes
`LC_MESSAGES`	the format for affirmative and negative responses
`LC_MONETARY`	rules and symbols for formatting monetary numeric information
`LC_NUMERIC`	rules and symbols for formatting nonmonetary numeric information
`LC_TIME`	rules and symbols for formatting time and date information

Localization of the system is achieved by setting appropriate values in environment variables to identify which locale should be used. The environment variables have the same names as their respective locale categories. Additionally, the `LANG`, `LC_ALL`, and `NLSPATH` environment variables are used. The `NLSPATH` environment variable specifies a colon-separated list of directory names where the message catalog files of the NLS database are located. The `LC_ALL` and `LANG` environment variables also determine the current locale.

The values of these environment variables contains a string format as:

```
language[_territory][.codeset][@modifier]
```

Valid values for the language field come from the ISO639 standard which defines two-character codes for many languages. Some common language codes are:

Language Name	*Code*	*Language Family*
ABKHAZIAN	AB	IBERO-CAUCASIAN
AFAN (OROMO)	OM	HAMITIC
AFAR	AA	HAMITIC
AFRIKAANS	AF	GERMANIC
ALBANIAN	SQ	INDO-EUROPEAN (OTHER)
AMHARIC	AM	SEMITIC
ARABIC	AR	SEMITIC
ARMENIAN	HY	INDO-EUROPEAN (OTHER)
ASSAMESE	AS	INDIAN
AYMARA	AY	AMERINDIAN
AZERBAIJANI	AZ	TURKIC/ALTAIC
BASHKIR	BA	TURKIC/ALTAIC
BASQUE	EU	BASQUE
BENGALI	BN	INDIAN
BHUTANI	DZ	ASIAN
BIHARI	BH	INDIAN
BISLAMA	BI	
BRETON	BR	CELTIC
BULGARIAN	BG	SLAVIC
BURMESE	MY	ASIAN
BYELORUSSIAN	BE	SLAVIC
CAMBODIAN	KM	ASIAN
CATALAN	CA	ROMANCE
CHINESE	ZH	ASIAN
CORSICAN	CO	ROMANCE
CROATIAN	HR	SLAVIC
CZECH	CS	SLAVIC
DANISH	DA	GERMANIC
DUTCH	NL	GERMANIC
ENGLISH	EN	GERMANIC
ESPERANTO	EO	INTERNATIONAL AUX.
ESTONIAN	ET	FINNO-UGRIC
FAROESE	FO	GERMANIC
FIJI	FJ	OCEANIC/INDONESIAN
FINNISH	FI	FINNO-UGRIC
FRENCH	FR	ROMANCE
FRISIAN	FY	GERMANIC
GALICIAN	GL	ROMANCE
GEORGIAN	KA	IBERO-CAUCASIAN
GERMAN	DE	GERMANIC
GREEK	EL	LATIN/GREEK
GREENLANDIC	KL	ESKIMO
GUARANI	GN	AMERINDIAN
GUJARATI	GU	INDIAN
HAUSA	HA	NEGRO-AFRICAN
HEBREW	HE	SEMITIC
HINDI	HI	INDIAN
HUNGARIAN	HU	FINNO-UGRIC

ICELANDIC	IS	GERMANIC
INDONESIAN	ID	OCEANIC/INDONESIAN
INTERLINGUA	IA	INTERNATIONAL AUX.
INTERLINGUE	IE	INTERNATIONAL AUX.
INUKTITUT	IU	
INUPIAK	IK	ESKIMO
IRISH	GA	CELTIC
ITALIAN	IT	ROMANCE
JAPANESE	JA	ASIAN
JAVANESE	JV	OCEANIC/INDONESIAN
KANNADA	KN	DRAVIDIAN
KASHMIRI	KS	INDIAN
KAZAKH	KK	TURKIC/ALTAIC
KINYARWANDA	RW	NEGRO-AFRICAN
KIRGHIZ	KY	TURKIC/ALTAIC
KURUNDI	RN	NEGRO-AFRICAN
KOREAN	KO	ASIAN
KURDISH	KU	IRANIAN
LAOTHIAN	LO	ASIAN
LATIN	LA	LATIN/GREEK
LATVIAN	LV	BALTIC
LINGALA	LN	NEGRO-AFRICAN
LITHUANIAN	LT	BALTIC
MACEDONIAN	MK	SLAVIC
MALAGASY	MG	OCEANIC/INDONESIAN
MALAY	MS	OCEANIC/INDONESIAN
MALAYALAM	ML	DRAVIDIAN
MALTESE	MT	SEMITIC
MAORI	MI	OCEANIC/INDONESIAN
MARATHI	MR	INDIAN
MOLDAVIAN	MO	ROMANCE
MONGOLIAN	MN	
NAURU	NA	
NEPALI	NE	INDIAN
NORWEGIAN	NO	GERMANIC
OCCITAN	OC	ROMANCE
ORIYA	OR	INDIAN
PASHTO	PS	IRANIAN
PERSIAN (farsi)	FA	IRANIAN
POLISH	PL	SLAVIC
PORTUGUESE	PT	ROMANCE
PUNJABI	PA	INDIAN
QUECHUA	QU	AMERINDIAN
RHAETO-ROMANCE	RM	ROMANCE
ROMANIAN	RO	ROMANCE
RUSSIAN	RU	SLAVIC
SAMOAN	SM	OCEANIC/INDONESIAN
SANGHO	SG	NEGRO-AFRICAN
SANSKRIT	SA	INDIAN
SCOTS GAELIC	GD	CELTIC
SERBIAN	SR	SLAVIC
SERBO-CROATIAN	SH	SLAVIC

SESOTHO	ST	NEGRO-AFRICAN
SETSWANA	TN	NEGRO-AFRICAN
SHONA	SN	NEGRO-AFRICAN
SINDHI	SD	INDIAN
SINGHALESE	SI	INDIAN
SISWATI	SS	NEGRO-AFRICAN
SLOVAK	SK	SLAVIC
SLOVENIAN	SL	SLAVIC
SOMALI	SO	HAMITIC
SPANISH	ES	ROMANCE
SUNDANESE	SU	OCEANIC/INDONESIAN
SWAHILI	SW	NEGRO-AFRICAN
SWEDISH	SV	GERMANIC
TAGALOG	TL	OCEANIC/INDONESIAN
TAJIK	TG	IRANIAN
TAMIL	TA	DRAVIDIAN
TATAR	TT	TURKIC/ALTAIC
TELUGU	TE	DRAVIDIAN
THAI	TH	ASIAN
TIBETAN	BO	ASIAN
TIGRINYA	TI	SEMITIC
TONGA	TO	OCEANIC/INDONESIAN
TSONGA	TS	NEGRO-AFRICAN
TURKISH	TR	TURKIC/ALTAIC
TURKMEN	TK	TURKIC/ALTAIC
TWI	TW	NEGRO-AFRICAN
UIGUR	UG	
UKRAINIAN	UK	SLAVIC
URDU	UR	INDIAN
UZBEK	UZ	TURKIC/ALTAIC
VIETNAMESE	VI	ASIAN
VOLAPUK	VO	INTERNATIONAL AUX.
WELSH	CY	CELTIC
WOLOF	WO	NEGRO-AFRICAN
XHOSA	XH	NEGRO-AFRICAN
YIDDISH	YI	GERMANIC
YORUBA	YO	NEGRO-AFRICAN
ZHUANG	ZA	
ZULU	ZU	NEGRO-AFRICAN

For example, the locale for the Danish language spoken in Denmark using the ISO 8859-1 character set is da_DK.ISO8859-1. The da stands for the Danish language and the DK stands for Denmark. The short form of da_DK is sufficient to indicate this locale.

The environment variable settings are queried by their priority level in the following manner:

- If the `LC_ALL` environment variable is set, all six categories use the locale it specifies.
- If the `LC_ALL` environment variable is not set, each individual category uses the locale specified by its corresponding environment variable.
- If the `LC_ALL` environment variable is not set, and a value for a particular `LC_*` environment variable is not set, the value of the `LANG` environment variable specifies the default locale for all categories. Only the `LANG` environment variable should be set in /etc/profile, since it makes it most easy for the user to

override the system default using the individual LC_* variables.

- If the LC_ALL environment variable is not set, a value for a particular LC_* environment variable is not set, and the value of the LANG environment variable is not set, the locale for that specific category defaults to the C locale. The C or POSIX locale assumes the ASCII character set and defines information for the six categories.

Character Sets

A character is any symbol used for the organization, control, or representation of data. A group of such symbols used to describe a particular language make up a character set. It is the encoding values in a character set that provide the interface between the system and its input and output devices.

The following character sets are supported in NetBSD:

ASCII — The American Standard Code for Information Exchange (ASCII) standard specifies 128 Roman characters and control codes, encoded in a 7-bit character encoding scheme.

ISO 8859 family — Industry-standard character sets specified by the ISO/IEC 8859 standard. The standard is divided into 15 numbered parts, with each part specifying broad script similarities. Examples include Western European, Central European, Arabic, Cyrillic, Hebrew, Greek, and Turkish. The character sets use an 8-bit character encoding scheme which is compatible with the ASCII character set.

Unicode — The Unicode character set is the full set of known abstract characters of all real-world scripts. It can be used in environments where multiple scripts must be processed simultaneously. Unicode is compatible with ISO 8859-1 (Western European) and ASCII. Many character encoding schemes are available for Unicode, including UTF-8, UTF-16 and UTF-32. These encoding schemes are multi-byte encodings. The UTF-8 encoding scheme uses 8-bit, variable-width encodings which is compatible with ASCII. The UTF-16 encoding scheme uses 16-bit, variable-width encodings. The UTF-32 encoding scheme using 32-bit, fixed-width encodings.

Font Sets

A font set contains the glyphs to be displayed on the screen for a corresponding character in a character set. A display must support a suitable font to display a character set. If suitable fonts are available to the X server, then X clients can include support for different character sets. xterm(1) includes support for Unicode with UTF-8 encoding. xfd(1) is useful for displaying all the characters in an X font.

The NetBSD wscons(4) console provides support for loading fonts using the wsfontload(8) utility. Currently, only fonts for the ISO8859-1 family of character sets are supported.

Internationalization for Programmers

To facilitate translations of messages into various languages and to make the translated messages available to the program based on a user's locale, it is necessary to keep messages separate from the programs and provide them in the form of message catalogs that a program can access at run time.

Access to locale information is provided through the setlocale(3) and nl_langinfo(3) interfaces. See their respective man pages for further information.

Message source files containing application messages are created by the programmer and converted to message catalogs. These catalogs are used by the application to retrieve and display messages, as needed.

NetBSD supports two message catalog interfaces: the X/Open catgets(3) interface and the Uniforum gettext(3) interface. The catgets(3) interface has the advantage that it belongs to a standard which is well supported. Unfortunately the interface is complicated to use and maintenance of the catalogs is difficult.

The implementation also doesn't support different character sets. The gettext(3) interface has not been standardized yet, however it is being supported by an increasing number of systems. It also provides many additional tools which make programming and catalog maintenance much easier.

Support for Multi-byte Encodings

Some character sets with multi-byte encodings may be difficult to decode, or may contain state (i.e., adjacent characters are dependent). ISO C specifies a set of functions using 'wide characters' which can handle multi-byte encodings properly. The behaviour of these functions is affected by the LC_CTYPE category of the current locale.

A wide character is specified in ISO C as being a fixed number of bits wide and is stateless. There are two types for wide characters: *wchar_t* and *wint_t*. *wchar_t* is a type which can contain one wide character and operates like 'char' type does for one character. *wint_t* can contain one wide character or WEOF (wide EOF).

There are functions that operate on *wchar_t*, and substitute for functions operating on 'char'. See wmemchr(3) and towlower(3) for details. There are some additional functions that operate on *wchar_t*. See wctype(3) and wctrans(3) for details.

Wide characters should be used for all I/O processing which may rely on locale-specific strings. The two primary issues requiring special use of wide characters are:

- All I/O is performed using multibyte characters. Input data is converted into wide characters immediately after reading and data for output is converted from wide characters to multi-byte encoding immediately before writing. Conversion is controlled by the mbstowcs(3), mbsrtowcs(3), wcstombs(3), wcsrtombs(3), mblen(3), mbrlen(3), and mbsinit(3).
- Wide characters are used directly for I/O, using getwchar(3), fgetwc(3), getwc(3), ungetwc(3), fgetws(3), putwchar(3), fputwc(3), putwc(3), and fputws(3). They are also used for formatted I/O functions for wide characters such as fwscanf(3), wscanf(3), swscanf(3), fwprintf(3), wprintf(3), swprintf(3), vfwprintf(3), vwprintf(3), and vswprintf(3), and wide character identifier of %lc, %C, %ls, %S for conventional formatted I/O functions.

SEE ALSO

gencat(1), xfd(1), xterm(1), catgets(3), gettext(3), nl_langinfo(3), setlocale(3), wsfontload(8)

BUGS

This man page is incomplete.

NAME

operator — C operator precedence and associativity

DESCRIPTION

Operator	Associativity
() [] -> .	left to right
! ~ ++ -- - (type) * & sizeof	right to left
* / %	left to right
+ -	left to right
<< >>	left to right
< ≤ > ≥	left to right
== !=	left to right
&	left to right
^	left to right
|	left to right
&&	left to right
||	left to right
?:	right to left
= += -= etc.	right to left
,	left to right

FILES

`/usr/share/misc/operator`

NAME

pkgsrc — NetBSD packages collection (framework for third-party software)

DESCRIPTION

The NetBSD Packages Collection (pkgsrc) is a framework for building and maintaining third-party software on NetBSD and other UNIX-like systems. It is used to enable freely available software to be configured and built easily on supported platforms.

Tools are available to install ready-to-use packages and to perform various administrative tasks for the package system.

SEE ALSO

pkg_add(1), pkg_delete(1), pkg_info(1), http://www.netbsd.org/docs/pkgsrc/

NAME

pkgsrc — NetBSD packages collection (framework for third-party software)

DESCRIPTION

The NetBSD Packages Collection (pkgsrc) is a framework for building and maintaining third-party software on NetBSD and other UNIX-like systems. It is used to enable freely available software to be configured and built easily on supported platforms.

Tools are available to install ready-to-use packages and to perform various administrative tasks for the package system.

SEE ALSO

pkg_add(1), pkg_delete(1), pkg_info(1), http://www.NetBSD.org/docs/pkgsrc/

NAME

re_format − POSIX 1003.2 regular expressions

DESCRIPTION

Regular expressions (‘‘RE’’s), as defined in POSIX 1003.2, come in two forms: modern REs (roughly those of *egrep*; 1003.2 calls these ‘‘extended’’ REs) and obsolete REs (roughly those of *ed*; 1003.2 ‘‘basic’’ REs). Obsolete REs mostly exist for backward compatibility in some old programs; they will be discussed at the end. 1003.2 leaves some aspects of RE syntax and semantics open; ‘†’ marks decisions on these aspects that may not be fully portable to other 1003.2 implementations.

A (modern) RE is one† or more non-empty† *branches*, separated by ‘|’. It matches anything that matches one of the branches.

A branch is one† or more *pieces*, concatenated. It matches a match for the first, followed by a match for the second, etc.

A piece is an *atom* possibly followed by a single† ‘*’, ‘+’, ‘?’, or *bound*. An atom followed by ‘*’ matches a sequence of 0 or more matches of the atom. An atom followed by ‘+’ matches a sequence of 1 or more matches of the atom. An atom followed by ‘?’ matches a sequence of 0 or 1 matches of the atom.

A *bound* is ‘{’ followed by an unsigned decimal integer, possibly followed by ‘,’ possibly followed by another unsigned decimal integer, always followed by ‘}’. The integers must lie between 0 and RE_DUP_MAX (255†) inclusive, and if there are two of them, the first may not exceed the second. An atom followed by a bound containing one integer *i* and no comma matches a sequence of exactly *i* matches of the atom. An atom followed by a bound containing one integer *i* and a comma matches a sequence of *i* or more matches of the atom. An atom followed by a bound containing two integers *i* and *j* matches a sequence of *i* through *j* (inclusive) matches of the atom.

An atom is a regular expression enclosed in ‘()’ (matching a match for the regular expression), an empty set of ‘()’ (matching the null string)†, a *bracket expression* (see below), ‘.’ (matching any single character), ‘^’ (matching the null string at the beginning of a line), ‘$’ (matching the null string at the end of a line), a ‘\’ followed by one of the characters ‘^.[$()|*+?{\’ (matching that character taken as an ordinary character), a ‘\’ followed by any other character† (matching that character taken as an ordinary character, as if the ‘\’ had not been present†), or a single character with no other significance (matching that character). A ‘{’ followed by a character other than a digit is an ordinary character, not the beginning of a bound†. It is illegal to end an RE with ‘\’.

A *bracket expression* is a list of characters enclosed in ‘[]’. It normally matches any single character from the list (but see below). If the list begins with ‘^’, it matches any single character (but see below) *not* from the rest of the list. If two characters in the list are separated by ‘−’, this is shorthand for the full *range* of characters between those two (inclusive) in the collating sequence, e.g. ‘[0-9]’ in ASCII matches any decimal digit. It is illegal† for two ranges to share an endpoint, e.g. ‘a-c-e’. Ranges are very collating-sequence-dependent, and portable programs should avoid relying on them.

To include a literal ‘]’ in the list, make it the first character (following a possible ‘^’). To include a literal ‘−’, make it the first or last character, or the second endpoint of a range. To use a literal ‘−’ as the first endpoint of a range, enclose it in ‘[.’ and ‘.]’ to make it a collating element (see below). With the exception of these and some combinations using ‘[’ (see next paragraphs), all other special characters, including ‘\’, lose their special significance within a bracket expression.

Within a bracket expression, a collating element (a character, a multi-character sequence that collates as if it were a single character, or a collating-sequence name for either) enclosed in ‘[.’ and ‘.]’ stands for the sequence of characters of that collating element. The sequence is a single element of the bracket expression’s list. A bracket expression containing a multi-character collating element can thus match more than one character, e.g. if the collating sequence includes a ‘ch’ collating element, then the RE ‘[[.ch.]]*c’ matches the first five characters of ‘chchcc’.

Within a bracket expression, a collating element enclosed in ‘[=’ and ‘=]’ is an equivalence class, standing for the sequences of characters of all collating elements equivalent to that one, including itself. (If there are no other equivalent collating elements, the treatment is as if the enclosing delimiters were ‘[.’ and ‘.]’.) For

example, if o and ô are the members of an equivalence class, then '[[=o=]]', '[[=ô=]]', and '[oô]' are all synonymous. An equivalence class may not† be an endpoint of a range.

Within a bracket expression, the name of a *character class* enclosed in '[:' and ':]' stands for the list of all characters belonging to that class. Standard character class names are:

alnum	digit	punct
alpha	graph	space
blank	lower	upper
cntrl	print	xdigit

These stand for the character classes defined in *ctype*(3). A locale may provide others. A character class may not be used as an endpoint of a range.

There are two special cases† of bracket expressions: the bracket expressions '[[:<:]]' and '[[:>:]]' match the null string at the beginning and end of a word respectively. A word is defined as a sequence of word characters which is neither preceded nor followed by word characters. A word character is an *alnum* character (as defined by *ctype*(3)) or an underscore. This is an extension, compatible with but not specified by POSIX 1003.2, and should be used with caution in software intended to be portable to other systems.

In the event that an RE could match more than one substring of a given string, the RE matches the one starting earliest in the string. If the RE could match more than one substring starting at that point, it matches the longest. Subexpressions also match the longest possible substrings, subject to the constraint that the whole match be as long as possible, with subexpressions starting earlier in the RE taking priority over ones starting later. Note that higher-level subexpressions thus take priority over their lower-level component subexpressions.

Match lengths are measured in characters, not collating elements. A null string is considered longer than no match at all. For example, 'bb*' matches the three middle characters of 'abbbc', '(wee|week)(knights|nights)' matches all ten characters of 'weeknights', when '(.*).*' is matched against 'abc' the parenthesized subexpression matches all three characters, and when '(a*)*' is matched against 'bc' both the whole RE and the parenthesized subexpression match the null string.

If case-independent matching is specified, the effect is much as if all case distinctions had vanished from the alphabet. When an alphabetic that exists in multiple cases appears as an ordinary character outside a bracket expression, it is effectively transformed into a bracket expression containing both cases, e.g. 'x' becomes '[xX]'. When it appears inside a bracket expression, all case counterparts of it are added to the bracket expression, so that (e.g.) '[x]' becomes '[xX]' and '[^x]' becomes '[^xX]'.

No particular limit is imposed on the length of REs†. Programs intended to be portable should not employ REs longer than 256 bytes, as an implementation can refuse to accept such REs and remain POSIX-compliant.

Obsolete (''basic'') regular expressions differ in several respects. '|', '+', and '?' are ordinary characters and there is no equivalent for their functionality. The delimiters for bounds are '\{' and '\}', with '{' and '}' by themselves ordinary characters. The parentheses for nested subexpressions are '\(' and '\)', with '(' and ')' by themselves ordinary characters. '^' is an ordinary character except at the beginning of the RE or† the beginning of a parenthesized subexpression, '$' is an ordinary character except at the end of the RE or† the end of a parenthesized subexpression, and '*' is an ordinary character if it appears at the beginning of the RE or the beginning of a parenthesized subexpression (after a possible leading '^'). Finally, there is one new type of atom, a *back reference*: '\' followed by a non-zero decimal digit *d* matches the same sequence of characters matched by the *d*th parenthesized subexpression (numbering subexpressions by the positions of their opening parentheses, left to right), so that (e.g.) '\([bc]\)\1' matches 'bb' or 'cc' but not 'bc'.

SEE ALSO

regex(3)

POSIX 1003.2, section 2.8 (Regular Expression Notation).

BUGS

Having two kinds of REs is a botch.

The current 1003.2 spec says that ‘)’ is an ordinary character in the absence of an unmatched ‘(’; this was an unintentional result of a wording error, and change is likely. Avoid relying on it.

Back references are a dreadful botch, posing major problems for efficient implementations. They are also somewhat vaguely defined (does ‘a\(\(b\)*\2\)*d’ match ‘abbbd’?). Avoid using them.

1003.2’s specification of case-independent matching is vague. The ‘‘one case implies all cases’’ definition given above is current consensus among implementors as to the right interpretation.

The syntax for word boundaries is incredibly ugly.

NAME

re_format − POSIX 1003.2 regular expressions

DESCRIPTION

Regular expressions (‘‘RE’’s), as defined in POSIX 1003.2, come in two forms: modern REs (roughly those of *egrep*; 1003.2 calls these ‘‘extended’’ REs) and obsolete REs (roughly those of *ed*; 1003.2 ‘‘basic’’ REs). Obsolete REs mostly exist for backward compatibility in some old programs; they will be discussed at the end. 1003.2 leaves some aspects of RE syntax and semantics open; ‘†’ marks decisions on these aspects that may not be fully portable to other 1003.2 implementations.

A (modern) RE is one† or more non-empty† *branches*, separated by ‘|’. It matches anything that matches one of the branches.

A branch is one† or more *pieces*, concatenated. It matches a match for the first, followed by a match for the second, etc.

A piece is an *atom* possibly followed by a single† ‘*’, ‘+’, ‘?’, or *bound*. An atom followed by ‘*’ matches a sequence of 0 or more matches of the atom. An atom followed by ‘+’ matches a sequence of 1 or more matches of the atom. An atom followed by ‘?’ matches a sequence of 0 or 1 matches of the atom.

A *bound* is ‘{’ followed by an unsigned decimal integer, possibly followed by ‘,’ possibly followed by another unsigned decimal integer, always followed by ‘}’. The integers must lie between 0 and RE_DUP_MAX (255†) inclusive, and if there are two of them, the first may not exceed the second. An atom followed by a bound containing one integer *i* and no comma matches a sequence of exactly *i* matches of the atom. An atom followed by a bound containing one integer *i* and a comma matches a sequence of *i* or more matches of the atom. An atom followed by a bound containing two integers *i* and *j* matches a sequence of *i* through *j* (inclusive) matches of the atom.

An atom is a regular expression enclosed in ‘()’ (matching a match for the regular expression), an empty set of ‘()’ (matching the null string)†, a *bracket expression* (see below), ‘.’ (matching any single character), ‘^’ (matching the null string at the beginning of a line), ‘$’ (matching the null string at the end of a line), a ‘\’ followed by one of the characters ‘^.[$()|*+?{\’ (matching that character taken as an ordinary character), a ‘\’ followed by any other character† (matching that character taken as an ordinary character, as if the ‘\’ had not been present†), or a single character with no other significance (matching that character). A ‘{’ followed by a character other than a digit is an ordinary character, not the beginning of a bound†. It is illegal to end an RE with ‘\’.

A *bracket expression* is a list of characters enclosed in ‘[]’. It normally matches any single character from the list (but see below). If the list begins with ‘^’, it matches any single character (but see below) *not* from the rest of the list. If two characters in the list are separated by ‘−’, this is shorthand for the full *range* of characters between those two (inclusive) in the collating sequence, e.g. ‘[0-9]’ in ASCII matches any decimal digit. It is illegal† for two ranges to share an endpoint, e.g. ‘a-c-e’. Ranges are very collating-sequence-dependent, and portable programs should avoid relying on them.

To include a literal ‘]’ in the list, make it the first character (following a possible ‘^’). To include a literal ‘−’, make it the first or last character, or the second endpoint of a range. To use a literal ‘−’ as the first endpoint of a range, enclose it in ‘[.’ and ‘.]’ to make it a collating element (see below). With the exception of these and some combinations using ‘[’ (see next paragraphs), all other special characters, including ‘\’, lose their special significance within a bracket expression.

Within a bracket expression, a collating element (a character, a multi-character sequence that collates as if it were a single character, or a collating-sequence name for either) enclosed in ‘[.’ and ‘.]’ stands for the sequence of characters of that collating element. The sequence is a single element of the bracket expression’s list. A bracket expression containing a multi-character collating element can thus match more than one character, e.g. if the collating sequence includes a ‘ch’ collating element, then the RE ‘[[.ch.]]*c’ matches the first five characters of ‘chchcc’.

Within a bracket expression, a collating element enclosed in ‘[=’ and ‘=]’ is an equivalence class, standing for the sequences of characters of all collating elements equivalent to that one, including itself. (If there are no other equivalent collating elements, the treatment is as if the enclosing delimiters were ‘[.’ and ‘.]’.) For

example, if o and ô are the members of an equivalence class, then '[[=o=]]', '[[=ô=]]', and '[oô]' are all synonymous. An equivalence class may not† be an endpoint of a range.

Within a bracket expression, the name of a *character class* enclosed in '[:' and ':]' stands for the list of all characters belonging to that class. Standard character class names are:

alnum	digit	punct
alpha	graph	space
blank	lower	upper
cntrl	print	xdigit

These stand for the character classes defined in *ctype*(3). A locale may provide others. A character class may not be used as an endpoint of a range.

There are two special cases† of bracket expressions: the bracket expressions '[[:<:]]' and '[[:>:]]' match the null string at the beginning and end of a word respectively. A word is defined as a sequence of word characters which is neither preceded nor followed by word characters. A word character is an *alnum* character (as defined by *ctype*(3)) or an underscore. This is an extension, compatible with but not specified by POSIX 1003.2, and should be used with caution in software intended to be portable to other systems.

In the event that an RE could match more than one substring of a given string, the RE matches the one starting earliest in the string. If the RE could match more than one substring starting at that point, it matches the longest. Subexpressions also match the longest possible substrings, subject to the constraint that the whole match be as long as possible, with subexpressions starting earlier in the RE taking priority over ones starting later. Note that higher-level subexpressions thus take priority over their lower-level component subexpressions.

Match lengths are measured in characters, not collating elements. A null string is considered longer than no match at all. For example, 'bb*' matches the three middle characters of 'abbbc', '(wee|week)(knights|nights)' matches all ten characters of 'weeknights', when '(.*).*' is matched against 'abc' the parenthesized subexpression matches all three characters, and when '(a*)*' is matched against 'bc' both the whole RE and the parenthesized subexpression match the null string.

If case-independent matching is specified, the effect is much as if all case distinctions had vanished from the alphabet. When an alphabetic that exists in multiple cases appears as an ordinary character outside a bracket expression, it is effectively transformed into a bracket expression containing both cases, e.g. 'x' becomes '[xX]'. When it appears inside a bracket expression, all case counterparts of it are added to the bracket expression, so that (e.g.) '[x]' becomes '[xX]' and '[^x]' becomes '[^xX]'.

No particular limit is imposed on the length of REs†. Programs intended to be portable should not employ REs longer than 256 bytes, as an implementation can refuse to accept such REs and remain POSIX-compliant.

Obsolete (''basic'') regular expressions differ in several respects. '|', '+', and '?' are ordinary characters and there is no equivalent for their functionality. The delimiters for bounds are '\{' and '\}', with '{' and '}' by themselves ordinary characters. The parentheses for nested subexpressions are '\(' and '\)', with '(' and ')' by themselves ordinary characters. '^' is an ordinary character except at the beginning of the RE or† the beginning of a parenthesized subexpression, '$' is an ordinary character except at the end of the RE or† the end of a parenthesized subexpression, and '*' is an ordinary character if it appears at the beginning of the RE or the beginning of a parenthesized subexpression (after a possible leading '^'). Finally, there is one new type of atom, a *back reference*: '\' followed by a non-zero decimal digit *d* matches the same sequence of characters matched by the *d*th parenthesized subexpression (numbering subexpressions by the positions of their opening parentheses, left to right), so that (e.g.) '\([bc]\)\1' matches 'bb' or 'cc' but not 'bc'.

SEE ALSO

regex(3)

POSIX 1003.2, section 2.8 (Regular Expression Notation).

BUGS

Having two kinds of REs is a botch.

The current 1003.2 spec says that ‘)’ is an ordinary character in the absence of an unmatched ‘(’; this was an unintentional result of a wording error, and change is likely. Avoid relying on it.

Back references are a dreadful botch, posing major problems for efficient implementations. They are also somewhat vaguely defined (does ‘a\(\(b\)*\2\)*d’ match ‘abbbd’?). Avoid using them.

1003.2’s specification of case-independent matching is vague. The ‘‘one case implies all cases’’ definition given above is current consensus among implementors as to the right interpretation.

The syntax for word boundaries is incredibly ugly.

NAME

release — layout of NetBSD releases and snapshots

DESCRIPTION

This document describes the layout of NetBSD releases and snapshots. This layout should be consistent between FTP servers and CD-ROMs, except possibly the path that leads to the release hierarchy.

In this document, the following special words have these definitions:

<*machine*> The platform for which the release was built, corresponding to the *hw.machine* sysctl variable, e.g. *i386* or *amiga*.

<*machine_arch*> The architecture for which a particular installation set was built, corresponding to the *hw.machine_arch* sysctl variable, e.g. *i386* or *m68k*.

<*rel*> The target release.

All **README** files are descriptions of the various files in directories that have "non-standard" contents. There may also be a **README** file at the top-level, describing who built the snapshot and under what circumstances (e.g. whether it's an official NetBSD snapshot, or not).

All **BSDSUM** files are historic BSD checksums for the various files in that directory, in the format produced by the command: **cksum -o 1 <file>**.

All **CKSUM** files are POSIX checksums for the various files in that directory, in the format produced by the command: **cksum <file>**.

All **MD5** files are MD5 digests for the various files in that directory, in the format produced by the command: **cksum -m <file>**.

All **SYSVSUM** files are historic AT&T System V UNIX checksums for the various files in that directory, in the format produced by the command: **cksum -o 2 <file>**.

The MD5 digest is the safest checksum, followed by the POSIX checksum. The other two checksums are provided only to ensure that the widest possible range of system can check the integrity of the release files.

Files that end in **.tgz** are gzipped tar archives. This is used in lieu of **.tar.gz** because the software used to download the sets may incorrectly auto-unpack files ending in **.gz** and to accommodate systems which only support 3 character extensions to file names.

All tar archives are relative to the target's **/** directory, and *do not* include the leading "/".

All compression of release files is to be performed with the command: **gzip -9**.

The root of the release hierarchy may be the root directory of a CD-ROM, but in all other cases it should be **.../NetBSD-**<*rel*>**/**.

The root of the release hierarchy should contain the following files and subdirectories:

SOURCE_DATE
A file containing the date, in UTC, of the source code from which the release or snapshot was built, in the default format produced by the command: **date -u**.

iso/ CDROM images in ISO 9660 format, usually created with "./build.sh ... iso-image ..." after a "./build.sh -x ... release ..." in `src` or created with "./build.sh ... iso-image-source ..." after a "./build.sh -x ... release sourcesets ..." in `src`.

Images in this directory, unlike images in the **.../NetBSD-**<*rel*>**/**<*machine*>**/installation/cdrom/** directory, should contain file systems that have an internal layout that corresponds to a complete release for one or more machine types. If built with "iso-image-source", then it will also contain a "source" directory. These images

are usually bootable.

BSDSUM

CKSUM

MD5

README

SYSVSUM

<*machine_arch*>**cd.iso**

shared/ Files shared by two or more machine types.

<*machine_arch*>**/** Files which may be shared by all systems of the same <*machine_arch*> will be located in **.../NetBSD-**<*rel*>**/shared/**<*machine_arch*>**/** with symbolic links pointing to these files from the <*machine*> subdirectory.

ALL/ Files which are completely machine-independent will be located in **.../NetBSD-**<*rel*>**/shared/ALL/** with symbolic links pointing to these files from the <*machine*> subdirectory.

source/ Source codes of the operating system and patches for it should be put into **.../NetBSD-**<*rel*>**/source/** using the following layout:

patches/ This directory contains various patch files appropriate for `patch`(1). Other patches may exist for fixing critical problems.

BSDSUM

CKSUM

MD5

README

SYSVSUM

diff-<*lastrel*>**-**<*rel*>**.gz** Diff against the last release, usually generated by **`cvs rdiff`**. For patch releases, diffs against the last release are included. If the last release was a patch release itself, the diff is against that patch release is included.

sets/ Sources for the various system sets, based on their modules in the CVS server.

BSDSUM

CKSUM

MD5

README

SYSVSUM

gnusrc.tgz Contains sources for all GPLed and possibly other programs that contains restrictions in their licensing that prevent others from using these programs in closed-source environments.

pkgsrc.tgz Package-sources for third party software ready to compile. See pkgsrc/README for more information.

sharesrc.tgz Contains machine-independent data files that can be shared across architectures/systems.

src.tgz The operating system's userland source code, including all programs, tools, toolchain, etc.

syssrc.tgz Kernel sources for all architectures plus sources of the tools needed to build kernels (like `config`(1)).

xsrc.tgz Source code of the X Window System used on all NetBSD architectures. Includes X clients and servers.

<*machine*>/ The binary releases in **.../NetBSD-**<*rel*>/<*machine*>/ follow the following layout:

INSTALL.txt Installation notes, including complete descriptions of files contained within the release hierarchy

INSTALL.more pretty version of this, suited for viewing with `more`(1)

INSTALL.html HTML version of this

INSTALL.ps PostScript version of this

binary/ system binaries

sets/ installation sets

BSDSUM

CKSUM

MD5

SYSVSUM

base.tgz The base binary distribution. This set contains the base NetBSD utilities that are necessary for the system to run and be minimally functional. It includes shared libraries for those architectures that support them. This set excludes all things listed in the sets described below.

comp.tgz The compiler tools distribution. This set contains the C and C++ compilers, assembler, linker, other toolchain components, and their manual pages. It also includes the system include files (`/usr/include`), and the static system libraries.

etc.tgz This set contains the system configuration files that reside in `/etc` and in several other places throughout the file system hierarchy.

games.tgz This set includes the games and their manual pages.

kern.tgz This set includes a generic kernel.

man.tgz This set includes all of the manual pages for the binaries and other software contained in the **base** set which are not included in the other sets.

misc.tgz This set includes the system dictionaries (which are rather large), the typesettable document set, and manual pages for other architectures, which happen to be installed from the source tree by default.

text.tgz This set includes the NetBSD text processing tools, including `groff`(1), all related programs, and their manual pages.

xbase.tgz This set includes the base X11 distribution, including manual pages and shared libraries for those architectures that support them, and excluding everything contained in the other X11 sets.

xcomp.tgz This set includes the X11 include files and static X11 libraries.

xcontrib.tgz This set includes binaries and manual pages for programs built from the X11 “contrib” sources.

xfont.tgz This set includes the X11 fonts.

xserver.tgz This set includes the X servers and manual pages for <machine>. *Note: this set may not be available on some platforms*.

kernel/ suitably named, gzipped kernels

BSDSUM

CKSUM

MD5

README

SYSVSUM

netbsd-GENERIC.gz A kernel built from the **GENERIC** kernel configuration file. This is meant as an example only; different platforms may have differently named kernels.

installation/ installation helper items

cdrom/ CDROM images in ISO 9660 format, usually created as part of “build.sh ... release ...” in `src`.

Images in this directory will typically be bootable, and will contain one or more of a kernel, installation tools, and rescue tools. They will not contain installation sets, source sets, or other components of a complete release.

BSDSUM

CKSUM

MD5

README

SYSVSUM

netbsd-<*machine_arch*>**.iso**

diskimage/ disk images, for those platforms that provide them

BSDSUM

CKSUM

MD5

README

SYSVSUM

diskimage-rz25.gz

floppy/ floppy images, for those platforms that provide them

BSDSUM

CKSUM

MD5

README

SYSVSUM

floppy-144.gz

miniroot/ miniroot images, for those platforms that provide them

BSDSUM

CKSUM

MD5

README

SYSVSUM

miniroot.gz

misc/ miscellaneous installation helper utilities, including boot selectors, floppy writing software, other software that runs under foreign operating systems, etc.

BSDSUM
CKSUM
MD5
README
SYSVSUM
...

netboot/ network boot programs
BSDSUM
CKSUM
MD5
README
SYSVSUM
netboot.gz

tapeimage/ tape images, for those platforms that provide them
BSDSUM
CKSUM
MD5
README
SYSVSUM
tapeimage-hp9144.gz

SEE ALSO

`cksum`(1), `date`(1), `gzip`(1), `split`(1), `tar`(1)

HISTORY

The **`release`** manual page first appeared in NetBSD 1.3.

NAME

script — interpreter script execution

DESCRIPTION

The system is capable of treating a text file containing commands intended for an interpreter, such as sh(1) or awk(1), as an executable program.

An "interpreter script" is a file which has been set executable (see chmod(2)) and which has a first line of the form:

```
#! pathname [argument]
```

The "#!" must appear as the first two characters of the file. A space between the "#!" and *pathname* is optional. At most one *argument* may follow *pathname*, and the length of the entire line is limited (see below).

If such a file is executed (such as via the execve(2) system call), the interpreter specified by the *pathname* is executed by the system. (The *pathname* is executed without regard to the PATH variable, so in general *pathname* should be an absolute path.)

The arguments passed to the interpreter will be as follows. *argv[0]* will be the path to the interpreter itself, as specified on the first line of the script. If there is an *argument* following *pathname* on the first line of the script, it will be passed as *argv[1]*. The subsequent elements of *argv* will be the path to the interpreter script file itself (i.e. the original *argv[0]*) followed by any further arguments passed when execve(2) was invoked to execute the script file.

By convention, it is expected that an interpreter will open the script file passed as an argument and process the commands within it. Typical interpreters treat '#' as a comment character, and thus will ignore the initial line of the script because it begins "#!", but there is no requirement for this per se.

On NetBSD, the length of the "#!" line, excluding the "#!" itself, is limited to PATH_MAX (as defined in ⟨limits.h⟩). Other operating systems impose much smaller limits on the length of the "#!" line (see below).

Note that the interpreter may not itself be an interpreter script. If *pathname* does not point to an executable binary, execution of the interpreter script will fail.

Trampolines and Portable Scripts

Different operating systems often have interpreters located in different locations, and the kernel executes the passed interpreter without regard to the setting of environment variables such as PATH. This makes it somewhat challenging to set the "#!" line of a script so that it will run identically on different systems.

Since the env(1) utility executes a command passed to it on its command line, it is often used as a "trampoline" to render scripts portable. If the leading line of a script reads

```
#! /usr/bin/env interp
```

then the env(1) command will execute the "interp" command it finds in its PATH, passing on to it all subsequent arguments with which it itself was called. Since /usr/bin/env is found on almost all POSIX style systems, this trick is frequently exploited by authors who need a script to execute without change on multiple systems.

Historical Note: Scripts without "#!"

Shell scripts predate the invention of the "#!" convention, which is implemented in the kernel. In the days of Version 7 AT&T UNIX, there was only one interpreter used on the system, /bin/sh, and the shell treated any file that failed to execute with an ENOEXEC error (see intro(2)) as a shell script.

Most shells (such as `sh`(1)) and certain other facilities (including `execlp`(3) and `execvp`(3) but not other types of `exec`(3) calls) still pass interpreter scripts that do not include the "#!" (and thus fail to execute with `ENOEXEC`) to `/bin/sh`.

As this behavior is implemented outside the kernel, there is no mechanism that forces it to be respected by all programs that execute other programs. It is thus not completely reliable. It is therefore important to always include

```
#!/bin/sh
```

in front of Bourne shell scripts, and to treat the traditional behavior as obsolete.

EXAMPLES

Suppose that an executable binary exists in `/bin/interp` and that the file `/tmp/script` contains:

```
#!/bin/interp -arg

[...]
```

and that `/tmp/script` is set mode 755.

Executing

```
$ /tmp/script one two three
```

at the shell will result in `/bin/interp` being executed, receiving the following arguments in *argv* (numbered from 0):

"/bin/interp", "-arg", "/tmp/script", "one", "two", "three"

Portability Note: Multiple arguments

The behavior of multiple arguments on the "#!" line is highly non-portable between different systems. In general, only one argument can be assumed to work consistently.

Consider the following variation on the previous example. Suppose that an executable binary exists in `/bin/interp` and that the file `/tmp/script` contains:

```
#!/bin/interp -x -y

[...]
```

and that `/tmp/script` is set mode 755.

Executing

```
$ /tmp/script one two three
```

at the shell will result in `/bin/interp` being executed, receiving the following arguments in *argv* (numbered from 0):

"/bin/interp", "-x -y", "/tmp/script", "one", "two", "three"

Note that "-x -y" will be passed on NetBSD as a single argument.

Although most POSIX style operating systems will pass only one *`argument`*, the behavior when multiple arguments are included is not consistent between platforms. Some, such as current releases of NetBSD, will concatenate multiple arguments into a single argument (as above), some will truncate them, and at least one will pass them as multiple arguments.

The NetBSD behavior is common but not universal. Sun's Solaris would present the above argument as "-x", dropping the " -y" entirely. Perhaps uniquely, recent versions of Apple's OS X will actually pass multiple arguments properly, i.e.:

"/bin/interp", "-x", "-y", "/tmp/script", "one", "two", "three"

The behavior of the system in the face of multiple arguments is thus not currently standardized, should not be relied on, and may be changed in future releases. In general, pass at most one argument, and do not rely on multiple arguments being concatenated.

SEE ALSO

awk(1), csh(1), ksh(1), sh(1), chmod(2), execve(2), intro(2), execlp(3), execvp(3), fd(4), options(4), setuid(7)

STANDARDS

The behavior of interpreter scripts is obliquely referred to, but never actually described in, IEEE Std 1003.1-2004 " ("POSIX.1").

The behavior is partially (but not completely) described in the System V Interface Definition, Fourth Edition ("SVID4").

Although it has never been formally standardized, the behavior described is largely portable across POSIX style systems, with two significant exceptions: the maximum length of the "#!" line, and the behavior if multiple arguments are passed. Please be aware that some operating systems limit the line to 32 or 64 characters, and that (as described above) the behavior in the face of multiple arguments is not consistent across systems.

HISTORY

The behavior of the kernel when encountering scripts that start in "#!" was not present in Version 7 AT&T UNIX. A Usenet posting to net.unix by Guy Harris on October 16, 1984 claims that the idea for the "#!" behavior was first proposed by Dennis Ritchie but that the first implementation was on BSD.

Historical manuals (specifically the exec man page) indicate that the behavior was present in 4BSD at least as early as April, 1981. Information on precisely when it was first implemented, and in which version of UNIX, is solicited.

SECURITY CONSIDERATIONS

Numerous security problems are associated with setuid interpreter scripts.

In addition to the fact that many interpreters (and scripts) are simply not designed to be robust in a setuid context, a race condition exists between the moment that the kernel examines the interpreter script file and the moment that the newly invoked interpreter opens the file itself.

Because of these security issues, NetBSD does not allow setuid interpreter scripts by default. In order to turn on setuid interpreter scripts,

options SETUIDSCRIPTS

must be set in the configuration of the running kernel. Setting this option implies the **FDSCRIPTS** option, which causes the kernel to open the script file on behalf of the interpreter and pass it in *argv* as /dev/fd/[fdnum]. (See fd(4) for an explanation of the /dev/fd/[fdnum] devices.) This design avoids the race condition, at the cost of denying the interpreter the actual name of the script file. See options(4) for more information.

However, the **FDSCRIPTS** mechanism is not a cure-all for security issues in setuid interpreters and scripts. Subtle techniques can be used to subvert even seemingly well written scripts. Scripts executed by Bourne type shells can be subverted in numerous ways, such as by setting the IFS variable before executing the script. Other interpreters possess their own vulnerabilities. Turning on **SETUIDSCRIPTS** is therefore very dangerous, and should not be done lightly if at all.

NAME

setuid — checklist for security of setuid programs

DESCRIPTION

Please note: This manual page was written long ago, and is in need of updating to match today's systems. We think it is valuable enough to include, even though parts of it are outdated. A carefully-researched updated version would be very useful, if anyone is feeling enthusiastic...

Writing a secure setuid (or setgid) program is tricky. There are a number of possible ways of subverting such a program. The most conspicuous security holes occur when a setuid program is not sufficiently careful to avoid giving away access to resources it legitimately has the use of. Most of the other attacks are basically a matter of altering the program's environment in unexpected ways and hoping it will fail in some security-breaching manner. There are generally three categories of environment manipulation: supplying a legal but unexpected environment that may cause the program to directly do something insecure, arranging for error conditions that the program may not handle correctly, and the specialized subcategory of giving the program inadequate resources in hopes that it won't respond properly.

The following are general considerations of security when writing a setuid program.

- The program should run with the weakest userid possible, preferably one used only by itself. A security hole in a setuid program running with a highly-privileged userid can compromise an entire system. Security-critical programs like `passwd`(1) should always have private userids, to minimize possible damage from penetrations elsewhere.
- The result of `getlogin`(2) or `ttyname`(3) may be wrong if the descriptors have been meddled with. There is *no* foolproof way to determine the controlling terminal or the login name (as opposed to uid) on V7.
- On some systems, the setuid bit may not be honored if the program is run by root, so the program may find itself running as root.
- Programs that attempt to use `creat`(3) for locking can foul up when run by root; use of `link`(2) is preferred when implementing locking. Using `chmod`(2) for locking is an obvious disaster.
- Breaking an existing lock is very dangerous; the breakdown of a locking protocol may be symptomatic of far worse problems. Doing so on the basis of the lock being 'old' is sometimes necessary, but programs can run for surprising lengths of time on heavily-loaded systems.
- Care must be taken that user requests for I/O are checked for permissions using the user's permissions, not the program's. Use of `access`(2) is recommended.
- Programs executed at user request (e.g. shell escapes) must not receive the setuid program's permissions; use of daughter processes and "setuid(getuid())" plus "setgid(getgid())" after `fork`(2) but before `exec`(3) is vital.
- Similarly, programs executed at user request must not receive other sensitive resources, notably file descriptors.

  Programs activated by one user but handling traffic on behalf of others (e.g. daemons) should avoid doing "setuid(getuid())" or "setgid(getgid())", since the original invoker's identity is almost certainly inappropriate. On systems which permit it, use of "setuid(geteuid())" and "setgid(getegid())" is recommended when performing work on behalf of the system as opposed to a specific user.
- There are inherent permission problems when a setuid program executes another setuid program, since the permissions are not additive. Care should be taken that created files are not owned by the wrong person. Use of "setuid(geteuid())" and its gid counterpart can help, if the system allows them.

- Care should be taken that newly-created files do not have the wrong permission or ownership even momentarily. Permissions should be arranged by using `umask`(2) in advance, rather than by creating the file wide-open and then using `chmod`(2). Ownership can get sticky due to the limitations of the setuid concept, although using a daughter process connected by a pipe can help.
- Setuid programs should be especially careful about error checking, and the normal response to a strange situation should be termination, rather than an attempt to carry on.

The following are ways in which the program may be induced to carelessly give away its special privileges.

- The directory the program is started in, or directories it may plausibly `chdir`(2) to, may contain programs with the same names as system programs, placed there in hopes that the program will activate a shell with a permissive `PATH` setting. `PATH` should *always* be standardized before invoking a shell (either directly or via `popen`(3) or `execvp`(3) or `execlp`(3)).
- Similarly, a bizarre `IFS` setting may alter the interpretation of a shell command in really strange ways, possibly causing a user-supplied program to be invoked. `IFS` too should always be standardized before invoking a shell.
- Environment variables in general cannot be trusted. Their contents should never be taken for granted.
- Setuid shell files (on systems which implement such) simply cannot cope adequately with some of these problems. They also have some nasty problems like trying to run a `.profile` when run under a suitable name. They are terminally insecure, and must be avoided.
- Relying on the contents of files placed in publically-writable directories, such as `/tmp`, is a nearly-incurable security problem. Setuid programs should avoid using `/tmp` entirely, if humanly possible. The sticky-directories modification (sticky bit on for a directory means only owner of a file can remove it) helps, but is not a complete solution.
- A related problem is that spool directories, holding information that the program will trust later, must never be publically writable even if the files in the directory are protected. Among other sinister manipulations that can be performed, note that on many Unixes, a core dump of a setuid program is owned by the program's owner and not by the user running it.

The following are unusual but possible error conditions that the program should cope with properly (resource-exhaustion questions are considered separately, see below).

- The value of *`argc`* might be 0.
- The setting of the `umask`(2) might not be sensible. In any case, it should be standardized when creating files not intended to be owned by the user.
- One or more of the standard descriptors might be closed, so that an opened file might get (say) descriptor 1, causing chaos if the program tries to do a `printf`(3).
- The current directory (or any of its parents) may be unreadable and unsearchable. On many systems `pwd`(1) does not run setuid-root, so it can fail under such conditions.
- Descriptors shared by other processes (i.e., any that are open on startup) may be manipulated in strange ways by said processes.
- The standard descriptors may refer to a terminal which has a bizarre mode setting, or which cannot be opened again, or which gives end-of-file on any read attempt, or which cannot be read or written successfully.
- The process may be hit by interrupt, quit, hangup, or broken-pipe signals, singly or in fast succession. The user may deliberately exploit the race conditions inherent in catching signals; ignoring signals is safe, but catching them is not.

- Although non-keyboard signals cannot be sent by ordinary users in V7, they may perhaps be sent by the system authorities (e.g. to indicate that the system is about to shut down), so the possibility cannot be ignored.
- On some systems there may be an `alarm`(3) signal pending on startup.
- The program may have children it did not create. This is normal when the process is part of a pipeline.
- In some non-V7 systems, users can change the ownerships of their files. Setuid programs should avoid trusting the owner identification of a file.
- User-supplied arguments and input data *must* be checked meticulously. Overly-long input stored in an array without proper bound checking can easily breach security. When software depends on a file being in a specific format, user-supplied data should never be inserted into the file without being checked first. Meticulous checking includes allowing for the possibility of non-ASCII characters.
- Temporary files left in public directories like `/tmp` might vanish at inconvenient times.

The following are resource-exhaustion possibilities that the program should respond properly to.

- The user might have used up all of his allowed processes, so any attempt to create a new one (via `fork`(2) or `popen`(3)) will fail.
- There might be many files open, exhausting the supply of descriptors.
- There might be many arguments.
- The arguments and the environment together might occupy a great deal of space.

Systems which impose other resource limitations can open setuid programs to similar resource-exhaustion attacks.

Setuid programs which execute ordinary programs without reducing authority pass all the above problems on to such unprepared children. Standardizing the execution environment is only a partial solution.

HISTORY

Written by Henry Spencer, and based on additional outside contributions.

AUTHORS

Henry Spencer ⟨henry@spsystems.net⟩

BUGS

The list really is rather long... and probably incomplete.

NAME

signal — signal facilities

DESCRIPTION

The ⟨signal.h⟩ header file defines the following signals:

Value		*Name*	*Default ActionDescription*
1	SIGHUP	terminate process	terminal line hangup
2	SIGINT	terminate process	interrupt program
3	SIGQUIT	create core image	quit program
4	SIGILL	create core image	illegal instruction
5	SIGTRAP	create core image	trace trap
6	SIGABRT	create core image	abort(3) call (formerly SIGIOT)
7	SIGEMT	create core image	emulate instruction executed
8	SIGFPE	create core image	floating-point exception
9	SIGKILL	terminate process	kill program (cannot be caught or ignored)
10	SIGBUS	create core image	bus error
11	SIGSEGV	create core image	segmentation violation
12	SIGSYS	create core image	invalid system call argument
13	SIGPIPE	terminate process	write to a pipe with no reader
14	SIGALRM	terminate process	real-time timer expired
15	SIGTERM	terminate process	software termination signal
16	SIGURG	discard signal	urgent condition present on socket
17	SIGSTOP	stop process	stop (cannot be caught or ignored)
18	SIGTSTP	stop process	stop signal generated from keyboard
19	SIGCONT	discard signal	continue after stop
20	SIGCHLD	discard signal	child status has changed
21	SIGTTIN	stop process	background read attempted from control terminal
22	SIGTTOU	stop process	background write attempted to control terminal
23	SIGIO	discard signal	I/O is possible on a descriptor (see fcntl(2))
24	SIGXCPU	terminate process	CPU time limit exceeded (see setrlimit(2))
25	SIGXFSZ	terminate process	file size limit exceeded (see setrlimit(2))
26	SIGVTALRM	terminate process	virtual time alarm (see setitimer(2))
27	SIGPROF	terminate process	profiling timer alarm (see setitimer(2))
28	SIGWINCH	discard signal	window size change
29	SIGINFO	discard signal	status request from keyboard
30	SIGUSR1	terminate process	user-defined signal 1
31	SIGUSR2	terminate process	user-defined signal 2
32	SIGPWR	discard signal	power failure/restart

A function that is async-signal-safe is either reentrant or non-interruptible by signals. This means that they can be used in signal handlers and in the child of threaded programs after doing fork(2).

The following functions are async-signal-safe. Any function not listed below is unsafe to use in signal handlers.

_Exit(2), _exit(2), abort(3), accept(2), access(2), alarm(3), bind(2), cfgetispeed(3), cfgetospeed(3), cfsetispeed(3), cfsetospeed(3), chdir(2), chmod(2), chown(2), clock_gettime(2), close(2), connect(2), creat(3), dup(2), dup2(2), execle(3), execve(2), fchmod(2), fchown(2), fcntl(2), fdatasync(2), fork(2), fpathconf(2), fstat(2), fsync(2), ftruncate(2), getegid(2), geteuid(2), getgid(2), getgroups(2), getpeername(2), getpgrp(2), getpid(2), getppid(2), getsockname(2), getsockopt(2), getuid(2), kill(2), link(2), listen(2), lseek(2), lstat(2), mkdir(2), mkfifo(2), open(2), pathconf(2), pause(3), pipe(2), poll(2), raise(3), read(2), readlink(2), recv(2), recvfrom(2), recvmsg(2),

rename(2), rmdir(2), select(2), sem_post(3), send(2), sendmsg(2), sendto(2), setgid(2), setpgid(2), setsid(2), setsockopt(2), setuid(2), shutdown(2), sigaction(2), sigaddset(3), sigdelset(3), sigemptyset(3), sigfillset(3), sigismember(3), sleep(3), signal(3), sigpause(3), sigpending(2), sigprocmask(2), sigset(3), sigsuspend(2), sockatmark(3), socket(2), socketpair(2), stat(2), symlink(2), sysconf(3), tcdrain(3), tcflow(3), tcflush(3), tcgetattr(3), tcgetpgrp(3), tcsendbreak(3), tcsetattr(3), tcsetpgrp(3), time(3), timer_getoverrun(2), timer_gettime(2), timer_settime(2), times(3), umask(2), uname(3), unlink(2), utime(3), wait(2), waitpid(2), write(2).

STANDARDS

These signals conform to ISO/IEC 9945-1:1990 (“POSIX.1”), with the exception of SIGTRAP, SIGEMT, SIGBUS, SIGSYS, SIGURG, SIGIO, SIGXCPU, SIGXFSZ, SIGVTALRM, SIGPROF, SIGWINCH, and SIGINFO which are Berkeley extensions (available on most BSD−derived systems), and SIGPWR which comes from System V.

HISTORY

SIGPWR was introduced in NetBSD 1.4.

NOTES

The current NetBSD kernel never generates the SIGPWR signal.

SEE ALSO

kill(1), kill(2), ptrace(2), sigaction(2), sigaltstack(2), sigprocmask(2), sigstack(2), sigsuspend(2), fpgetmask(3), fpsetmask(3), setjmp(3), sigblock(3), siginterrupt(3), signal(3), sigpause(3), sigsetmask(3), sigsetops(3), tty(4)

NAME

sticky — Description of the ‘sticky’ (S_ISVTX) bit functionality

DESCRIPTION

A special file mode, called the *sticky bit* (mode S_ISVTX), is used to indicate special treatment for directories. See chmod(2) or the file /usr/include/sys/stat.h

STICKY FILES

For regular files, the use of mode S_ISVTX is reserved and can be set only by the super-user. NetBSD does not currently treat regular files that have the sticky bit set specially, but this behavior might change in the future.

STICKY DIRECTORIES

A directory whose “sticky bit” is set becomes a directory in which the deletion of files is restricted. A file in a sticky directory may only be removed or renamed by a user if the user has write permission for the directory and the user is the owner of the file, the owner of the directory, or the super-user. This feature is usefully applied to directories such as /tmp which must be publicly writable but should deny users the license to arbitrarily delete or rename each others’ files.

Any user may create a sticky directory. See chmod(1) for details about modifying file modes.

HISTORY

The sticky bit first appeared in V7, and this manual page appeared in section 8. Its initial use was to mark sharable executables that were frequently used so that they would stay in swap after the process exited. Sharable executables were compiled in a special way so their text and read-only data could be shared amongst processes. vi(1) and sh(1) were such executables. This is where the term “sticky” comes from - the program would stick around in swap, and it would not have to be fetched again from the file system. Of course as long as there was a copy in the swap area, the file was marked busy so it could not be overwritten. On V7 this meant that the file could not be removed either, because busy executables could not be removed, but this restriction was lifted in BSD releases.

To replace such executables was a cumbersome process. One had first to remove the sticky bit, then execute the binary so that the copy from swap was flushed, overwrite the executable, and finally reset the sticky bit.

Later, on SunOS 4, the sticky bit got an additional meaning for files that had the bit set and were not executable: read and write operations from and to those files would go directly to the disk and bypass the buffer cache. This was typically used on swap files for NFS clients on an NFS server, so that swap I/O generated by the clients on the servers would not evict useful data from the server’s buffer cache.

BUGS

Neither open(2) nor mkdir(2) will create a file with the sticky bit set.

NAME

symlink — symbolic link handling

DESCRIPTION

Symbolic links are files that act as pointers to other files. To understand their behavior, you must first understand how hard links work.

A hard link to a file is indistinguishable from the original file because it is a reference to the object underlying the original file name. Changes to a file are independent of the name used to reference the file. Hard links may not refer to directories and may not reference files on different file systems.

A symbolic link contains the name of the file to which it is linked, i.e. it is a pointer to another name, and not to an underlying object. For this reason, symbolic links may reference directories and may span file systems.

Because a symbolic link and its referenced object coexist in the filesystem name space, confusion can arise in distinguishing between the link itself and the referenced object. Historically, commands and system calls have adopted their own link following conventions in a somewhat ad-hoc fashion. Rules for more a uniform approach, as they are implemented in this system, are outlined here. It is important that local applications conform to these rules, too, so that the user interface can be as consistent as possible.

Symbolic links are handled either by operating on the link itself, or by operating on the object referenced by the link. In the latter case, an application or system call is said to "follow" the link.

Symbolic links may reference other symbolic links, in which case the links are dereferenced until an object that is not a symbolic link is found, a symbolic link which references a file which doesn't exist is found, or a loop is detected. Loop detection is done by placing an upper limit on the number of links that may be followed, and an error results if this limit is exceeded.

There are three separate areas that need to be discussed. They are as follows:

1. Symbolic links used as file name arguments for system calls.
2. Symbolic links specified as command line arguments to utilities that are not traversing a file tree.
3. Symbolic links encountered by utilities that are traversing a file tree (either specified on the command line or encountered as part of the file hierarchy walk).

System calls

The first area is symbolic links used as file name arguments for system calls.

Except as noted below, all system calls follow symbolic links. For example, if there were a symbolic link "`slink`" which pointed to a file named "`afile`", the system call "`open("slink" ...)`" would return a file descriptor to the file "afile".

There are eight system calls that do not follow links, and which operate on the symbolic link itself. They are: `lchflags`(2), `lchmod`(2), `lchown`(2), `lstat`(2), `lutimes`(2), `readlink`(2), `rename`(2), and `unlink`(2). Because `remove`(3) is an alias for `unlink`(2), it also does not follow symbolic links.

The 4.4BSD system differs from historical 4BSD systems in that the system call `chown`(2) has been changed to follow symbolic links.

If the filesystem is mounted with the *symperm* `mount`(8) option, the symbolic link file permission bits have the following effects:

The `readlink`(2) system call requires read permissions on the symbolic link.

System calls that follow symbolic links will fail without execute/search permissions on all the symbolic links followed.

The write, sticky, set-user-ID-on-execution and set-group-ID-on-execution symbolic link mode bits have no effect on any system calls (including `execve`(2)).

Commands not traversing a file tree

The second area is symbolic links, specified as command line file name arguments, to commands which are not traversing a file tree.

Except as noted below, commands follow symbolic links named as command line arguments. For example, if there were a symbolic link "`slink`" which pointed to a file named "`afile`", the command "`cat slink`" would display the contents of the file "`afile`".

It is important to realize that this rule includes commands which may optionally traverse file trees, e.g. the command "`chown file`" is included in this rule, while the command "`chown -R file`" is not (The latter is described in the third area, below).

If it is explicitly intended that the command operate on the symbolic link instead of following the symbolic link, e.g., it is desired that "`file slink`" display the type of file that "`slink`" is, whether it is a symbolic link or not, the **`-h`** option should be used. In the above example, "`file slink`" would report the type of the file referenced by "`slink`", while "`file -h slink`" would report that "`slink`" was a symbolic link.

There are three exceptions to this rule. The `mv`(1) and `rm`(1) commands do not follow symbolic links named as arguments, but respectively attempt to rename and delete them. (Note, if the symbolic link references a file via a relative path, moving it to another directory may very well cause it to stop working, since the path may no longer be correct).

The `ls`(1) command is also an exception to this rule. For compatibility with historic systems (when **`ls`** is not doing a tree walk, i.e. the **`-R`** option is not specified), the **`ls`** command follows symbolic links named as arguments if the **`-L`** option is specified, or if the **`-F`**, **`-d`** or **`-l`** options are not specified. (If the **`-L`** option is specified, **`ls`** always follows symbolic links. **`ls`** is the only command where the **`-L`** option affects its behavior even though it is not doing a walk of a file tree).

The 4.4BSD system differs from historical 4BSD systems in that the **`chown`**, **`chgrp`** and **`file`** commands follow symbolic links specified on the command line.

Commands traversing a file tree

The following commands either optionally or always traverse file trees: `chflags`(1), `chgrp`(1), `chmod`(1), `cp`(1), `du`(1), `find`(1), `ls`(1), `pax`(1), `rm`(1), `tar`(1) and `chown`(8).

It is important to realize that the following rules apply equally to symbolic links encountered during the file tree traversal and symbolic links listed as command line arguments.

The first rule applies to symbolic links that reference files that are not of type directory. Operations that apply to symbolic links are performed on the links themselves, but otherwise the links are ignored.

For example, the command "`chown -R user slink directory`" will ignore "`slink`", because the **`-h`** flag must be used to change owners of symbolic links. Any symbolic links encountered during the tree traversal will also be ignored. The command "`rm -r slink directory`" will remove "`slink`", as well as any symbolic links encountered in the tree traversal of "`directory`", because symbolic links may be removed. In no case will either **`chown`** or **`rm`** affect the file which "`slink`" references in any way.

The second rule applies to symbolic links that reference files of type directory. Symbolic links which reference files of type directory are never "followed" by default. This is often referred to as a "physical" walk, as opposed to a "logical" walk (where symbolic links referencing directories are followed).

As consistently as possible, you can make commands doing a file tree walk follow any symbolic links named on the command line, regardless of the type of file they reference, by specifying the **`-H`** (for "half−logical") flag. This flag is intended to make the command line name space look like the logical name space. (Note,

for commands that do not always do file tree traversals, the **-H** flag will be ignored if the **-R** flag is not also specified).

For example, the command "`chown -HR user slink`" will traverse the file hierarchy rooted in the file pointed to by "`slink`". Note, the **-H** is not the same as the previously discussed **-h** flag. The **-H** flag causes symbolic links specified on the command line to be dereferenced both for the purposes of the action to be performed and the tree walk, and it is as if the user had specified the name of the file to which the symbolic link pointed.

As consistently as possible, you can make commands doing a file tree walk follow any symbolic links named on the command line, as well as any symbolic links encountered during the traversal, regardless of the type of file they reference, by specifying the **-L** (for "logical") flag. This flag is intended to make the entire name space look like the logical name space. (Note, for commands that do not always do file tree traversals, the **-L** flag will be ignored if the **-R** flag is not also specified).

For example, the command "`chown -LR user slink`" will change the owner of the file referenced by "`slink`". If "`slink`" references a directory, **chown** will traverse the file hierarchy rooted in the directory that it references. In addition, if any symbolic links are encountered in any file tree that **chown** traverses, they will be treated in the same fashion as "`slink`".

As consistently as possible, you can specify the default behavior by specifying the **-P** (for "physical") flag. This flag is intended to make the entire name space look like the physical name space.

For commands that do not by default do file tree traversals, the **-H**, **-L** and **-P** flags are ignored if the **-R** flag is not also specified. In addition, you may specify the **-H**, **-L** and **-P** options more than once; the last one specified determines the command's behavior. This is intended to permit you to alias commands to behave one way or the other, and then override that behavior on the command line.

The `ls`(1) and `rm`(1) commands have exceptions to these rules. The **rm** command operates on the symbolic link, and not the file it references, and therefore never follows a symbolic link. The **rm** command does not support the **-H**, **-L** or **-P** options.

To maintain compatibility with historic systems, the **ls** command never follows symbolic links unless the **-L** flag is specified. If the **-L** flag is specified, **ls** follows all symbolic links, regardless of their type, whether specified on the command line or encountered in the tree walk. The **ls** command does not support the **-H** or **-P** options.

MAGIC SYMLINKS

Magic symlinks can be enabled by setting "vfs.generic.magiclinks" with `sysctl`(8). When magic symlinks are enabled "magic" patterns in symlinks are expanded. Those patterns begin with "@" (an at-sign), and end at the end of the pathname component (i.e. at the next "/", or at the end of the symbolic link if there are no more slashes).

To illustrate the pattern matching rules, assume that "@foo" is a valid magic string:

@foo	would be matched
@foo/bar	would be matched
bar@foo	would be matched
@foobar	would not be matched

Magic strings may also be delimited with '{' and '}' characters, allowing for more complex patterns in symbolic links such as:

```
@{var1}-@{var2}.@{var3}
```

The following patterns are supported:

@domainname	Expands to the machine's domain name, as set by `setdomainname`(3).
@hostname	Expands to the machine's host name, as set by `sethostname`(3).
@emul	Expands to the name of the current process's emulation.
@kernel_ident	Expands to the name of the `config`(1) file used to generate the running kernel.
@machine	Expands to the value of `MACHINE` for the system (equivalent to the output of "uname -m").
@machine_arch	Expands to the value of `MACHINE_ARCH` for the system (equivalent to the output of "uname -p").
@osrelease	Expands to the operating system release of the running kernel (equivalent to the output of "uname -r").
@ostype	Expands to the operating system type of the running kernel (equivalent to the output of "uname -s"). This will always be "NetBSD" on NetBSD systems.
@ruid	Exapnds to the real user-id of the process.
@uid	Expands to the effective user-id of the process.

SEE ALSO

`chflags`(1), `chgrp`(1), `chmod`(1), `cp`(1), `du`(1), `find`(1), `ln`(1), `ls`(1), `mv`(1), `pax`(1), `rm`(1), `tar`(1), `uname`(1), `chown`(2), `execve`(2), `lchflags`(2), `lchmod`(2), `lchown`(2), `lstat`(2), `lutimes`(2), `mount`(2), `readlink`(2), `rename`(2), `symlink`(2), `unlink`(2), `fts`(3), `remove`(3), `chown`(8), `mount`(8)

HISTORY

Magic symlinks appeared in NetBSD 4.0.

NAME

sysctl — system information variables

DESCRIPTION

The sysctl(3) library function and the sysctl(8) utility are used to get and set values of system variables, maintained by the kernel. The variables are organized in a tree and identified by a sequence of numbers, conventionally separated by dots with the topmost identifier at the left side. The numbers have corresponding text names. The sysctlnametomib(3) function or the **-M** argument to the sysctl(8) utility can be used to convert the text representation to the numeric one.

The individual sysctl variables are described below, both the textual and numeric form where applicable. The textual names can be used as argument to the sysctl(8) utility and in the file /etc/sysctl.conf. The numeric names are usually defined as preprocessor constants and are intended for use by programs. Every such constant expands to one integer, which identifies the sysctl variable relative to the upper level of the tree. See the sysctl(3) manual page for programming examples.

Top level names

The top level names are defined with a CTL_ prefix in ⟨sys/sysctl.h⟩, and are as follows. The next and subsequent levels down are found in the include files listed here, and described in separate sections below.

Name	Constant	Next level names	Description
kern	CTL_KERN	sys/sysctl.h	High kernel limits
vm	CTL_VM	uvm/uvm_param.h	Virtual memory
vfs	CTL_VFS	sys/mount.h	Filesystem
net	CTL_NET	sys/socket.h	Networking
debug	CTL_DEBUG	sys/sysctl.h	Debugging
hw	CTL_HW	sys/sysctl.h	Generic CPU, I/O
machdep	CTL_MACHDEP	sys/sysctl.h	Machine dependent
user	CTL_USER	sys/sysctl.h	User-level
ddb	CTL_DDB	sys/sysctl.h	In-kernel debugger
proc	CTL_PROC	sys/sysctl.h	Per-process
vendor	CTL_VENDOR	?	Vendor specific
emul	CTL_EMUL	sys/sysctl.h	Emulation settings
security	CTL_SECURITY	sys/sysctl.h	Security settings

The debug.* subtree

The debugging variables vary from system to system. A debugging variable may be added or deleted without need to recompile **sysctl** to know about it. Each time it runs, **sysctl** gets the list of debugging variables from the kernel and displays their current values. The system defines twenty (*struct ctldebug*) variables named debug0 through debug19. They are declared as separate variables so that they can be individually initialized at the location of their associated variable. The loader prevents multiple use of the same variable by issuing errors if a variable is initialized in more than one place. For example, to export the variable dospecialcheck as a debugging variable, the following declaration would be used:

```
int dospecialcheck = 1;
struct ctldebug debug5 = { "dospecialcheck", &dospecialcheck };
```

Note that the dynamic implementation of **sysctl** currently in use largely makes this particular **sysctl** interface obsolete. See sysctl(8) for more information.

The vfs.* subtree

A distinguished second level name, vfs.generic (VFS_GENERIC), is used to get general information about all filesystems. One of its third level identifiers is vfs.generic.maxtypenum (VFS_MAXTYPENUM) that gives the highest valid filesystem type number. Its other third level identifier is

`vfs.generic.conf` (`VFS_CONF`) that returns configuration information about the filesystem type given as a fourth level identifier. The remaining second level identifiers are the filesystem type number returned by a `statvfs`(2) call or from `vfs.generic.conf`. The third level identifiers available for each filesystem are given in the header file that defines the mount argument structure for that filesystem.

The hw.* subtree

The string and integer information available for the `hw` level is detailed below. The changeable column shows whether a process with appropriate privilege may change the value.

Second level name	Type	Changeable
hw.alignbytes	integer	no
hw.byteorder	integer	no
hw.cnmagic	string	yes
hw.disknames	string	no
hw.diskstats	struct	no
hw.machine	string	no
hw.machine_arch	string	no
hw.model	string	no
hw.ncpu	integer	no
hw.pagesize	integer	no
hw.physmem	integer	no
hw.physmem64	quad	no
hw.usermem	integer	no
hw.usermem64	quad	no

`hw.alignbytes` (`HW_ALIGNBYTES`)
: Alignment constraint for all possible data types. This shows the value `ALIGNBYTES` in `/usr/include/machine/param.h`, at the kernel compilation time.

`hw.byteorder` (`HW_BYTEORDER`)
: The byteorder (4,321, or 1,234).

`hw.cnmagic` (`HW_CNMAGIC`)
: The console magic key sequence.

`hw.disknames` (`HW_DISKNAMES`)
: The list of (space separated) disk device names on the system.

`hw.iostatnames` (`HW_IOSTATNAMES`)
: A space separated list of devices that will have I/O statistics collected on them.

`hw.iostats` (`HW_IOSTATS`)
: Return statistical information on the NFS mounts, disk and tape devices on the system. An array of *struct io_sysctl* structures is returned, whose size depends on the current number of such objects in the system. The third level name is the size of the *struct io_sysctl*. The type of object can be determined by examining the *type* element of *struct io_sysctl*. Which can be `IOSTAT_DISK` (disk drive), `IOSTAT_TAPE` (tape drive), or `IOSTAT_NFS` (NFS mount).

`hw.machine` (`HW_MACHINE`)
: The machine class.

`hw.machine_arch` (`HW_MACHINE_ARCH`)
: The machine CPU class.

`hw.model` (`HW_MODEL`)
: The machine model.

`hw.ncpu` (`HW_NCPU`)
The number of CPUs.

`hw.pagesize` (`HW_PAGESIZE`)
The software page size.

`hw.physmem` (`HW_PHYSMEM`)
The bytes of physical memory as a 32-bit integer.

`hw.physmem64` (`HW_PHYSMEM64`)
The bytes of physical memory as a 64-bit integer.

`hw.usermem` (`HW_USERMEM`)
The bytes of non-kernel memory as a 32-bit integer.

`hw.usermem64` (`HW_USERMEM64`)
The bytes of non-kernel memory as a 64-bit integer.

The kern.* subtree

The string and integer information available for the `kern` level is detailed below. The changeable column shows whether a process with appropriate privilege may change the value. The types of data currently available are process information, system vnodes, the open file entries, routing table entries, virtual memory statistics, load average history, and clock rate information.

Second level name	Type	Changeable
kern.argmax	integer	no
kern.autonicetime	integer	yes
kern.autoniceval	integer	yes
kern.boottime	struct timeval	no
kern.bufq	node	not applicable
kern.ccpu	integer	no
kern.clockrate	struct clockinfo	no
kern.consdev	integer	no
kern.cp_id	struct	no
kern.cp_time	uint64_t[ ]	no
kern.defcorename	string	yes
kern.domainname	string	yes
kern.drivers	struct kinfo_drivers	no
kern.file	struct file	no
kern.forkfsleep	integer	yes
kern.fscale	integer	no
kern.fsync	integer	no
kern.hardclock_ticks	integer	no
kern.hostid	integer	yes
kern.hostname	string	yes
kern.iov_max	integer	no
kern.job_control	integer	no
kern.labeloffset	integer	no
kern.labelsector	integer	no
kern.login_name_max	integer	no
kern.logsigexit	integer	yes
kern.mapped_files	integer	no

kern.maxfiles	integer	yes
kern.maxpartitions	integer	no
kern.maxphys	integer	no
kern.maxproc	integer	yes
kern.maxptys	integer	yes
kern.maxvnodes	integer	yes
kern.mbuf	node	not applicable
kern.memlock	integer	no
kern.memlock_range	integer	no
kern.memory_protection	integer	no
kern.monotonic_clock	integer	no
kern.msgbuf	integer	no
kern.msgbufsize	integer	no
kern.ngroups	integer	no
kern.ntptime	struct ntptimeval	no
kern.osrelease	string	no
kern.osrev	integer	no
kern.ostype	string	no
kern.pipe	node	not applicable
kern.posix1	integer	no
kern.posix_barriers	integer	no
kern.posix_reader_writer_locks	integer	no
kern.posix_semaphores	integer	no
kern.posix_spin_locks	integer	no
kern.posix_threads	integer	no
kern.posix_timers	integer	no
kern.proc	struct kinfo_proc	no
kern.proc2	struct kinfo_proc2	no
kern.proc_args	string	no
kern.prof	node	not applicable
kern.rawpartition	integer	no
kern.root_device	string	no
kern.root_partition	integer	no
kern.rtc_offset	integer	yes
kern.saved_ids	integer	no
kern.securelevel	integer	raise only
kern.synchronized_io	integer	no
kern.ipc	node	not applicable
kern.timex	struct	no
kern.tkstat	node	not applicable
kern.urandom	integer	no
kern.version	string	no
kern.vnode	struct vnode	no

`kern.argmax` (`KERN_ARGMAX`)
The maximum bytes of argument to `execve`(2).

`kern.autonicetime` (`KERN_AUTONICETIME`)
The number of seconds of CPU-time a non-root process may accumulate before having its priority lowered from the default to the value of KERN_AUTONICEVAL. If set to 0, automatic lowering of priority is not performed, and if set to −1 all non-root processes are immediately lowered.

`kern.autoniceval` (`KERN_AUTONICEVAL`)
The priority assigned for automatically niced processes.

`kern.boottime` (`KERN_BOOTTIME`)
A *struct timeval* structure is returned. This structure contains the time that the system was booted.

`kern.ccpu` (`KERN_CCPU`)
The scheduler exponential decay value.

`kern.clockrate` (`KERN_CLOCKRATE`)
A *struct clockinfo* structure is returned. This structure contains the clock, statistics clock and profiling clock frequencies, the number of micro-seconds per hz tick, and the clock skew rate.

`kern.consdev` (`KERN_CONSDEV`)
Console device.

`kern.cp_id` (`KERN_CP_ID`)
Mapping of CPU number to CPU id.

`kern.cp_time` (`KERN_CP_TIME`)
Returns an array of CPUSTATES uint64_ts. This array contains the number of clock ticks spent in different CPU states. On multi-processor systems, the sum across all CPUs is returned unless appropriate space is given for one data set for each CPU. Data for a specific CPU can also be obtained by adding the number of the CPU at the end of the MIB, enlarging it by one.

`kern.defcorename` (`KERN_DEFCORENAME`)
Default template for the name of core dump files (see also `proc.pid.corename` in the per-process variables `proc.`∗, and `core`(5) for format of this template). The default value is **`%n.core`** and can be changed with the kernel configuration option **`options DEFCORENAME`** (see `options`(4)).

`kern.domainname` (`KERN_DOMAINNAME`)
Get or set the YP domain name.

`kern.dump_on_panic` (`KERN_DUMP_ON_PANIC`)
Perform a crash dump on system panic.

`kern.drivers` (`KERN_DRIVERS`)
Return an array of *struct kinfo_drivers* that contains the name and major device numbers of all the device drivers in the current kernel. The *d_name* field is always a NUL terminated string. The *d_bmajor* field will be set to −1 if the driver doesn't have a block device.

`kern.file` (`KERN_FILE`)
Return the entire file table. The returned data consists of a single *struct filelist* followed by an array of *struct file*, whose size depends on the current number of such objects in the system.

`kern.forkfsleep` (`KERN_FORKFSLEEP`)
If `fork`(2) system call fails due to limit on number of processes (either the global maxproc limit or user's one), wait for this many milliseconds before returning `EAGAIN` error to process. Useful to keep heavily forking runaway processes in bay. Default zero (no sleep). Maximum is 20 seconds.

`kern.fscale` (`KERN_FSCALE`)
The kernel fixed-point scale factor.

`kern.fsync` (`KERN_FSYNC`)
Return 1 if the POSIX 1003.1b File Synchronization Option is available on this system, otherwise 0.

`kern.hardclock_ticks` (`KERN_HARDCLOCK_TICKS`)
Returns the number of `hardclock`(9) ticks.

`kern.hostid` (`KERN_HOSTID`)
Get or set the host id.

`kern.hostname` (`KERN_HOSTNAME`)
Get or set the hostname.

`kern.iov_max` (`KERN_IOV_MAX`)
Return the maximum number of *iovec* structures that a process has available for use with `preadv`(2), `pwritev`(2), `readv`(2), `recvmsg`(2), `sendmsg`(2) and `writev`(2).

`kern.job_control` (`KERN_JOB_CONTROL`)
Return 1 if job control is available on this system, otherwise 0.

`kern.labeloffset` (`KERN_LABELOFFSET`)
The offset within the sector specified by KERN_LABELSECTOR of the `disklabel`(5).

`kern.labelsector` (`KERN_LABELSECTOR`)
The sector number containing the `disklabel`(5).

`kern.login_name_max` (`KERN_LOGIN_NAME_MAX`)
The size of the storage required for a login name, in bytes, including the terminating NUL.

`kern.logsigexit` (`KERN_LOGSIGEXIT`)
If this flag is non-zero, the kernel will `log`(9) all process exits due to signals which create a `core`(5) file, and whether the coredump was created.

`kern.mapped_files` (`KERN_MAPPED_FILES`)
Returns 1 if the POSIX 1003.1b Memory Mapped Files Option is available on this system, otherwise 0.

`kern.maxfiles` (`KERN_MAXFILES`)
The maximum number of open files that may be open in the system.

`kern.maxpartitions` (`KERN_MAXPARTITIONS`)
The maximum number of partitions allowed per disk.

`kern.maxphys` (`KERN_MAXPHYS`)
Maximum raw I/O transfer size.

`kern.maxproc` (`KERN_MAXPROC`)
The maximum number of simultaneous processes the system will allow.

`kern.maxptys` (`KERN_MAXPTYS`)
The maximum number of pseudo terminals. This value can be both raised and lowered, though it cannot be set lower than number of currently used ptys. See also `pty`(4).

`kern.maxvnodes` (`KERN_MAXVNODES`)
The maximum number of vnodes available on the system. This can only be raised.

`kern.mbuf` (`KERN_MBUF`)
Return information about the mbuf control variables. Mbufs are data structures which store network packets and other data structures in the networking code, see `mbuf`(9). The third level names for the mbuf variables are detailed below. The changeable column shows whether a process with appropriate privilege may change the value.

Third level name	Type	Changeable
kern.mbuf.mblowat	integer	yes
kern.mbuf.mclbytes	integer	yes
kern.mbuf.mcllowat	integer	yes
kern.mbuf.msize	integer	yes
kern.mbuf.nmbclusters	integer	yes

The variables are as follows:

`kern.mbuf.mblowat` (`MBUF_MBLOWAT`)
The mbuf low water mark.

`kern.mbuf.mclbytes` (`MBUF_MCLBYTES`)
The mbuf cluster size.

`kern.mbuf.mcllowat` (`MBUF_MCLLOWAT`)
The mbuf cluster low water mark.

`kern.mbuf.msize` (`MBUF_MSIZE`)
The mbuf base size.

`kern.mbuf.nmbclusters` (`MBUF_NMBCLUSTERS`)
The limit on the number of mbuf clusters. The variable can only be increased, and only increased on machines with direct-mapped pool pages.

`kern.memlock` (`KERN_MEMLOCK`)
Returns 1 if the POSIX 1003.1b Process Memory Locking Option is available on this system, otherwise 0.

`kern.memlock_range` (`KERN_MEMLOCK_RANGE`)
Returns 1 if the POSIX 1003.1b Range Memory Locking Option is available on this system, otherwise 0.

`kern.memory_protection` (`KERN_MEMORY_PROTECTION`)
Returns 1 if the POSIX 1003.1b Memory Protection Option is available on this system, otherwise 0.

`kern.monotonic_clock` (`KERN_MONOTONIC_CLOCK`)
Returns the standard version the implementation of the POSIX 1003.1b Monotonic Clock Option conforms to, otherwise 0.

`kern.msgbuf` (`KERN_MSGBUF`)
The kernel message buffer, rotated so that the head of the circular kernel message buffer is at the start of the returned data. The returned data may contain NUL bytes.

`kern.msgbufsize` (`KERN_MSGBUFSIZE`)
The maximum number of characters that the kernel message buffer can hold.

`kern.ngroups` (`KERN_NGROUPS`)
The maximum number of supplemental groups.

`kern.ntptime` (`KERN_NTPTIME`)
A *struct ntptimeval* structure is returned. This structure contains data used by the `ntpd`(8) program.

`kern.osrelease` (`KERN_OSRELEASE`)
The system release string.

`kern.osrevision` (`KERN_OSREV`)
The system revision string.

`kern.ostype` (`KERN_OSTYPE`)
The system type string.

`kern.pipe` (`KERN_PIPE`)
Pipe settings. The third level names for the integer pipe settings is detailed below. The changeable column shows whether a process with appropriate privilege may change the value.

Third level name	Type	Changeable
kern.pipe.kvasiz	integer	yes
kern.pipe.maxbigpipes	integer	yes
kern.pipe.maxkvasz	integer	yes
kern.pipe.limitkva	integer	yes
kern.pipe.nbigpipes	integer	yes

The variables are as follows:

`kern.pipe.kvasiz` (`KERN_PIPE_KVASIZ`)
Amount of kernel memory consumed by pipe buffers.

`kern.pipe.maxbigpipes` (`KERN_PIPE_MAXBIGPIPES`)
Maximum number of "big" pipes.

`kern.pipe.maxkvasz` (`KERN_PIPE_MAXKVASZ`)
Maximum amount of kernel memory to be used for pipes.

`kern.pipe.limitkva` (`KERN_PIPE_LIMITKVA`)
Limit for direct transfers via page loan.

`kern.pipe.nbigpipes` (`KERN_PIPE_NBIGPIPES`)
Number of "big" pipes.

`kern.posix1version` (`KERN_POSIX1`)
The version of ISO/IEC 9945 (POSIX 1003.1) with which the system attempts to comply.

`kern.posix_barriers` (`KERN_POSIX_BARRIERS`)
The version of IEEE Std 1003.1 (“POSIX.1”) and its Barriers option to which the system attempts to conform, otherwise 0.

`kern.posix_reader_writer_locks` (`KERN_POSIX_READER_WRITER_LOCKS`)
The version of IEEE Std 1003.1 (“POSIX.1”) and its Read-Write Locks option to which the system attempts to conform, otherwise 0.

`kern.posix_semaphores` (`KERN_POSIX_SEMAPHORES`)
The version of IEEE Std 1003.1 (“POSIX.1”) and its Semaphores option to which the system attempts to conform, otherwise 0.

`kern.posix_spin_locks` (`KERN_POSIX_SPIN_LOCKS`)
The version of IEEE Std 1003.1 (“POSIX.1”) and its Spin Locks option to which the system attempts to conform, otherwise 0.

`kern.posix_threads` (`KERN_POSIX_THREADS`)
The version of IEEE Std 1003.1 (“POSIX.1”) and its Threads option to which the system attempts to conform, otherwise 0.

`kern.posix_timers` (`KERN_POSIX_TIMERS`)
The version of IEEE Std 1003.1 (“POSIX.1”) and its Timers option to which the system attempts to conform, otherwise 0.

`kern.proc` (`KERN_PROC`)
Return the entire process table, or a subset of it. An array of *struct kinfo_proc* structures is returned, whose size depends on the current number of such objects in the system. The third and fourth level numeric names are as follows:

Third level name	**Fourth level is:**
KERN_PROC_ALL	None
KERN_PROC_GID	A group ID
KERN_PROC_PID	A process ID
KERN_PROC_PGRP	A process group
KERN_PROC_RGID	A real group ID
KERN_PROC_RUID	A real user ID
KERN_PROC_SESSION	A session ID
KERN_PROC_TTY	A tty device
KERN_PROC_UID	A user ID

`kern.proc2` (`KERN_PROC2`)
As for KERN_PROC, but an array of *struct kinfo_proc2* structures are returned. The fifth level name is the size of the *struct kinfo_proc2* and the sixth level name is the number of structures to return.

`kern.proc_args` (`KERN_PROC_ARGS`)
Return the argv or environment strings (or the number thereof) of a process. Multiple strings are returned separated by NUL characters. The third level name is the process ID. The fourth level name is as follows:

KERN_PROC_ARGV	The argv strings
KERN_PROC_ENV	The environ strings
KERN_PROC_NARGV	The number of argv strings
KERN_PROC_NENV	The number of environ strings

`kern.profiling` (`KERN_PROF`)
Return profiling information about the kernel. If the kernel is not compiled for profiling, attempts to retrieve any of the KERN_PROF values will fail with `EOPNOTSUPP`. The third level names for the string and integer profiling information is detailed below. The changeable column shows whether a process with appropriate privilege may change the value.

Third level name	**Type**	**Changeable**
kern.profiling.count	u_short[ ]	yes
kern.profiling.froms	u_short[ ]	yes
kern.profiling.gmonparam	struct gmonparam	no
kern.profiling.state	integer	yes
kern.profiling.tos	struct tostruct	yes

The variables are as follows:

`kern.profiling.count` (`GPROF_COUNT`)
Array of statistical program counter counts.

`kern.profiling.froms` (`GPROF_FROMS`)
Array indexed by program counter of call-from points.

`kern.profiling.gmonparams` (`GPROF_GMONPARAM`)
Structure giving the sizes of the above arrays.

`kern.profiling.state` (`GPROF_STATE`)
Profiling state. If set to GMON_PROF_ON, starts profiling. If set to GMON_PROF_OFF, stops profiling.

`kern.profiling.tos` (`GPROF_TOS`)
Array of *struct tostruct* describing destination of calls and their counts.

`kern.rawpartition` (`KERN_RAWPARTITION`)
The raw partition of a disk (a == 0).

`kern.root_device` (`KERN_ROOT_DEVICE`)
The name of the root device (e.g., "wd0").

`kern.root_partition` (`KERN_ROOT_PARTITION`)
The root partition on the root device (a == 0).

`kern.rtc_offset` (`KERN_RTC_OFFSET`)
Return the offset of real time clock from UTC in minutes.

`kern.saved_ids` (`KERN_SAVED_IDS`)
Returns 1 if saved set-group and saved set-user ID is available.

`kern.sbmax` (`KERN_SBMAX`)
Maximum socket buffer size.

`kern.securelevel` (`KERN_SECURELVL`)
The system security level. This level may be raised by processes with appropriate privilege. It may only be lowered by process 1.

`kern.somaxkva` (`KERN_SOMAXKVA`)
Maximum amount of kernel memory to be used for socket buffers.

`kern.synchronized_io` (`KERN_SYNCHRONIZED_IO`)
Returns 1 if the POSIX 1003.1b Synchronized I/O Option is available on this system, otherwise 0.

`kern.ipc` (`KERN_SYSVIPC`)
Return information about the SysV IPC parameters. The third level names for the ipc variables are detailed below.

Third level name	Type	Changeable
kern.ipc.sysvmsg	integer	no
kern.ipc.sysvsem	integer	no
kern.ipc.sysvshm	integer	no
kern.ipc.sysvipc_info	struct	no
kern.ipc.shmmax	integer	no
kern.ipc.shmmni	integer	yes
kern.ipc.shmseg	integer	yes
kern.ipc.shmmaxpgs	integer	yes
kern.ipc.shm_use_phys	integer	yes

`kern.ipc.sysvmsg` (`KERN_SYSVIPC_MSG`)
Returns 1 if System V style message queue functionality is available on this system, otherwise 0.

`kern.ipc.sysvsem` (`KERN_SYSVIPC_SEM`)
Returns 1 if System V style semaphore functionality is available on this system, otherwise 0.

`kern.ipc.sysvshm` (`KERN_SYSVIPC_SHM`)
Returns 1 if System V style share memory functionality is available on this system, otherwise 0.

`kern.ipc.sysvipc_info` (`KERN_SYSVIPC_INFO`)
Return System V style IPC configuration and run-time information. The fourth level name selects the System V style IPC facility.

Fourth level name	Type
KERN_SYSVIPC_MSG_INFO	struct msg_sysctl_info
KERN_SYSVIPC_SEM_INFO	struct sem_sysctl_info
KERN_SYSVIPC_SHM_INFO	struct shm_sysctl_info

`KERN_SYSVIPC_MSG_INFO`
Return information on the System V style message facility. The **msg_sysctl_info** structure is defined in ⟨`sys/msg.h`⟩.

`KERN_SYSVIPC_SEM_INFO`
Return information on the System V style semaphore facility. The **sem_sysctl_info** structure is defined in ⟨`sys/sem.h`⟩.

`KERN_SYSVIPC_SHM_INFO`
Return information on the System V style shared memory facility. The **shm_sysctl_info** structure is defined in ⟨`sys/shm.h`⟩.

`kern.ipc.shmmax` (`KERN_SYSVIPC_SHMMAX`)
Max shared memory segment size in bytes.

`kern.ipc.shmmni` (`KERN_SYSVIPC_SHMMNI`)
Max number of shared memory identifiers.

`kern.ipc.shmseg` (`KERN_SYSVIPC_SHMSEG`)
Max shared memory segments per process.

`kern.ipc.shmmaxpgs` (`KERN_SYSVIPC_SHMMAXPGS`)
Max amount of shared memory in pages.

`kern.ipc.shm_use_phys` (`KERN_SYSVIPC_SHMUSEPHYS`)
Locking of shared memory in physical memory. If 0, memory can be swapped out, otherwise it will be locked in physical memory.

`kern.timex` (`KERN_TIMEX`)
Not available.

`kern.tkstat` (`KERN_TKSTAT`)
Return information about the number of characters sent and received on ttys. The third level names for the tty statistic variables are detailed below. The changeable column shows whether a process with appropriate privilege may change the value.

Third level name	Type	Changeable
kern.tkstat.cancc	quad	no
kern.tkstat.nin	quad	no
kern.tkstat.nout	quad	no
kern.tkstat.rawcc	quad	no

The variables are as follows:

`kern.tkstat.cancc` (`KERN_TKSTAT_CANCC`)
The number of canonical input characters.

`kern.tkstat.nin` (`KERN_TKSTAT_NIN`)
The total number of input characters.

`kern.tkstat.nout` (`KERN_TKSTAT_NOUT`)
The total number of output characters.

`kern.tkstat.rawcc` (`KERN_TKSTAT_RAWCC`)
The number of raw input characters.

`kern.urandom` (`KERN_URND`)
Random integer value.

`kern.veriexec`
Tunings for Verixec.

`kern.veriexec.algorithms`
Returns a string with the supported algorithms in Veriexec.

`kern.veriexec.count`
Sub-nodes are added to this node as new mounts are monitored by Veriexec. Each mount will be under its own tableN node. Under each node there will be three variables, indicating the mount point, the file-system type, and the number of entries.

`kern.veriexec.strict`
Controls the strict level of Veriexec. See `security`(8) for more information on each level's implications.

`kern.veriexec.verbose`
Controls the verbosity level of Veriexec. If 0, only the minimal indication required will be given about what's happening - fingerprint mismatches, removal of entries from the tables, modification of a fingerprinted file. If 1, more messages will be printed (ie., when a file with a valid fingerprint is accessed). Verbose level 2 is debug mode.

`kern.version` (`KERN_VERSION`)
The system version string.

`kern.vnode` (`KERN_VNODE`)
Return the entire vnode table. Note, the vnode table is not necessarily a consistent snapshot of the system. The returned data consists of an array whose size depends on the current number of such objects in the system. Each element of the array contains the kernel address of a vnode *struct vnode* $*$ followed by the vnode itself *struct vnode*.

`kern.coredump.setid`
Settings related to set-id processes coredumps. By default, set-id processes do not dump core in situations where other processes would. The settings in this node allows an administrator to change this behavior.

`kern.coredump.setid.dump`
If non-zero, set-id processes will dump core.

`kern.coredump.setid.group`
The group-id for the set-id processes' coredump.

`kern.coredump.setid.mode`
The mode for the set-id processes' coredump. See `chmod`(1).

`kern.coredump.setid.owner`
The user-id that will be used as the owner of the set-id processes' coredump.

`kern.coredump.setid.path`
The path to which set-id processes' coredumps will be saved to. Same syntax as kern.defcorename.

The machdep.∗ subtree

The set of variables defined is architecture dependent. Most architectures define at least the following variables.

Second level name	Type	Changeable
`CPU_CONSDEV`	`dev_t`	`no`

The net.∗ subtree

The string and integer information available for the `net` level is detailed below. The changeable column shows whether a process with appropriate privilege may change the value. The second and third levels are typically the protocol family and protocol number, though this is not always the case.

Second level name	Type	Changeable
net.route	routing messages	no
net.inet	IPv4 values	yes
net.inet6	IPv6 values	yes
net.key	IPsec key management values	yes

`net.route` (`PF_ROUTE`)
Return the entire routing table or a subset of it. The data is returned as a sequence of routing messages (see `route`(4) for the header file, format and meaning). The length of each message is contained in the message header.

The third level name is a protocol number, which is currently always 0. The fourth level name is an address family, which may be set to 0 to select all address families. The fifth and sixth level names are as follows:

Fifth level name	Sixth level is:
NET_RT_FLAGS	rtflags
NET_RT_DUMP	None
NET_RT_IFLIST	None

`net.inet` (`PF_INET`)
Get or set various global information about the IPv4 (Internet Protocol version 4). The third level name is the protocol. The fourth level name is the variable name. The currently defined protocols and names are:

Protocol name	Variable name	Type	Changeable
arp	down	integer	yes
arp	keep	integer	yes
arp	prune	integer	yes
arp	refresh	integer	yes
carp	allow	integer	yes
carp	preempt	integer	yes
carp	log	integer	yes
carp	arpbalance	integer	yes

icmp	errppslimit	integer	yes
icmp	maskrepl	integer	yes
icmp	rediraccept	integer	yes
icmp	redirtimeout	integer	yes
ip	allowsrcrt	integer	yes
ip	anonportmax	integer	yes
ip	anonportmin	integer	yes
ip	checkinterface	integer	yes
ip	directed-broadcast	integer	yes
ip	do_loopback_cksum	integer	yes
ip	forwarding	integer	yes
ip	forwsrcrt	integer	yes
ip	gifttl	integer	yes
ip	grettl	integer	yes
ip	hashsize	integer	yes
ip	hostzerobroadcast	integer	yes
ip	lowportmin	integer	yes
ip	lowportmax	integer	yes
ip	maxflows	integer	yes
ip	maxfragpackets	integer	yes
ip	mtudisc	integer	yes
ip	mtudisctimeout	integer	yes
ip	random_id	integer	yes
ip	redirect	integer	yes
ip	subnetsarelocal	integer	yes
ip	ttl	integer	yes
tcp	rfc1323	integer	yes
tcp	sendspace	integer	yes
tcp	recvspace	integer	yes
tcp	mssdflt	integer	yes
tcp	syn_cache_limit	integer	yes
tcp	syn_bucket_limit	integer	yes
tcp	syn_cache_interval	integer	yes
tcp	init_win	integer	yes
tcp	init_win_local	integer	yes
tcp	mss_ifmtu	integer	yes
tcp	win_scale	integer	yes
tcp	timestamps	integer	yes
tcp	compat_42	integer	yes
tcp	cwm	integer	yes
tcp	cwm_burstsize	integer	yes
tcp	ack_on_push	integer	yes
tcp	keepidle	integer	yes
tcp	keepintvl	integer	yes
tcp	keepcnt	integer	yes
tcp	slowhz	integer	no
tcp	keepinit	integer	yes
tcp	log_refused	integer	yes
tcp	rstppslimit	integer	yes

tcp	ident	struct	no
tcp	drop	struct	no
tcp	sack.enable	integer	yes
tcp	sack.globalholes	integer	no
tcp	sack.globalmaxholes	integer	yes
tcp	sack.maxholes	integer	yes
tcp	ecn.enable	integer	yes
tcp	ecn.maxretries	integer	yes
tcp	congctl.selected	string	yes
tcp	congctl.available	string	yes
tcp	abc.enable	integer	yes
tcp	abc.aggressive	integer	yes
udp	checksum	integer	yes
udp	do_loopback_cksum	integer	yes
udp	recvspace	integer	yes
udp	sendspace	integer	yes

The variables are as follows:

`arp.down`
: Failed ARP entry lifetime.

`arp.keep`
: Valid ARP entry lifetime.

`arp.prune`
: ARP cache pruning interval.

`arp.refresh`
: ARP entry refresh interval.

`carp.allow`
: If set to 0, incoming `carp`(4) packets will not be processed. If set to any other value, processing will occur. Enabled by default.

`carp.arpbalance`
: If set to any value other than 0, the ARP balancing functionality of `carp`(4) is enabled. When ARP requests are received for an IP address which is part of any virtual host, carp will hash the source IP in the ARP request to select one of the virtual hosts from the set of all the virtual hosts which have that IP address. The master of that host will respond with the correct virtual MAC address. Disabled by default.

`carp.log`
: If set to any value other than 0, `carp`(4) will log errors. Disabled by default.

`carp.preempt`
: If set to 0, `carp`(4) will not attempt to become master if it is receiving advertisements from another active master. If set to any other value, carp will become master of the virtual host if it believes it can send advertisements more frequently than the current master. Disabled by default.

`ip.allowsrcrt`
: If set to 1, the host accepts source routed packets.

`ip.anonportmax`
: The highest port number to use for TCP and UDP ephemeral port allocation. This cannot be set to less than 1024 or greater than 65535, and must be greater than

`ip.anonportmin`.

`ip.anonportmin`
: The lowest port number to use for TCP and UDP ephemeral port allocation. This cannot be set to less than 1024 or greater than 65535.

`ip.checkinterface`
: If set to non-zero, the host will reject packets addressed to it that arrive on an interface not bound to that address. Currently, this must be disabled if ipnat is used to translate the destination address to another local interface, or if addresses are added to the loopback interface instead of the interface where the packets for those packets are received.

`ip.directed-broadcast`
: If set to 1, enables directed broadcast behavior for the host.

`ip.do_loopback_cksum`
: Perform IP checksum on loopback.

`ip.forwarding`
: If set to 1, enables IP forwarding for the host, meaning that the host is acting as a router.

`ip.forwsrcrt`
: If set to 1, enables forwarding of source-routed packets for the host. This value may only be changed if the kernel security level is less than 1.

`ip.gifttl`
: The maximum time-to-live (hop count) value for an IPv4 packet generated by `gif`(4) tunnel interface.

`ip.grettl`
: The maximum time-to-live (hop count) value for an IPv4 packet generated by `gre`(4) tunnel interface.

`ip.hashsize`
: The size of IPv4 Fast Forward hash table. This value must be a power of 2 (64, 256...). A larger hash table size results in fewer collisions. Also see `ip.maxflows`.

`ip.hostzerobroadcast`
: All zeroes address is broadcast address.

`ip.lowportmax`
: The highest port number to use for TCP and UDP reserved port allocation. This cannot be set to less than 0 or greater than 1024, and must be greater than `ip.lowportmin`.

`ip.lowportmin`
: The lowest port number to use for TCP and UDP reserved port allocation. This cannot be set to less than 0 or greater than 1024, and must be smaller than `ip.lowportmax`.

`ip.maxflows`
: IPv4 Fast Forwarding is enabled by default. If set to 0, IPv4 Fast Forwarding is disabled. `ip.maxflows` controls the maximum amount of flows which can be created. The default value is 256.

`ip.maxfragpackets`
: The maximum number of fragmented packets the node will accept. 0 means that the node will not accept any fragmented packets. −1 means that the node will accept as many fragmented packets as it receives. The flag is provided basically for avoiding possible DoS attacks.

`ip.mtudisc`
If set to 1, enables Path MTU Discovery (RFC 1191). When Path MTU Discovery is enabled, the transmitted TCP segment size will be determined by the advertised maximum segment size (MSS) from the remote end, as constrained by the path MTU. If MTU Discovery is disabled, the transmitted segment size will never be greater than `tcp.mssdflt` (the local maximum segment size).

`ip.mtudisctimeout`
The number of seconds in which a route added by the Path MTU Discovery engine will time out. When the route times out, the Path MTU Discovery engine will attempt to probe a larger path MTU.

`ip.random_id`
Assign random ip_id values.

`ip.redirect`
If set to 1, ICMP redirects may be sent by the host. This option is ignored unless the host is routing IP packets, and should normally be enabled on all systems.

`ip.subnetsarelocal`
If set to 1, subnets are to be considered local addresses.

`ip.ttl` The maximum time-to-live (hop count) value for an IP packet sourced by the system. This value applies to normal transport protocols, not to ICMP.

`icmp.errppslimit`
The variable specifies the maximum number of outgoing ICMP error messages, per second. ICMP error messages that exceeded the value are subject to rate limitation and will not go out from the node. Negative value disables rate limitation.

`icmp.maskrepl`
If set to 1, ICMP network mask requests are to be answered.

`icmp.rediraccept`
If set to non-zero, the host will accept ICMP redirect packets. Note that routers will never accept ICMP redirect packets, and the variable is meaningful on IP hosts only.

`icmp.redirtimeout`
The variable specifies lifetime of routing entries generated by incoming ICMP redirect. This defaults to 600 seconds.

`icmp.returndatabytes`
Number of bytes to return in an ICMP error message.

`tcp.ack_on_push`
If set to 1, TCP is to immediately transmit an ACK upon reception of a packet with PUSH set. This can avoid losing a round trip time in some rare situations, but has the caveat of potentially defeating TCP's delayed ACK algorithm. Use of this option is generally not recommended, but the variable exists in case your configuration really needs it.

`tcp.compat_42`
If set to 1, enables work-arounds for bugs in the 4.2BSD TCP implementation. Use of this option is not recommended, although it may be required in order to communicate with extremely old TCP implementations.

`tcp.cwm`
: If set to 1, enables use of the Hughes/Touch/Heidemann Congestion Window Monitoring algorithm. This algorithm prevents line-rate bursts of packets that could otherwise occur when data begins flowing on an idle TCP connection. These line-rate bursts can contribute to network and router congestion. This can be particularly useful on World Wide Web servers which support HTTP/1.1, which has lingering connections.

`tcp.cwm_burstsize`
: The Congestion Window Monitoring allowed burst size, in terms of packet count.

`tcp.delack_ticks`
: Number of ticks to delay sending an ACK.

`tcp.do_loopback_cksum`
: Perform TCP checksum on loopback.

`tcp.init_win`
: A value indicating the TCP initial congestion window. If this value is 0, an auto-tuning algorithm designed to use an initial window of approximately 4K bytes is in use. Otherwise, this value indicates a fixed number of packets.

`tcp.init_win_local`
: Like `tcp.init_win`, but used when communicating with hosts on a local network.

`tcp.keepcnt`
: Number of keepalive probes sent before declaring a connection dead. If set to zero, there is no limit; keepalives will be sent until some kind of response is received from the peer.

`tcp.keepidle`
: Time a connection must be idle before keepalives are sent (if keepalives are enabled for the connection). See also tcp.slowhz.

`tcp.keepintvl`
: Time after a keepalive probe is sent until, in the absence of any response, another probe is sent. See also tcp.slowhz.

`tcp.log_refused`
: If set to 1, refused TCP connections to the host will be logged.

`tcp.keepinit`
: Timeout in seconds during connection establishment.

`tcp.mss_ifmtu`
: If set to 1, TCP calculates the outgoing maximum segment size based on the MTU of the appropriate interface. If set to 0, it is calculated based on the greater of the MTU of the interface, and the largest (non-loopback) interface MTU on the system.

`tcp.mssdflt`
: The default maximum segment size both advertised to the peer and to use when either the peer does not advertise a maximum segment size to us during connection setup or Path MTU Discovery (`ip.mtudisc`) is disabled. Do not change this value unless you really know what you are doing.

`tcp.recvspace`
: The default TCP receive buffer size.

`tcp.rfc1323`
If set to 1, enables RFC 1323 extensions to TCP.

`tcp.rstppslimit`
The variable specifies the maximum number of outgoing TCP RST packets, per second. TCP RST packet that exceeded the value are subject to rate limitation and will not go out from the node. Negative value disables rate limitation.

`tcp.ident`
Return the user ID of a connected socket pair. (RFC1413 Identification Protocol lookups.)

`tcp.drop`
Drop a TCP socket pair connection.

`tcp.sack.enable`
If set to 1, enables RFC 2018 Selective ACKnowledgement.

`tcp.sack.globalholes`
Global number of TCP SACK holes.

`tcp.sack.globalmaxholes`
Global maximum number of TCP SACK holes.

`tcp.sack.maxholes`
Maximum number of TCP SACK holes allowed per connection.

`tcp.ecn.enable`
If set to 1, enables RFC 3168 Explicit Congestion Notification.

`tcp.ecn.maxretries`
Number of times to retry sending the ECN-setup packet.

`tcp.sendspace`
The default TCP send buffer size.

`tcp.slowhz`
The units for tcp.keepidle and tcp.keepintvl; those variables are in ticks of a clock that ticks tcp.slowhz times per second. (That is, their values must be divided by the tcp.slowhz value to get times in seconds.)

`tcp.syn_bucket_limit`
The maximum number of entries allowed per hash bucket in the TCP compressed state engine.

`tcp.syn_cache_limit`
The maximum number of entries allowed in the TCP compressed state engine.

`tcp.timestamps`
If rfc1323 is enabled, a value of 1 indicates RFC 1323 time stamp options, used for measuring TCP round trip times, are enabled.

`tcp.win_scale`
If rfc1323 is enabled, a value of 1 indicates RFC 1323 window scale options, for increasing the TCP window size, are enabled.

`tcp.congctl.available`
The available TCP congestion control algorithms.

`tcp.congctl.selected`
The currently selected TCP congestion control algorithm.

`tcp.abc.enable`
If set to 1, use RFC 3465 Appropriate Byte Counting (ABC). If set to 0, use traditional Packet Counting.

`tcp.abc.aggressive`
Choose the L parameter found in RFC 3465. L is the maximum cwnd increase for an ack during slow start. If set to 1, use L=2∗SMSS. If set to 0, use L=1∗SMSS. It has no effect unless tcp.abc.enable is set to 1.

`udp.checksum`
If set to 1, UDP checksums are being computed. Received non-zero UDP checksums are always checked. Disabling UDP checksums is strongly discouraged.

`udp.sendspace`
The default UDP send buffer size.

`udp.recvspace`
The default UDP receive buffer size.

For variables net.∗.ipsec, please refer to `ipsec`(4).

`net.inet6` (`PF_INET6`)
Get or set various global information about the IPv6 (Internet Protocol version 6). The third level name is the protocol. The fourth level name is the variable name. The currently defined protocols and names are:

Protocol name	Variable name	Type	Changeable
icmp6	errppslimit	integer	yes
icmp6	mtudisc_hiwat	integer	yes
icmp6	mtudisc_lowat	integer	yes
icmp6	nd6_debug	integer	yes
icmp6	nd6_delay	integer	yes
icmp6	nd6_maxnudhint	integer	yes
icmp6	nd6_mmaxtries	integer	yes
icmp6	nd6_prune	integer	yes
icmp6	nd6_umaxtries	integer	yes
icmp6	nd6_useloopback	integer	yes
icmp6	nodeinfo	integer	yes
icmp6	rediraccept	integer	yes
icmp6	redirtimeout	integer	yes
ip6	accept_rtadv	integer	yes
ip6	anonportmax	integer	yes
ip6	anonportmin	integer	yes
ip6	auto_flowlabel	integer	yes
ip6	dad_count	integer	yes
ip6	defmcasthlim	integer	yes
ip6	forwarding	integer	yes
ip6	gifhlim	integer	yes
ip6	hashsize	integer	yes
ip6	hlim	integer	yes

ip6	hdrnestlimit	integer	yes
ip6	kame_version	string	no
ip6	keepfaith	integer	yes
ip6	log_interval	integer	yes
ip6	lowportmax	integer	yes
ip6	lowportmin	integer	yes
ip6	maxflows	integer	yes
ip6	maxfragpackets	integer	yes
ip6	maxfrags	integer	yes
ip6	redirect	integer	yes
ip6	rr_prune	integer	yes
ip6	use_deprecated	integer	yes
ip6	v6only	integer	yes
udp6	do_loopback_cksum		integeryes
udp6	recvspace	integer	yes
udp6	sendspace	integer	yes

The variables are as follows:

`ip6.accept_rtadv`
: If set to non-zero, the node will accept ICMPv6 router advertisement packets and auto-configures address prefixes and default routers. The node must be a host (not a router) for the option to be meaningful.

`ip6.anonportmax`
: The highest port number to use for TCP and UDP ephemeral port allocation. This cannot be set to less than 1024 or greater than 65535, and must be greater than `ip6.anonportmin`.

`ip6.anonportmin`
: The lowest port number to use for TCP and UDP ephemeral port allocation. This cannot be set to less than 1024 or greater than 65535.

`ip6.auto_flowlabel`
: On connected transport protocol packets, fill IPv6 flowlabel field to help intermediate routers to identify packet flows.

`ip6.dad_count`
: The variable configures number of IPv6 DAD (duplicated address detection) probe packets. The packets will be generated when IPv6 interface addresses are configured.

`ip6.defmcasthlim`
: The default hop limit value for an IPv6 multicast packet sourced by the node. This value applies to all the transport protocols on top of IPv6. There are APIs to override the value, as documented in `ip6`(4).

`ip6.forwarding`
: If set to 1, enables IPv6 forwarding for the node, meaning that the node is acting as a router. If set to 0, disables IPv6 forwarding for the node, meaning that the node is acting as a host. IPv6 specification defines node behavior for “router” case and “host” case quite differently, and changing this variable during operation may cause serious trouble. It is recommended to configure the variable at bootstrap time, and bootstrap time only.

`ip6.gifhlim`
: The maximum hop limit value for an IPv6 packet generated by `gif`(4) tunnel interface.

`ip6.hdrnestlimit`
The number of IPv6 extension headers permitted on incoming IPv6 packets. If set to 0, the node will accept as many extension headers as possible.

`ip6.hashsize`
The size of IPv6 Fast Forward hash table. This value must be a power of 2 (64, 256...). A larger hash table size results in fewer collisions. Also see `ip6.maxflows`.

`ip6.hlim`
The default hop limit value for an IPv6 unicast packet sourced by the node. This value applies to all the transport protocols on top of IPv6. There are APIs to override the value, as documented in `ip6`(4).

`ip6.kame_version`
The string identifies the version of KAME IPv6 stack implemented in the kernel.

`ip6.keepfaith`
If set to non-zero, it enables "FAITH" TCP relay IPv6-to-IPv4 translator code in the kernel. Refer `faith`(4) and `faithd`(8) for detail.

`ip6.log_interval`
The variable controls amount of logs generated by IPv6 packet forwarding engine, by setting interval between log output (in seconds).

`ip6.lowportmax`
The highest port number to use for TCP and UDP reserved port allocation. This cannot be set to less than 0 or greater than 1024, and must be greater than `ip6.lowportmin`.

`ip6.lowportmin`
The lowest port number to use for TCP and UDP reserved port allocation. This cannot be set to less than 0 or greater than 1024, and must be smaller than `ip6.lowportmax`.

`ip6.maxflows`
IPv6 Fast Forwarding is enabled by default. If set to 0, IPv6 Fast Forwarding is disabled. `ip6.maxflows` controls the maximum amount of flows which can be created. The default value is 256.

`ip6.maxfragpackets`
The maximum number of fragmented packets the node will accept. 0 means that the node will not accept any fragmented packets. −1 means that the node will accept as many fragmented packets as it receives. The flag is provided basically for avoiding possible DoS attacks.

`ip6.maxfrags`
The maximum number of fragments the node will accept. 0 means that the node will not accept any fragments. −1 means that the node will accept as many fragments as it receives. The flag is provided basically for avoiding possible DoS attacks.

`ip6.redirect`
If set to 1, ICMPv6 redirects may be sent by the node. This option is ignored unless the node is routing IP packets, and should normally be enabled on all systems.

`ip6.rr_prune`
The variable specifies interval between IPv6 router renumbering prefix babysitting, in seconds.

`ip6.use_deprecated`
The variable controls use of deprecated address, specified in RFC 2462 5.5.4.

`ip6.v6only`
The variable specifies initial value for `IPV6_V6ONLY` socket option for `AF_INET6` socket. Please refer to `ip6`(4) for detail.

`icmp6.errppslimit`
The variable specifies the maximum number of outgoing ICMPv6 error messages, per second. ICMPv6 error messages that exceeded the value are subject to rate limitation and will not go out from the node. Negative value disables rate limitation.

`icmp6.mtudisc_hiwat`

`icmp6.mtudisc_lowat`
The variables define the maximum number of routing table entries, created due to path MTU discovery (prevents denial-of-service attacks with ICMPv6 too big messages). When IPv6 path MTU discovery happens, we keep path MTU information into the routing table. If the number of routing table entries exceed the value, the kernel will not attempt to keep the path MTU information. `icmp6.mtudisc_hiwat` is used when we have verified ICMPv6 too big messages. `icmp6.mtudisc_lowat` is used when we have unverified ICMPv6 too big messages. Verification is performed by using address/port pairs kept in connected pcbs. Negative value disables the upper limit.

`icmp6.nd6_debug`
If set to non-zero, kernel IPv6 neighbor discovery code will generate debugging messages. The debug outputs are useful to diagnose IPv6 interoperability issues. The flag must be set to 0 for normal operation.

`icmp6.nd6_delay`
The variable specifies `DELAY_FIRST_PROBE_TIME` timing constant in IPv6 neighbor discovery specification (RFC 2461), in seconds.

`icmp6.nd6_maxnudhint`
IPv6 neighbor discovery permits upper layer protocols to supply reachability hints, to avoid unnecessary neighbor discovery exchanges. The variable defines the number of consecutive hints the neighbor discovery layer will take. For example, by setting the variable to 3, neighbor discovery layer will take 3 consecutive hints in maximum. After receiving 3 hints, neighbor discovery layer will perform normal neighbor discovery process.

`icmp6.nd6_mmaxtries`
The variable specifies `MAX_MULTICAST_SOLICIT` constant in IPv6 neighbor discovery specification (RFC 2461).

`icmp6.nd6_prune`
The variable specifies interval between IPv6 neighbor cache babysitting, in seconds.

`icmp6.nd6_umaxtries`
The variable specifies `MAX_UNICAST_SOLICIT` constant in IPv6 neighbor discovery specification (RFC 2461).

`icmp6.nd6_useloopback`
If set to non-zero, kernel IPv6 stack will use loopback interface for local traffic.

`icmp6.nodeinfo`
The variable enables responses to ICMPv6 node information queries. If you set the variable to 0, responses will not be generated for ICMPv6 node information queries. Since

node information queries can have a security impact, it is possible to fine tune which responses should be answered. Two separate bits can be set.

1 Respond to ICMPv6 FQDN queries, e.g. `ping6 -w`.

2 Respond to ICMPv6 node addresses queries, e.g. `ping6 -a`.

`icmp6.rediraccept`
If set to non-zero, the host will accept ICMPv6 redirect packets. Note that IPv6 routers will never accept ICMPv6 redirect packets, and the variable is meaningful on IPv6 hosts (non-router) only.

`icmp6.redirtimeout`
The variable specifies lifetime of routing entries generated by incoming ICMPv6 redirect.

`udp6.do_loopback_cksum`
Perform UDP checksum on loopback.

`udp6.recvspace`
Default UDP receive buffer size.

`udp6.sendspace`
Default UDP send buffer size.

We reuse net.*.tcp for TCP over IPv6, and therefore we do not have variables net.*.tcp6. Variables net.inet6.udp6 have identical meaning to net.inet.udp. Please refer to `PF_INET` section above. For variables net.*.ipsec6, please refer to `ipsec`(4).

`net.key` (`PF_KEY`)
Get or set various global information about the IPsec key management. The third level name is the variable name. The currently defined variable and names are:

Variable name	Type	Changeable
debug	integer	yes
spi_try	integer	yes
spi_min_value	integer	yes
spi_max_value	integer	yes
larval_lifetime	integer	yes
blockacq_count	integer	yes
blockacq_lifetime	integer	yes
esp_keymin	integer	yes
esp_auth	integer	yes
ah_keymin	integer	yes

The variables are as follows:

`debug` Turn on debugging message from within the kernel. The value is a bitmap, as defined in `/usr/include/netkey/key_debug.h`.

`spi_try`
The number of times the kernel will try to obtain an unique SPI when it generates it from random number generator.

`spi_min_value`
Minimum SPI value when generating it within the kernel.

`spi_max_value`
Maximum SPI value when generating it within the kernel.

`larval_lifetime`
Lifetime for LARVAL SAD entries, in seconds.

`blockacq_count`
Number of ACQUIRE PF_KEY messages to be blocked after an ACQUIRE message. It avoids flood of ACQUIRE PF_KEY from being sent from the kernel to the key management daemon.

`blockacq_lifetime`
Lifetime of ACQUIRE PF_KEY message.

`esp_keymin`
Minimum ESP key length, in bits. The value is used when the kernel creates proposal payload on ACQUIRE PF_KEY message.

`esp_auth`
Whether ESP authentication should be used or not. Non-zero value indicates that ESP authentication should be used. The value is used when the kernel creates proposal payload on ACQUIRE PF_KEY message.

`ah_keymin`
Minimum AH key length, in bits, The value is used when the kernel creates proposal payload on ACQUIRE PF_KEY message.

The proc.∗ subtree

The string and integer information available for the `proc` level is detailed below. The changeable column shows whether a process with appropriate privilege may change the value. These values are per-process, and as such may change from one process to another. When a process is created, the default values are inherited from its parent. When a set-user-ID or set-group-ID binary is executed, the value of PROC_PID_CORENAME is reset to the system default value. The second level name is either the magic value PROC_CURPROC, which points to the current process, or the PID of the target process.

Third level name	Type	Changeable
proc.pid.corename	string	yes
proc.pid.rlimit	node	not applicable
proc.pid.stopfork	int	yes
proc.pid.stopexec	int	yes
proc.pid.stopexit	int	yes

`proc.pid.corename` (`PROC_PID_CORENAME`)
The template used for the core dump file name (see `core`(5) for details). The base name must either be **`core`** or end with the suffix ‘‘.core’’ (the super-user may set arbitrary names). By default it points to KERN_DEFCORENAME.

`proc.pid.rlimit` (`PROC_PID_LIMIT`)
Return resources limits, as defined for the `getrlimit`(2) and `setrlimit`(2) system calls. The fourth level name is one of:

`proc.pid.rlimit.cputime` (`PROC_PID_LIMIT_CPU`)
The maximum amount of CPU time (in seconds) to be used by each process.

`proc.pid.rlimit.filesize` (`PROC_PID_LIMIT_FSIZE`)
The largest size (in bytes) file that may be created.

`proc.pid.rlimit.datasize` (`PROC_PID_LIMIT_DATA`)
The maximum size (in bytes) of the data segment for a process; this defines how far a program may extend its break with the `sbrk`(2) system call.

`proc.pid.rlimit.stacksize` (`PROC_PID_LIMIT_STACK`)
The maximum size (in bytes) of the stack segment for a process; this defines how far a program's stack segment may be extended. Stack extension is performed automatically by the system.

`proc.pid.rlimit.coredumpsize` (`PROC_PID_LIMIT_CORE`)
The largest size (in bytes) `core` file that may be created.

`proc.pid.rlimit.memoryuse` (`PROC_PID_LIMIT_RSS`)
The maximum size (in bytes) to which a process's resident set size may grow. This imposes a limit on the amount of physical memory to be given to a process; if memory is tight, the system will prefer to take memory from processes that are exceeding their declared resident set size.

`proc.pid.rlimit.memorylocked` (`PROC_PID_LIMIT_MEMLOCK`)
The maximum size (in bytes) which a process may lock into memory using the `mlock`(2) function.

`proc.pid.rlimit.maxproc` (`PROC_PID_LIMIT_NPROC`)
The maximum number of simultaneous processes for this user id.

`proc.pid.rlimit.descriptors` (`PROC_PID_LIMIT_NOFILE`)
The maximum number of open files for this process.

The fifth level name is one of `soft` (`PROC_PID_LIMIT_TYPE_SOFT`) or `hard` (`PROC_PID_LIMIT_TYPE_HARD`), to select respectively the soft or hard limit. Both are of type integer.

`proc.pid.stopfork` (`PROC_PID_STOPFORK`)
If non zero, the process' children will be stopped after `fork`(2) calls. The children is created in the SSTOP state and is never scheduled for running before being stopped. This feature helps attaching a process with a debugger such as `gdb`(1) before it had the opportunity to actually do anything.

This value is inherited by the process's children, and it also apply to emulation specific system calls that fork a new process, such as **`sproc`**() or **`clone`**().

`proc.pid.stopexec` (`PROC_PID_STOPEXEC`)
If non zero, the process will be stopped on next `exec`(3) call. The process created by `exec`(3) is created in the SSTOP state and is never scheduled for running before being stopped. This feature helps attaching a process with a debugger such as `gdb`(1) before it had the opportunity to actually do anything.

This value is inherited by the process's children.

`proc.pid.stopexit` (`PROC_PID_STOPEXIT`)
If non zero, the process will be stopped on when it has cause to exit, either by way of calling `exit`(3), `_exit`(2), or by the receipt of a specific signal. The process is stopped before any of its resources or vm space is released allowing examination of the termination state of a process before it disappears. This feature can be used to examine the final conditions of the process's vmspace

via pmap(1) or its resource settings with sysctl(8) before it disappears.

This value is also inherited by the process's children.

The user.* subtree (CTL_USER)

The string and integer information available for the user level is detailed below. The changeable column shows whether a process with appropriate privilege may change the value.

Second level name	Type	Changeable
user.atexit_max	integer	no
user.bc_base_max	integer	no
user.bc_dim_max	integer	no
user.bc_scale_max	integer	no
user.bc_string_max	integer	no
user.coll_weights_max	integer	no
user.cs_path	string	no
user.expr_nest_max	integer	no
user.line_max	integer	no
user.posix2_c_bind	integer	no
user.posix2_c_dev	integer	no
user.posix2_char_term	integer	no
user.posix2_fort_dev	integer	no
user.posix2_fort_run	integer	no
user.posix2_localedef	integer	no
user.posix2_sw_dev	integer	no
user.posix2_upe	integer	no
user.posix2_version	integer	no
user.re_dup_max	integer	no
user.stream_max	integer	no
user.stream_max	integer	no
user.tzname_max	integer	no

user.atexit_max (USER_ATEXIT_MAX)
: The maximum number of functions that may be registered with atexit(3).

user.bc_base_max (USER_BC_BASE_MAX)
: The maximum ibase/obase values in the bc(1) utility.

user.bc_dim_max (USER_BC_DIM_MAX)
: The maximum array size in the bc(1) utility.

user.bc_scale_max (USER_BC_SCALE_MAX)
: The maximum scale value in the bc(1) utility.

user.bc_string_max (USER_BC_STRING_MAX)
: The maximum string length in the bc(1) utility.

user.coll_weights_max (USER_COLL_WEIGHTS_MAX)
: The maximum number of weights that can be assigned to any entry of the LC_COLLATE order keyword in the locale definition file.

user.cs_path (USER_CS_PATH)
: Return a value for the PATH environment variable that finds all the standard utilities.

user.expr_nest_max (USER_EXPR_NEST_MAX)
: The maximum number of expressions that can be nested within parenthesis by the expr(1) utility.

`user.line_max` (`USER_LINE_MAX`)
The maximum length in bytes of a text-processing utility's input line.

`user.posix2_char_term` (`USER_POSIX2_CHAR_TERM`)
Return 1 if the system supports at least one terminal type capable of all operations described in POSIX 1003.2, otherwise 0.

`user.posix2_c_bind` (`USER_POSIX2_C_BIND`)
Return 1 if the system's C-language development facilities support the C-Language Bindings Option, otherwise 0.

`user.posix2_c_dev` (`USER_POSIX2_C_DEV`)
Return 1 if the system supports the C-Language Development Utilities Option, otherwise 0.

`user.posix2_fort_dev` (`USER_POSIX2_FORT_DEV`)
Return 1 if the system supports the FORTRAN Development Utilities Option, otherwise 0.

`user.posix2_fort_run` (`USER_POSIX2_FORT_RUN`)
Return 1 if the system supports the FORTRAN Runtime Utilities Option, otherwise 0.

`user.posix2_localedef` (`USER_POSIX2_LOCALEDEF`)
Return 1 if the system supports the creation of locales, otherwise 0.

`user.posix2_sw_dev` (`USER_POSIX2_SW_DEV`)
Return 1 if the system supports the Software Development Utilities Option, otherwise 0.

`user.posix2_upe` (`USER_POSIX2_UPE`)
Return 1 if the system supports the User Portability Utilities Option, otherwise 0.

`user.posix2_version` (`USER_POSIX2_VERSION`)
The version of POSIX 1003.2 with which the system attempts to comply.

`user.re_dup_max` (`USER_RE_DUP_MAX`)
The maximum number of repeated occurrences of a regular expression permitted when using interval notation.

`user.stream_max` (`USER_STREAM_MAX`)
The minimum maximum number of streams that a process may have open at any one time.

`user.tzname_max` (`USER_TZNAME_MAX`)
The minimum maximum number of types supported for the name of a timezone.

The vm.* subtree (CTL_VM)
The string and integer information available for the `vm` level is detailed below. The changeable column shows whether a process with appropriate privilege may change the value.

Second level name	Type	Changeable
vm.anonmax	int	yes
vm.anonmin	int	yes
vm.bufcache	int	yes
vm.bufmem	int	no
vm.bufmem_hiwater	int	yes
vm.bufmem_lowater	int	yes
vm.execmax	int	yes
vm.execmin	int	yes
vm.filemax	int	yes

vm.filemin	int	yes
vm.loadavg	struct loadavg	no
vm.maxslp	int	no
vm.nkmempages	int	no
vm.uspace	int	no
vm.uvmexp	struct uvmexp	no
vm.uvmexp2	struct uvmexp_sysctl	no
vm.vmmeter	struct vmtotal	no

`vm.anonmax` (`VM_ANONMAX`)
The percentage of physical memory which will be reclaimed from other types of memory usage to store anonymous application data.

`vm.anonmin` (`VM_ANONMIN`)
The percentage of physical memory which will be always be available for anonymous application data.

`vm.bufcache` (`VM_BUFCACHE`)
The percentage of physical memory which will be available for the buffer cache.

`vm.bufmem` (`VM_BUFMEM`)
The amount of kernel memory that is being used by the buffer cache.

`vm.bufmem_lowater` (`VM_BUFMEM_LOWATER`)
The minimum amount of kernel memory to reserve for the buffer cache.

`vm.bufmem_hiwater` (`VM_BUFMEM_HIWATER`)
The maximum amount of kernel memory to be used for the buffer cache.

`vm.execmax` (`VM_EXECMAX`)
The percentage of physical memory which will be reclaimed from other types of memory usage to store cached executable data.

`vm.execmin` (`VM_EXECMIN`)
The percentage of physical memory which will be always be available for cached executable data.

`vm.filemax` (`VM_FILEMAX`)
The percentage of physical memory which will be reclaimed from other types of memory usage to store cached file data.

`vm.filemin` (`VM_FILEMIN`)
The percentage of physical memory which will be always be available for cached file data.

`vm.loadavg` (`VM_LOADAVG`)
Return the load average history. The returned data consists of a *struct loadavg*.

`vm.maxslp` (`VM_MAXSLP`)
The value of the maxslp kernel global variable.

`vm.vmmeter` (`VM_METER`)
Return system wide virtual memory statistics. The returned data consists of a *struct vmtotal*.

`vm.uspace` (`VM_USPACE`)
The number of bytes allocated for each kernel stack.

`vm.uvmexp` (`VM_UVMEXP`)
Return system wide virtual memory statistics. The returned data consists of a *struct uvmexp*.

`vm.uvmexp2` (`VM_UVMEXP2`)
Return system wide virtual memory statistics. The returned data consists of a *struct uvmexp_sysctl*.

The ddb.* subtree (**CTL_DDB**)
The integer information available for the `ddb` level is detailed below. The changeable column shows whether a process with appropriate privilege may change the value.

Second level name	Type	Changeable
ddb.radix	integer	yes
ddb.maxoff	integer	yes
ddb.lines	integer	yes
ddb.tabstops	integer	yes
ddb.onpanic	integer	yes
ddb.fromconsole	integer	yes

`ddb.radix` (`DBCTL_RADIX`)
The input and output radix.

`ddb.maxoff` (`DBCTL_MAXOFF`)
The maximum symbol offset.

`ddb.lines` (`DBCTL_LINES`)
Number of display lines.

`ddb.tabstops` (`DBCTL_TABSTOPS`)
Tab width.

`ddb.onpanic` (`DBCTL_ONPANIC`)
If non-zero, DDB will be entered when the kernel panics.

`ddb.fromconsole` (`DBCTL_FROMCONSOLE`)
If not zero, DDB may be entered by sending a break on a serial console or by a special key sequence on a graphics console.

These MIB nodes are also available as variables from within the DDB. See `ddb`(4) for more details.

The security.* subtree (**CTL_SECURITY**)
The `security` level contains various security-related settings for the system. Available settings are detailed below.

`security.curtain`
If non-zero, will filter return objects according to the user-id requesting information about them, preventing from users any access to objects they don't own.

At the moment, it affects `ps`(1), `netstat`(1) (for `PF_INET`, `PF_INET6`, and `PF_UNIX` PCBs), and `w`(1).

`security.models`
NetBSD supports pluggable security models. Every security model used, whether if loaded as an LKM or built with the system, is required to add an entry to this node with at least one element, "name", indicating the name of the security model.

In addition to the name, any settings and other information private to the security model will be available under this node. See `secmodel`(9) for more information.

`security.pax`
Settings for PaX -- exploit mitigation features. For more information on any of the PaX features, please see `paxctl`(8) and `security`(8).

`security.pax.aslr.enable`
Enable PaX ASLR (Address Space Layout Randomization).

The value of this knob must be non-zero for PaX ASLR to be enabled, even if a program is set to explicit enable.

`security.pax.aslr.global`
Specifies the default global policy for programs without an explicit enable/disable flag.

When non-zero, all programs will get PaX ASLR, except those exempted with `paxctl`(8). Otherwise, all programs will not get PaX ASLR, except those specifically marked as such with `paxctl`(8).

`security.pax.mprotect.enable`
Enable PaX MPROTECT restrictions.

These are `mprotect`(2) restrictions to better enforce a WˆX policy. The value of this knob must be non-zero for PaX MPROTECT to be enabled, even if a program is set to explicit enable.

`security.pax.mprotect.global`
Specifies the default global policy for programs without an explicit enable/disable flag.

When non-zero, all programs will get the PaX MPROTECT restrictions, except those exempted with `paxctl`(8). Otherwise, all programs will not get the PaX MPROTECT restrictions, except those specifically marked as such with `paxctl`(8).

`security.pax.segvguard.enable`
Enable PaX Segvguard.

PaX Segvguard can detect and prevent certain exploitation attempts, where an attacker may try for example to brute-force function return addresses of respawning daemons.

Note: The NetBSD interface and implementation of the Segvguard is still experimental, and may change in future releases.

`security.pax.segvguard.global`
Specifies the default global policy for programs without an explicit enable/disable flag.

When non-zero, all programs will get the PaX Segvguard, except those exempted with `paxctl`(8). Otherwise, no program will get the PaX Segvguard restrictions, except those specifically marked as such with `paxctl`(8).

`security.pax.segvguard.expiry_timeout`
If the max number was not reached within this timeout (in seconds), the entry will expire.

`security.pax.segvguard.suspend_timeout`
Number of seconds to suspend a user from running a faulting program when the limit was exceeded.

`security.pax.segvguard.max_crashes`
Max number of segfaults a program can receive before suspension.

The vendor.* subtree (CTL_VENDOR)

The `vendor` toplevel name is reserved to be used by vendors who wish to have their own private MIB tree. Intended use is to store values under "vendor.<yourname>.*".

SEE ALSO

`sysctl`(3), `ipsec`(4), `tcp`(4), `security`(8), `sysctl`(8)

HISTORY

The **`sysctl`** variables first appeared in 4.4BSD.