NAME

intro – introduction to miscellany

DESCRIPTION

This section describes miscellaneous facilities such as macro packages, character set tables, etc.

NAME

ascii − map of ASCII character set

SYNOPSIS

cat /usr/pub/ascii

DESCRIPTION

Ascii is a map of the ASCII character set, to be printed as needed. It contains:

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

FILES

/usr/pub/ascii

NAME

environ − user environment

DESCRIPTION

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

PATH The sequence of directory prefixes that *sh*(1), *time*(1), *nice*(1), *nohup*(1), etc., apply in searching for a file known by an incomplete path name. The prefixes are separated by colons (:). *Login*(1) sets **PATH=:/bin:/usr/bin**.

SHELL
The name of the shell which is executed upon escaping from some interactive commands (such as *ed*(1) and *mail*(1)) and used by the *execvp*(3C) and *system*(3S) library routines.

HOME
Name of the user's login directory, set by *login*(1) from the password file *passwd*(5).

TERM
The kind of terminal for which output is to be prepared. This information is used by commands, such as *mm*(1) or *plot*(1G), which may exploit special capabilities of that terminal.

Further names may be placed in the environment by the *export* command and "name=value" arguments in *sh*(1), or by *exec*(2). It is unwise to conflict with certain shell variables that are frequently exported by **.profile** files: **MAIL, PS1, PS2, IFS.**

SEE ALSO

env(1), login(1), sh(1), exec(2), getenv(3C), profile(5)

NAME

greek − graphics for extended TTY-37 type-box

SYNOPSIS

cat /usr/pub/greek [| **greek** −Tterminal]

DESCRIPTION

Greek gives the mapping from ASCII to the "shift out" graphics in effect between SO and SI on TELETYPE® Model 37 terminals equipped with a 128-character type-box. These are the default greek characters produced by *nroff*(1). The filters of *greek*(1) attempt to print them on various other terminals. The file contains:

alpha	α	A	beta	β	B	gamma	γ	\
GAMMA	Γ	G	delta	δ	D	DELTA	Δ	W
epsilon	ϵ	S	zeta	ζ	Q	eta	η	N
THETA	Θ	T	theta	θ	O	lambda	λ	L
LAMBDA	Λ	E	mu	μ	M	nu	ν	@
xi	ξ	X	pi	π	J	PI	Π	P
rho	ρ	K	sigma	σ	Y	SIGMA	Σ	R
tau	τ	I	phi	ϕ	U	PHI	Φ	F
psi	ψ	V	PSI	Ψ	H	omega	ω	C
OMEGA	Ω	Z	nabla	∇	[	not	$\neg$	_
partial	∂	]	integral	$\int$	^			

SEE ALSO

greek(1), nroff(1).

NAME

regexp − regular expression compile and match routines

SYNOPSIS

```
#define INIT <declarations>
#define GETC() <getc code>
#define PEEKC() <peekc code>
#define UNGETC(c) <ungetc code>
#define RETURN(pointer) <return code>
#define ERROR(val) <error code>

#include <regexp.h>

char *compile(instring, expbuf, endbuf, eof)
char *instring, *expbuf, *endbuf;

int step(string, expbuf)
char *string, *expbuf;
```

DESCRIPTION

This page describes general purpose regular expression matching routines in the form of *ed*(1), defined in **/usr/include/regexp.h**. Programs such as *ed*(1), *sed*(1), *grep*(1), *bs*(1), *expr*(1), etc., which perform regular expression matching use this source file. In this way, only this file need be changed to maintain regular expression compatibility.

The interface to this file is unpleasantly complex. Programs that include this file must have the following five macros declared before the "#include <regexp.h>" statement. These macros are used by the *compile* routine.

GETC()	Return the value of the next character in the regular expression pattern. Successive calls to **GETC()** should return successive characters of the regular expression.
PEEKC()	Return the next character in the regular expression. Successive calls to **PEEKC()** should return the same character (which should also be the next character returned by **GETC()**).
UNGETC(*c***)**	Cause the argument *c* to be returned by the next call to **GETC()** (and **PEEKC()**). No more that one character of pushback is ever needed and this character is guaranteed to be the last character read by **GETC()**. The value of the macro **UNGETC(***c***)** is always ignored.
RETURN(*pointer***)**	This macro is used on normal exit of the *compile* routine. The value of the argument *pointer* is a pointer to the character after the last character of the compiled regular expression. This is useful to programs which have memory allocation to manage.
ERROR(*val***)**	This is the abnormal return from the *compile* routine. The argument *val* is an error number (see table below for meanings). This call should never return.

ERROR	*MEANING*
11	Range endpoint too large.
16	Bad number.
25	"\digit" out of range.
36	Illegal or missing delimiter.
41	No remembered search string.
42	\(\) imbalance.
43	Too many \(.
44	More than 2 numbers given in \{ \}.
45	} expected after \.
46	First number exceeds second in \{ \}.
49	[ ] imbalance.
50	Regular expression overflow.

The syntax of the *compile* routine is as follows:

compile(instring, expbuf, endbuf, eof)

The first parameter "instring" is never used explicitly by the *compile* routine but is useful for programs that pass down different pointers to input characters. It is sometimes used in the INIT declaration (see below). Programs which call functions to input characters or have characters in an external array can pass down a value of ((char *) 0) for this parameter.

The next parameter "expbuf" is a character pointer. It points to the place where the compiled regular expression will be placed.

The parameter "endbuf" is one more that the highest address that the compiled regular expression may be placed. If the compiled expression cannot fit in (endbuf−expbuf) bytes, a call to ERROR(50) is made.

The parameter "eof" is the character which marks the end of the regular expression. For example, in *ed*(1), this character is usually a /.

Each programs that includes this file must have a **#define** statement for INIT. This definition will be placed right after the declaration for the function *compile* and the opening curly brace ({). It is used for dependent declarations and initializations. Most often it is used to set a register variable to point the beginning of the regular expression so that this register variable can be used in the declarations for GETC(), PEE() and UNGETC(). Otherwise it can be used to declare external variables that might be used by GETC(), PEEKC() and UNGETC(). See the example below of the declarations taken from *grep*(1).

There are other functions in this file which perform actual regular expression matching, one of which is the function *step*. The call to *step* is as follows:

step(string, expbuf)

The first parameter to *step* is a pointer to a string of characters to be checked for a match. This string should be null terminated.

The second parameter "expbuf" is the compiled regular expression which was obtained by a call of the function *compile*.

The function *step* returns one, if the given string matches the regular expression, and zero if the expressions do not match. If there is a match, two external character pointers are set as a side effect to the call to *step*. The variable set in *step* is "loc1". This is a pointer to the first character that matched the regular expression. The variable "loc2", which is set by the function *advance*, points the character after the last character that matches the regular expression. Thus if the regular expression matches the entire line, loc1 will point to the first character of

"string" and "loc2" will point to the null at the end of "string".

Step uses the external variable "circf" which is set by *compile* if the regular expression begins with ^. If this is set then *step* will only try to match the regular expression to the beginning of the string. If more than one regular expression is to be compiled before the the first is executed the value of "circf" should be saved for each compiled expression and "circf" should be set to that saved value before each call to *step*.

The function *advance* is called from *step* with the same arguments as *step*. The purpose of *step* is to step through the "string" argument and call *advance* until *advance* returns a one indicating a match or until the end of "string" is reached. If one wants to constrain "string" to the beginning of the line in all cases, *step* need not be called, simply call *advance*.

When *advance* encounters a * or \{ \} sequence in the regular expression it will advance its pointer to the string to be matched as far as possible and will recursively call itself trying to match the rest of the string to the rest of the regular expression. As long as there is no match, *advance* will back up along the string until it finds a match or reaches the point in the string that initially matched the * or \{ \}. It is sometimes desirable to stop this backing up before the initial point in the string is reached. If the external character pointer "locs" is equal to the point in the string at sometime during the backing up process, *advance* will break out of the loop that backs up and will return zero. This is used be *ed*(1) and *sed*(1) for substitutions done globally (not just the first occurrence, but the whole line) so, for example, expressions like "s/y*//g" do not loop forever.

The routines *ecmp* and *getrange* are trivial and are called by the routines previously mentioned.

EXAMPLES

The following is an example of how the regular expression macros and calls look from *grep*(1):

```
#define INIT        register char *sp = instring;  /* First arg points to RE string */
#define GETC()      (*sp++)
#define PEEKC()     (*sp)
#define UNGETC(c)   (--sp)
#define RETURN(c)   return;
#define ERROR(c)    regerr()

#include <regexp.h>
...
        compile(*argv, expbuf, &expbuf[ESIZE], '\0');
...
        if(step(linebuf, expbuf))
                succeed();
```

FILES

/usr/include/regexp.h

SEE ALSO

ed(1), grep(1), sed(1).

BUGS

The handling of "circf" is kludgy.
The routine *ecmp* is equivalent to the Standard I/O routine *strncmp* and should be replaced by that routine.
The actual code is probably easier to understand than this manual page.

NAME

stat − data returned by stat system call

SYNOPSIS

#include <types.h>
#include <stat.h>

DESCRIPTION

The system calls *stat* and *fstat*(2) return data whose structure is defined by this include file. The encoding of the field *st_mode* is defined in this file also.

```
/*      @(#)stat.h3.1       */
struct  stat
{
        dev_t       st_dev;
        ino_t       st_ino;
        int         st_mode;
        int         st_nlink;
        int         st_uid;
        int         st_gid;
        dev_t       st_rdev;
        off_t       st_size;
        time_t      st_atime;
        time_t      st_mtime;
        time_t      st_ctime;
};

#define S_IFMT   0170000          /* type of file */
#define          S_IFDIR  0040000 /* directory */
#define          S_IFCHR  0020000 /* character special */
#define          S_IFBLK  0060000 /* block special */
#define          S_IFREG  0100000 /* regular */
#define          S_IFMPC  0030000 /* multiplexed char special */
#define          S_IFMPB  0070000 /* multiplexed block special */
#define S_ISUID  0004000          /* set user id on execution */
#define S_ISGID  0002000          /* set group id on execution */
#define S_ISVTX  0001000          /* save swapped text even after use */
#define S_IREAD  0000400          /* read permission, owner */
#define S_IWRITE 0000200          /* write permission, owner */
#define S_IEXEC  0000100          /* execute/search permission, owner */
```

FILES

/usr/include/sys/stat.h

SEE ALSO

stat(2)

NAME

types − primitive system data types

SYNOPSIS

#include <sys/types.h>

DESCRIPTION

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

```
/*      @(#)/usr/src/ucb/sys/types.h  3.1      */
/*
 * Typedefs
 */
typedef struct { int r[1]; } *      physadr;
typedef         unsigned            daddr_t;
typedef         char *              caddr_t;
typedef         unsigned int        ino_t;
typedef         long                time_t;
typedef         int                 label_t[6];
typedef         int                 dev_t;
typedef         long                off_t;
typedef         long                paddr_t;
typedef         unsigned int        spcnt_t;
```

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

SEE ALSO

fs(5)