

builtin [-ds] [-f file] [name ...]
with no arguments, print all built-in commands

-d delete built-in *name*

-f load new built-ins from shared library *file*

-s only print special built-ins

cd [-LP] [dir]
change current directory to *dir* (\$HOME default). Directory path search using value of \$CDPATH

-L use logical path for cd .., \$PWD (default)

-P use physical path for cd .., \$PWD

cd [-LP] -
change current directory to \$OLDPWD

cd [-LP] *old new*
substitute *new* for *old* in current directory; change to result

command [-pV] *name* [arg ...]
without -v or -V, execute *name* with arguments *arg*; failure in a built-in does not exit shell

-p use a default search path, not \$PATH

-v behave like whence

-V behave like whence -v

continue [n]
do next iteration of enclosing for, while, until, or select loop. If *n* supplied, iterate *n*th enclosing loop

disown [job ...]
when a login shell terminates, do not send a SIGHUP to *job*, or to all active jobs if none listed

echo [words]
echo *words*; expands \-escapes on some systems (print and printf are more portable)

eval [words]
evaluate *words* and execute result

exec [-a name] [-c] [words]
execute *words* in place of shell. If redirections only, change shell's open files

-a use *name* for argv[0]

-c clear the environment first

exit [n]
exit with return value *n*; use \$? if no *n*

export [-p] [name=value] ...
export variables to environment of commands. With no arguments, print names and values of exported variables

-p print export before each variable

false
do nothing, exit 1

fg [jobid]
put *jobid* in the foreground

getconf [name [pathname]]
print configuration parameters for *name* and *pathname*; parameters are defined by POSIX

getopts [-a name] *optstring name* [arg ...]
parse parameters and options, use "\$@" if no args

-a use *name* for error and usage messages

hist [-e editor][-lnr][-N num][first [last]]
print a range of commands from *first* to *last* from last \$HISTSIZE commands; negative numbers are relative to current command

-e run *editor* if supplied, if not, run \${HISTEDIT:-\$FCEDIT} (default /bin/ed) on commands; execute result(s) if file with commands was updated

-l list on standard output instead of editing

-n don't print line numbers

-Nt *num* is relative to current command

-r reverse order of commands

hist -s [old=new] [command]
substitute *new* for *old* in *command* (or last command if no *command*) and execute the result

jobs [-lnp] [jobid ...]
list information about jobs

-l also list process id

-n list jobs whose status has changed

-p only list process groups

kill [-n *sigum*] [-s *signame*] *jobid* ...
send SIGTERM or given signal to named *jobids*. Signals are names listed in /usr/include/signal.h without the prefix "SIG." Stopped jobs get a SIGCONT first if *sig* is either SIGTERM or SIGHUP

-n send signal number *sigum*

-s send signal named *signame*

kill -l [sigs ...]
list signal names and/or numbers

let *arg* ...
evaluate each *arg* as an arithmetic expression; exit 0 if last expression was nonzero, 1 otherwise (see *Arithmetic Evaluation*)

newgrp [words]
same as exec newgrp *words*

print [-f format] [-enprRsu[n]] [arg ...]
if no flags, or just - or --, act like echo (however, the single - is obsolete)

-e† interpret escape sequences (the default)

-f print as if by printf command; ignore -n, -r, and -R options

-n don't add final newline

-p print to stdin of coprocess

-r ignore echo escape conventions

-R like -r, also ignore other opts except -n

-s print to history file

-u print to file descriptor *n* (fd1 default)

printf *format* [arg ...]
print output like ANSI C printf, with extensions

pwd [-LP]
print working directory name

-L print logical path (default)

-P print physical path

read [-A name] [-d delim] [-n count] [-prsu[n]] [-t timeout] [name?prompt] [names ...]
read stdin and assign to *names*; \$IFS splits input. ? in first *name* uses rest of *name* as prompt to stderr. REPLY is set if no *name* given. Exit 0 unless end-of-file encountered or a timeout occurs; \$TMOUT applies if no -t

-A read words into indexed array *name*

-d read to *delim* instead of newline

-n† read at most *count* bytes

-p read from stdout of coprocess

-r \ at end of line does not do line continuation

-s save input in history file

-t wait *timeout* seconds when reading from a terminal or pipe

-u read from file descriptor *n* (fd0 default)

readonly [-p] [name=value] ...
mark *names* read-only; print list if no *names*

-p print readonly before each variable

return [n]
exit function with return value. With no *n*, return status of last command; if not in function or in . script, like exit

set [-flags] [-o option] [words]
set flags and options (see *Options To set*)

words set positional parameters

set [+flags] [+o option] [words]
unset flags and options

shift [n]
rename positional parameters; \$1=\$n+1 ...
n defaults to 1; *n* can be an arithmetic expression

sleep *seconds*
stop execution for given number of whole or fractional seconds

test
evaluate conditional expressions (see *Conditional Expressions*)

trap [-p] [word] [sigs]
execute *word* if signal in *sigs* received *sigs* are numbers or signal names without "SIG". With no *word* or *sigs*, print traps. With no *word*, reset *sigs* to entry defaults. If *word* is null string, ignore *sigs*.

-p print traps with quoting

true
do nothing, exit 0

typeset -f[tux] [name ...]
each *name* is a function

-t turn on execution tracing

-u function is undefined, use \$FPATH and shared directories in \$PATH to find it

-x obsolete: no effect

typeset [+AEFHILlnprRtuxZ[n]] [name=value]]
set attributes and values of variables; inside functions, create new copies of the variables; using + instead of - turns attributes off; with no names or attributes, print all variable names and attributes

-A *name* is an associative array

-E *name* is a floating-point number; *n* is number of significant figures. Use %g to expand

-F *name* is a floating-point number; *n* is number of decimal places. Use %f to expand

-H Unix to host file-name mapping on non-Unix systems

-i *name* is an integer; *n* is output base, otherwise base is 10

-l convert all uppercase characters to lower case; turn off -u

-L left justify; *n* is width, or set from first assignment; turn off -R

-n *name* is a nameref variable

-p print typeset commands to recreate variables with same attributes

-r mark *names* readonly

-R right justify; *n* is width, or set from first assignment; turn off -L

-t tag the variables (obsolete)

-u convert all lowercase characters to uppercase; turn off -l

-ui† *name* is an unsigned integer; *n* is output base, otherwise base is 10

-x mark *names* for export

-Z right justify, fill with leading zeros; *n* is width, or set from first assignment

ulimit [type] [options] [limit]
set or print per-process limits

type (default is set both, print soft):

-H hard limit

-S soft limit

options:

-a all (display only)

-c core file size

-d "k" of data segment

-f maximum file size

-m "k" of physical memory

-n maximum file descriptors +1

-p size of pipe buffers

-s "k" of stack segment

-t cpu seconds

-v "k" of virtual memory

-f assumed if no options given. Sizes for -, -c, -f, and -p are all in 512-byte blocks. The new *limit* can be an arithmetic expression. Use unlimited to set limits to maximum

umask [-S] [mask]
set file creation permissions mask to complement of *mask* if octal, or symbolic value as in chmod. With no arguments, prints current mask. Octal *mask* is permissions to remove. Symbolic *mask* is permissions to keep

-S print current mask in symbolic form

unalias [-a] *names*
remove aliases *names*

-a remove all aliases

unset [-fnv] [names]
unset variables *names* (same as -v)

-f unset functions *names*

-n unset nameref, not referenced variable

-v unset variables *names*

wait [jobid...]
wait for job *jobid*; if no *job*, wait for all children

whence [-afpv] *name* ...
indicate how each *name* is treated by the shell

-a print all interpretations of *name*

-f skip the search for functions

-p do a path search even if *name* is a keyword, alias, or function

-v produce verbose information

Copyright © 1991, 1995, 1998, 2000, 2002
Specialized Systems Consultants, Inc.
P.O. Box 55549, Seattle, WA 98155-0549, USA.
All Rights Reserved.

The information in this card is used by permission. SSC sells a four-color version of this card that covers both ksh88 and ksh93 in much more detail. See <http://www.ssc.com> for more information.

Visit O'Reilly on the Web at www.oreilly.com

Control Commands

! *pipeline*
execute *pipeline*. If exit status is nonzero, exit zero. If exit status is zero, exit 1.

case *word* in [(|pat1[|pat2]...)] *list* ; ; ... esac
execute *list* associated with *pat* that matches *word*. Field splitting is not done for *word*. *pat* is a ksh pattern (see *Patterns*). | is used to indicate an OR condition. Use ;& instead of ; ; to execute next *list*

for *name* [in *words*] ; do *list* ; done
sequentially assign each *word* to *name* and execute *list*; if in *words* is missing use positional parameters

for (([e1]; [e2]; [e3])) ; do *list* ; done
evaluate *e1* once; evaluate *e2*, as long as it is nonzero, execute *list* and then evaluate *e3*. Any missing *eN* is treated as 1

function *name* { *list* ; }
define function *name*, body is *list*, ksh semantics

name () { *list* ; }
define function *name*, body is *list*. System V and POSIX syntax, POSIX semantics

if *list1* ; then *list2* [; e1if *list3* ; then *list4*]... [; else *list5*] ; fi
if executing *list1* returns successful exit status, execute *list2* else ...

select *name* [in *words*] ; do *list* ; done
print a menu of *words*, prompt with \$PS3 and read a line from stdin, saving it in REPLY; if set, time out after \$TMOUT seconds. If the line is the number of one of the words, set *name* to it, otherwise set *name* to null. Execute *list*. If in *words* is missing use positional parameters. Empty lines cause the menu to be printed again. If REPLY set to "" in *list*, reprint menu

time [*pipeline*]
execute *pipeline*; print elapsed, system and user times on stderr. If *pipeline*, is missing print user and system times for current shell and completed children on stderr

until *list1* ; do *list2* ; done
like while but negate termination test

while *list1* ; do *list2* ; done
execute *list1*; if last command in *list1* had a successful exit status, execute *list2* followed by *list1*; repeat until last command in *list1* returns an unsuccessful exit status

((...))
arithmetic evaluation, like let "... " (see *Arithmetic Evaluation*)

[[*expression*]]
evaluate *expression*, return successful exit status if true, unsuccessful if false (see *Conditional Expressions* for details)

(*list*)
execute *list* in a subshell environment

{ *list* ; }
list is executed in current shell only

Conditional Expressions

Used with the [[...]] compound command, which does not do pattern expansion or word splitting.

string
true if *string* is not null

-a *file*
true if *file* exists (-e is preferred)

-b *file*
true if *file* is a block device

-c *file*
true if *file* is a character device

-d *file*
true if *file* is a directory

-e *file*
true if *file* exists

-f *file*
true if *file* is a regular file

-g *file*
true if *file* has setgid bit set

-G *file*
true if *file*'s group is effective gid

-h *file*
true if *file* is a symbolic link

-k *file*
true if *file* has sticky bit set

-L *file*
true if *file* is a symbolic link

-n *string*
true if *string* has nonzero length

-o *option*
true if *option* is on

-O *file*
true if *file*'s owner is effective uid

-p *file*
true if *file* is a fifo (named pipe)

-r *file*
true if *file* is readable

-s *file*
true if *file* has nonzero size

-S *file*
true if *file* is a socket

-t *filedes*
true if *filedes* is a terminal

-u *file*
true if *file* has setuid bit set

-w *file*
true if *file* is writable

-x *file*
true if *file* is executable

-z *string*
true if *string* has zero length

file1 -nt *file2*
true if *file1* is newer than *file* or *file2* does not exist

file1 -ot *file2*
true if *file1* is older than *file2* or *file2* does not exist

file1 -ef *file2*
true if *file1* and *file2* are the same file

string = *pattern*
true if *string* matches *pattern* obsolete in ksh93

string == *pattern*
same as = (preferred in ksh93)

string != *pattern*
true if *string* does not match *pattern*

string1 < *string2*
true if *string1* is before *string2*

string1 > *string2*
true if *string1* is after *string2*

exp1 -eq *exp2*
true if *exp1* equals *exp2*

exp1 -ne *exp2*
true if *exp1* does not equal *exp2*

exp1 -lt *exp2*
true if *exp1* is less than *exp2*

exp1 -gt *exp2*
true if *exp1* is greater than *exp2*

exp1 -le *exp2*
true if *exp1* is less than or equal to *exp2*

exp1 -ge *exp2*
true if *exp1* is greater than or equal to *exp2*

(*expression*)
true if *expression* is true, for grouping

! *expression*
true if *expression* is false

exp1 && *exp2*
true if *exp1* AND *exp2* are true

exp1 || *exp2*
true if *exp1* OR *exp2* is true

If *file* is /dev/fd/*n*, file descriptor *n* is checked. Both && and || are short circuit. For =, ==, and !=, quote any part of *pattern* to treat it as a string. Operands of comparison operators undergo arithmetic evaluation.

Arithmetic Evaluation

Arithmetic evaluation is done with the `let` built-in command, the `((...))` command, and the `$(...)` notation for producing the result of an expression.

All arithmetic is done using `double` floating-point numbers. Use `typeset -i` to get integer variables, `typeset -ui` to create unsigned integers[†], and `typeset -E` or `-F` for floating-point variables. Integer constants look like `[base#]n` where *base* is a decimal number between 2 and 64, and *n* is in that base. The digits are 0–9, a–z, A–Z, @, and `_`. Constants in bases up to 36 may use mixed-case letters.[†] Floating-point constants are as in ANSI C.

[†] `ksh` accepts ANSI C octal and hexadecimal constants, as well as C character constants. Use a trailing `U`, `u`, `L`, and/or `I`, for integer constants that are unsigned and/or long.

The following operators based on C, with the same precedence and associativity, are available.

<code>++</code>	<code>--</code>	auto-increment, auto-decrement both prefix and postfix									
<code>+</code>		unary +, no effect									
<code>!</code>	<code>~</code>	logical, bitwise, arithmetic negation									
<code>**</code>		exponentiation [†]									
<code>*</code>	<code>/</code>	multiply, divide, modulus									
<code>+</code>	<code>-</code>	addition, subtraction									
<code><<</code>	<code>>></code>	left shift, right shift									
<code><</code>	<code><=</code>	<code>></code>	<code>>=</code>	comparisons							
<code>==</code>	<code>!=</code>	equals, not equals									
<code>&</code>		bitwise AND									
<code>^</code>		bitwise XOR									
<code> </code>		bitwise OR									
<code>&&</code>		logical AND, short circuit									
<code> </code>		logical OR, short circuit									
<code>?:</code>		in-line conditional									
<code>=</code>	<code>+=</code>	<code>--</code>	<code>*=</code>	<code>/=</code>	<code>%=</code>	<code>&=</code>	<code> =</code>	<code>^=</code>	<code><<=</code>	<code>>>=</code>	assignment operators
<code>,</code>		sequential expression evaluation									

Inside `let`, `((...))`, and `$(...)`, variable names do not need a `$` to get their values.

The following mathematical functions are available. They are called using C function call syntax.

<code>abs</code>	absolute value
<code>acos</code>	arc cosine
<code>asin</code>	arc sine
<code>atan</code>	arc tangent
<code>atan2</code> [†]	arctan two vars
<code>cos</code>	cosine
<code>cosh</code>	hyperbolic cosine
<code>exp</code>	exponential, <i>e</i> ^{<i>x</i>}
<code>fmod</code> [†]	floating remainder
<code>hypot</code> [†]	euclidean distance
<code>int</code>	integer part
<code>log</code>	natural logarithm
<code>pow</code> [†]	exponentiation
<code>sin</code>	sine
<code>sinh</code>	hyperbolic sine
<code>sqrt</code>	square root
<code>tan</code>	tangent
<code>tanh</code>	hyperbolic tangent

Variable Substitution

<code>\$name</code>	reference to shell variable <i>name</i>
<code>\${name}</code>	use braces to delimit shell variable <i>name</i> . Required if <i>name</i> contains periods
<code>\${name-word}</code>	use variable <i>name</i> if set, else use <i>word</i>
<code>\${name=word}</code>	as above but also set <i>name</i> to <i>word</i>
<code>\${name?word}</code>	use <i>name</i> if set, otherwise print <i>word</i> and exit (interactive shells do not exit)
<code>\${name+word}</code>	use <i>word</i> if <i>name</i> set, otherwise use nothing
<code>\${name[n]}</code>	element <i>n</i> in array <i>name</i>
<code>\${name[word]}</code>	element <i>word</i> in associative array <i>name</i>
<code>\${#name}</code>	length of shell variable <i>name</i>
<code>\${#name[*]}</code>	number of elements in array <i>name</i>
<code>\${#name[@]}</code>	number of elements in array <i>name</i>
<code>\${!name}</code>	name of variable referenced by <i>name</i> ; same as <i>name</i> unless <i>name</i> is a nameref
<code>\${!name[subscript]}</code>	value of actual <i>subscript</i> for array <i>name</i>
<code>\${!name[*]}</code>	list of subscripts in array <i>name</i> ; inside double quotes expand to one word, each arg separated by first char of <code>\$IFS</code>
<code>\${!name[@]}</code>	list of subscripts in array <i>name</i> ; inside double quotes expand to separate words
<code>\${!prefix*}</code>	
<code>\${!prefix@}</code>	list of variables whose names begin with <i>prefix</i>
<code>\${name#pat}</code>	remove shortest leading substring of <i>name</i> that matches <i>pat</i>
<code>\${name##pat}</code>	remove longest leading substring of <i>name</i> that matches <i>pat</i>
<code>\${name%pat}</code>	remove shortest trailing substring of <i>name</i> that matches <i>pat</i>
<code>\${name%%pat}</code>	remove longest trailing substring of <i>name</i> that matches <i>pat</i>
<code>\${name:start}</code>	
<code>\${name:start:length}</code>	<i>length</i> characters of <i>name</i> starting at <i>start</i> (counting from 0); use rest of value if no <i>length</i> . If <i>name</i> is <code>*</code> or <code>@</code> or an array indexed by <code>*</code> or <code>@</code> , <i>start</i> and <i>length</i> indicate the array index and count of elements. <i>start</i> and <i>length</i> can be arithmetic expressions
<code>\${name/pattern/string}</code>	value of <i>name</i> with first match of <i>pattern</i> replaced with <i>string</i>
<code>\${name/pattern}</code>	value of <i>name</i> with first match of <i>pattern</i> deleted
<code>\${name//pattern/string}</code>	value of <i>name</i> with every match of <i>pattern</i> replaced with <i>string</i>

<code>\$(name/#pattern/string)</code>	EDITOR	command-line editor if VISUAL unset
value of <i>name</i> with match of <i>pattern</i> replaced with <i>string</i> ; match must occur at beginning	ENV	in interactive shells, value is variable, command, and arithmetic substituted for path of start-up file
<code>\$(name/%pattern/string)</code>	FCEDIT	obsolete default editor for <code>hist</code> command
value of <i>name</i> with match of <i>pattern</i> replaced with <i>string</i> ; match occurs at end	FIGNORE	pattern giving the set of filenames to ignore when doing pattern matching
Note: for <code>-</code> , <code>=</code> , <code>?</code> , and <code>+</code> , using <i>name</i> : instead of <i>name</i> tests whether <i>name</i> is set and non-NULL; using <i>name</i> tests only whether <i>name</i> is set.	FPATH	search path for functions defined with <code>typeset -u</code> ; provide <code>.</code> (dot) explicitly to search the current directory [†]
For <code>#</code> , <code>##</code> , <code>%</code> , <code>%%</code> , <code>/</code> , <code>//</code> , <code>/#</code> , and <code>/%</code> , when <i>name</i> is <code>*</code> or <code>@</code> or an array indexed by <code>*</code> or <code>@</code> , the substring or substitution operation is applied to each element. The replacement text can use the <code>\n</code> notation (see <i>Patterns</i>).	HISTCMD	history number of current command
	HISTEDIT	default editor for <code>hist</code> command; if set, overrides <code>\$FCEDIT</code>

Predefined Variables

The values of `PS1`, `PS3`, and `PS4` undergo variable, command, and arithmetic substitution before being printed. Variables whose names contain periods require `${...}`.

<code>\$n</code>	use positional parameter <i>n</i> , <i>n</i> <= 9
<code>\${n}</code>	use positional parameter <i>n</i>
<code>\$*</code>	all positional parameters
<code>@</code>	all positional parameters
<code>"\$*"</code>	equivalent to <code>"\$1 \$2 ..."</code>
<code>"\$@"</code>	equivalent to <code>"\$1" "\$2" ...</code>
<code>##</code>	number of positional parameters
<code>\${#*}</code>	number of positional parameters
<code>\${#@}</code>	number of positional parameters
<code>\$-</code>	options to shell or by set
<code>\$?</code>	value returned by last command
<code>\$\$</code>	process number of current shell
<code>!</code>	process number of last background cmd
<code>_</code>	value of last positional argument in last command
<code>.sh.edchar</code>	character(s) entered when processing a KEYBD trap; changing it replaces the character(s) that caused the trap
<code>.sh.edcol</code>	position of cursor in most recent KEYBD trap
<code>.sh.edmode</code>	equal to <code>ESC</code> when processing a KEYBD trap in <code>vi</code> mode, null otherwise
<code>.sh.edtext</code>	characters in the input buffer in a KEYBD trap
<code>.sh.match</code> [†]	indexed array of text that matched in a variable substitution operation; index 0 is the whole value, other indices match parenthesized subexpressions.
<code>.sh.name</code>	name of the variable running a discipline function
<code>.sh.subscript</code>	subscript of the variable running a discipline function
<code>.sh.value</code>	value of the variable inside the set and get discipline functions
<code>.sh.version</code>	identifies the version of the shell
<code>CDPATH</code>	search path for <code>cd</code> command
<code>COLUMNS</code>	length of window for editing and <code>select</code> menus

EDITOR	command-line editor if VISUAL unset
ENV	in interactive shells, value is variable, command, and arithmetic substituted for path of start-up file
FCEDIT	obsolete default editor for <code>hist</code> command
FIGNORE	pattern giving the set of filenames to ignore when doing pattern matching
FPATH	search path for functions defined with <code>typeset -u</code> ; provide <code>.</code> (dot) explicitly to search the current directory [†]
HISTCMD	history number of current command
HISTEDIT	default editor for <code>hist</code> command; if set, overrides <code>\$FCEDIT</code>
HISTFILE	stores command history
HISTSIZE	number of previous commands to keep available
HOME	home directory for <code>cd</code> command and for tilde expansion
IFS	field separators (space, tab, newline)
LANG	name of current locale
LC_ALL	current locale; overrides LANG and other LC_ variables
LC_COLLATE	current locale for character collation
LC_CTYPE	current locale for character class functions (see <i>Patterns</i>)
LC_NUMERIC	current locale for decimal point
LINES	controls vertical size of <code>select</code> menus
LINENO	line number of line being executed in script or function
MAIL	name of a mail file, if any
MAILCHECK	check for mail every <i>n</i> seconds (600 default)
MAILPATH	filenames to check for new mail; uses <code>:</code> separator; <i>filename</i> may be followed by <i>?message</i> ; <code>_</code> in <i>message</i> is matched mail filename
OLDPWD	previous working directory
OPTARG	value of last argument processed by <code>getopts</code>
OPTIND	index of last argument processed by <code>getopts</code>
PATH	command and function search path
PPID	process id of shell's parent
PS1	primary prompt string (<code>\$</code>)
PS2	secondary prompt string (<code>></code>)
PS3	<code>select</code> command prompt string (<code>#?</code>)
PS4	debug prompt string (<code>+</code>)
PWD	current working directory
RANDOM	set each time it's referenced, 0 – 32767
REPLY	set by <code>select</code> and <code>read</code> commands

SECONDS	number of seconds since shell invocation
SHELL	name of shell programs should use
TMOU	number of seconds to wait idly before terminating; for <code>prompt</code> , <code>select</code> and <code>read</code>
VISUAL	choice of command-line editor

Patterns

<code>?</code>	match single character in filename
<code>*</code>	match 0 or more characters in filename
<code>[chars]</code>	match any of <i>chars</i> (pair separated by a <code>-</code> matches a range)
<code>[!chars]</code>	match any except <i>chars</i>
<code>?(pat-list)</code>	optionally match any of the patterns
<code>*(pat-list)</code>	match 0 or more of any of the patterns
<code>+(pat-list)</code>	match 1 or more of any of the patterns
<code>@(pat-list)</code>	match exactly 1 of any of the patterns
<code>!(pat-list)</code>	match anything but any of the patterns
<code>\n</code>	text matched by <i>n</i> th sub-pattern in (...)
<code>{n}(pat-list)</code> [†]	match exactly <i>n</i> of any of the patterns
<code>{n,m}(pat-list)</code> [†]	match <i>n</i> to <i>m</i> of any of the patterns

pat-list is a list of one or more patterns separated by `|`. All the patterns must match if `&` is used instead of `|`. `&` has higher precedence than `|`. Add a minus before the `(` to match the shortest match instead of the longest match.[†] Use `~(±opt[:pat])` within a pattern to control matching for the *pat* if present, or the rest of the (sub)pattern if not. `A` - enables the option, `+` disables it. Options are `g` for “greedy” (longest) matching, and `i` to ignore case.[†] The POSIX `[[=c=]]` and `[[.c.]]` notations for same-weight characters and collating elements are accepted. The notation `[[:class:]]` defines characters classes:

<code>alnum</code>	alphanumeric
<code>alpha</code>	alphabetic
<code>blank</code>	space or tab
<code>cntrl</code>	control
<code>digit</code>	decimal
<code>graph</code>	nonspaces
<code>lower</code>	lowercase
<code>print</code>	printable
<code>punct</code>	punctuation
<code>space</code>	whitespace
<code>upper</code>	uppercase
<code>word</code> [†]	like <code>[[:alnum:]_]</code>
<code>xdigit</code>	hexadecimal

ANSI C escape sequences are valid in patterns. In addition, you may use these escape sequences:[†]

<code>\d</code>	same as <code>[[:digit:]]</code>
<code>\D</code>	same as <code>[[!:digit:]]</code>
<code>\s</code>	same as <code>[[:space:]]</code>
<code>\S</code>	same as <code>[[!:space:]]</code>
<code>\w</code>	same as <code>[[:word:]]</code>
<code>\W</code>	same as <code>[[!:word:]]</code>

When expanding filenames, `.` and `..` are ignored, filenames matching the pattern in `$FIGNORE` are also ignored, and a leading `.` must be supplied in the pattern to match filenames that begin with `.`

Input/Output

Redirections are done left to right, after pipes are set up. Default file descriptors are `stdin` and `stdout`. File descriptors above 2 are marked close-on-exec. Redirections to/from the null string are not allowed.

<code>[n]<file</code>	use <i>file</i> for input
<code>[n]>file</code>	use <i>file</i> for output
<code>[n]> file</code>	like <code>></code> but overrides <code>noclobber</code>
<code>[n]>>file</code>	like <code>></code> but append to <i>file</i> if it exists
<code>[n]<>file</code>	open <i>file</i> for read/write (default: <code>fd0</code>)
<code>[n]<&m</code>	duplicate input file descriptor from <i>m</i>
<code>[n]>&m</code>	duplicate output file descriptor from <i>m</i>
<code>[n]<&m-</code>	move input file descriptor <i>m</i>
<code>[n]>&m-</code>	move output file descriptor <i>m</i>
<code>[n]<&-</code>	close input file descriptor
<code>[n]>&-</code>	close output file descriptor
<code>[n]<<word</code>	input comes from shell script, treat line with <i>word</i> as EOF on input; if any of <i>word</i> is quoted, no additional processing is done on input by the shell. Otherwise:

- do variable, command, arithmetic substitutions
- ignore escaped newlines
- use `\` to quote `\`, `$`, ```, and first character of *word*

<code>[n]<<-word</code>	as above with leading tabs ignored
<code>[n]<&p</code>	move input from coprocess to <code>fd n</code>
<code>[n]>&p</code>	move output to coprocess to <code>fd n</code>

Built-in Commands

These commands are executed directly by the shell. Almost all accept `-?` to print a usage message, and `--` to mark the end of options. Assignments for the `alias`, `export`, `readonly`, and `typeset` commands are special: tilde expansion is done after the `=`, and variable substitutions do *not* undergo field splitting.

`. file [arg ...]`
read and execute commands from *file*. If arguments, save and restore positional params. If *file* is a function name declared with `function`, execute it using POSIX semantics

`:` null command; returns 0 exit status

`[...]` synonym for `test`; see `test` below

`alias [-ptx] [name[=value] ...]`
create an alias; with no arguments, print all aliases

- `-p` print alias before each alias
- `-t` set or print a tracked alias
- `-x` obsolete: has no effect

`bg [jobid]`
put *jobid* in the background

`break [n]`
exit from enclosing `for`, `while`, `until`, or `select` loop. If *n* supplied, exit from *n*th enclosing loop