NAME

sh, ., break, case, cd, continue, eval, exec, exit, export, for, if, read, readonly, set, shift, trap, umask, wait, while − shell

SYNOPSIS

sh [**−eiknqstvxu**] [**−c** *str*] **[***file*]

OPTIONS

−c Execute the commands in *str*

−e Quit on error

−i Interactive mode; ignore QUIT, TERMINATE, INTERRUPT

−k Look for name=value everywhere on command line

−n Do not execute commands

−q Change qflag from sig_ign to sig_del

−s Read commands from standard input

−t Exit after reading and executing one command

−v Echo input lines as they are read

−x Trace

−u Unset variables

EXAMPLES

sh script # Run a shell script

DESCRIPTION

Sh is the shell, which forms the user's main interface with the system. On startup, the shell reads /etc/profile and $HOME/.profile, if they exist, and executes any commands they contain. The Minix shell has most of the features of the V7 (Bourne) shell, including redirection of input and output, pipes, magic characters, background processes, and shell scripts. A brief summary follows, but whole books have been written on shell programming alone.

Some of the more common notations are:

date	#	Regular command
sort <file	#	Redirect *stdin* (standard input)
sort <file1 >file2	#	Redirect *stdin* and *stdout*
cc file.c 2>error	#	Redirect *stderr*
a.out >f 2>&1	#	Combine standard output and standard error
sort <file1 >>file2	#	Append output to *file2*
sort <file1 >file2 &	#	Background job
(ls −l; a.out) &	#	Run two background commands sequentially
sort <file | wc	#	Two-process pipeline
sort <f | uniq | wc	#	Three-process pipeline
ls −l *.c	#	List all files ending in *.c*
ls −l [*a-c*]*	#	List all files beginning with *a*, *b*, or *c*
ls −l ?	#	List all one-character file names
ls \?	#	List the file whose name is question mark
ls ′???′	#	List the file whose name is three question marks
v=/usr/ast	#	Set shell variable *v*
ls −l $v	#	Use shell variable *v*
PS1=′Hi! ′	#	Change the primary prompt to *Hi!*
PS2=′More: ′	#	Change the secondary prompt to *More:*
ls −l $HOME	#	List the home directory
echo $PATH	#	Echo the search path
echo $?	#	Echo exit status of previous command in decimal
echo $$	#	Echo shell's pid in decimal

echo $!	#	Echo PID of last background process
echo $#	#	Echo number of parameters (shell script)
echo $2	#	Echo second parameter (shell script)
echo "$2"	#	Echo second parameter without expanding spaces
echo $*	#	Echo all parameters (shell script)
echo $@	#	Echo all parameters (shell script)
echo "$@"	#	Echo all parameters without expanding spaces

The shell uses the following variables for specific purposes:

SHELL	the path of the current shell
HOME	the default value for the cd(1) command
PATH	the directories to be searched to find commands
IFS	the internal field separators for command strings
PS1	the primary shell prompt
PS2	the secondary shell prompt

There are various forms of substitution on the shell command line:

‘...‘	Command string between back-quotes is replaced by its output
"..."	Permits variable substitution between quotes
’...’	Inhibits variable substitution between quotes
$VAR	Replaced by contents of variable VAR
${VAR}	Delimits variable VAR from any following string

The expressions below depend on whether or not VAR has ever been set. If VAR has been set, they give:

${VAR-str}	Replace expression by VAR, else by str
${VAR=str}	Replace expression by VAR, else by str and set VAR to str
${VAR?str}	Replace expression by VAR, else print str and exit shell
${VAR+str}	Replace expression by str, else by null string

If a colon is placed after VAR, the expressions depend on whether or not VAR is currently set and non-null.

The shell has a number of built-in commands:

<table>
<tr><td>:</td><td>return true status</td></tr>
<tr><td>. fn</td><td>execute shell script fn on current path</td></tr>
<tr><td>break [n]</td><td>break from a for, until or while loop; exit n levels</td></tr>
<tr><td>continue [n]</td><td>continue a for, until or while loop; resume nth loop</td></tr>
<tr><td>cd [dir]</td><td>change current working directory; move to $HOME</td></tr>
<tr><td>eval cmd</td><td>rescan cmd, performing substitutions</td></tr>
<tr><td>eval</td><td>rescan the current command line</td></tr>
<tr><td>exec cmd</td><td>execute cmd without creating a new process</td></tr>
<tr><td>exec <|></td><td>with no command name, modify shell I/O</td></tr>
<tr><td>exit [n]</td><td>exit a shell program, with exit value n</td></tr>
<tr><td>export [var]</td><td>export var to shell’s children; list exported variables</td></tr>
<tr><td>pwd</td><td>print the name of the current working directory</td></tr>
<tr><td>read var</td><td>read a line from stdin and assign to var</td></tr>
<tr><td>readonly [var]</td><td>make var readonly; list readonly variables</td></tr>
<tr><td>set -f</td><td>set shell flag (+f unsets flag)</td></tr>
<tr><td>set str</td><td>set positional parameter to str</td></tr>
<tr><td>set</td><td>show the current shell variables</td></tr>
<tr><td>shift</td><td>reassign positional parameters (except ${0}) one left</td></tr>
<tr><td>times</td><td>print accumulated user and system times for processes</td></tr>
<tr><td>trap arg sigs</td><td>trap signals sigs and run arg on receipt</td></tr>
<tr><td>trap</td><td>list trapped signals</td></tr>
<tr><td>umask [n]</td><td>set the user file creation mask; show the current umask</td></tr>
<tr><td>wait [n]</td><td>wait for process pid n; wait for all processes</td></tr>
</table>

The shell also contains a programming language, which has the following operators and flow control statements:

#	Comment	The rest of the line is ignored
=	Assignment	Set a shell variable
&&	Logical AND	Execute second command only if first succeeds
||	Logical OR	Execute second command only if first fails
(...)	Group	Execute enclosed commands before continuing

for	For loop (for ... in ... do ... done)
case	Case statement ((case ...) ... ;; ... esac)
esac	Case statement end
while	While loop (while ... do ... done)
do	Do/For/While loop start (do ... until ...)
done	For/While loop end
if	Conditional statement (if ... else ... elif ... fi)
in	For loop selection
then	Conditional statement start
else	Conditional statement alternative
elif	Conditional statement end
until	Do loop end
fi	Conditional statement end

SEE ALSO

echo(1), **expr**(1), **pwd**(1), **true**(1).