NAME

sh – command interpreter (the new Bourne Shell)

SYNOPSIS

sh [−ecnpstvx] [arg] ...

DESCRIPTION

sh is a command interpreter (shell) that executes commands read from a terminal or a file. See **invocation** for the meaning of arguments to the shell.

Introduction. Simple commands to the shell consist of one or more words separated by blanks. The first word is the name of the command to be executed; remaining words are passed as arguments to the command. For example:

```
who
```

prints the names of users currently logged in.

```
ls −l
```

lists (in long form) the names of files in the current directory.

Most commands produce output on the "standard output" that is, initially, connected to the terminal. This output may be sent to a file by saying, for example,

```
ls −l >file
```

The notation >*file* is interpreted by the shell and is not passed on to the command as an argument. If the file does not exist it is created. If the file exists it will be overwritten with the output from the command.

The standard input (by default the terminal) may also be taken from a file by saying, for example,

```
wc <file
```

The standard output of a command may be connected to the standard input of another by writing, for example,

```
ls −l | wc
```

The effect is the same as

```
ls −l >file; wc <file
```

except that *file* is not created. Instead the two processes are connected by a pipe and are run in parallel.

Commands. A *simple-command* is a sequence of non blank *words* separated by blanks (a blank is a **tab** or a **space**). The first word specifies the name of the command to be executed. Except as specified below the remaining words are passed as arguments to the invoked command. The command name is passed as argument 0 (see *exec* (II)). The *value* of a simple-command is its exit status if it terminates normally or 200+*status* if it terminates abnormally (see *signal* (II) for a list of status values).

A *pipeline* is a sequence of one or more *commands* separated by |. The standard output of each command but the last is connected by a *pipe* (II) to the standard input of the next command. Each command is run as a separate process; the shell waits for the last command to terminate.

A *list* is a sequence of one or more *pipelines* separated by ;, **&**, **&&** or || and optionally terminated by ; or **&**. A semicolon causes sequential execution; an ampersand causes the preceding *pipeline* to be executed without waiting for it to finish. The symbol **&&** (||) causes the *list* following to be executed only if the preceding *pipeline* returns a zero (non zero) value. Newlines may appear in a *list* to delimit commands.

A *command* is either a simple-command or one of the following. The value returned by a command is that of the last simple-command executed in the command.

for *name* [**in** *word* ...] **do** *list* **done**

Each time a **for** command is executed *name* is set to the next word in the **for** word list. The default **for** word list is **$***. Execution ends when there are no more words in the list.

case *word* **in** [*pattern*) *list* ;;] ... **esac**

A **case** command executes the *list* associated with the first pattern that matches *word.* The form of the patterns is the same as that used for file name generation.

if *list* **then** *list* [**elif** *list* **then** *list*] ... [**else** *list*] **fi**

The *list* following **if** is executed and if it returns zero the *list* following **then** is executed. Otherwise, the *list* following **elif** is executed and if its value is zero the *list* following **then** is executed. Failing that the **else** *list* is executed.

while *list* [**do** *list*] **done**

A **while** command repeatedly executes the **while** *list* and if its value is zero executes the **do** *list;* otherwise the loop terminates. The value returned by a **while** command is that of the last executed command in the **do** *list.* **until** may be used in place of **while** to negate the loop termination test.

Two forms of command grouping are available. In the first, the *list* is executed by a subshell:

(*list*) or { *list* }

The following words are only recognised as the first word of a command and when not quoted.

if then else elif fi case in esac for while until do done { }

Parameter substitution. The character **$** is used to introduce substitutable parameters that may be given values, for example, using the set command or when the shell is invoked.

$ {*parameter*}
A *parameter* is a sequence of letters, digits or underscores (a *name*), a digit, or the character *. The value, if any, of the parameter is substituted. If *parameter* is a digit then it is a positional parameter. If *parameter* is * then all the positional parameters, starting with **$1**, are substituted separated by spaces. **$0** is set from argument zero when the shell is invoked. The braces may be omitted.

$ {*parameter*−*string*}
If *parameter* is set then substitute its value; otherwise substitute *string.*

$ {*parameter*= *string*}
If *parameter* is not set then set it to *string;* the value of the parameter is then substituted.

$ {*parameter* ? *string*}
If *parameter* is set then substitute its value; otherwise, print *string* and exit from the shell.

$ {*parameter*+*string*}
If *parameter* is set then substitute *string;* otherwise substitute nothing.

The following *parameters* are set by the shell.

n	The number of positional parameters.
r	The value returned by the last executed command.
pid	The process number of this shell.
pcs	The process number of the last invoked background command.

The following *parameters* are used but not set by the shell.

h	The default argument (home directory) for the **cd** command.
p	The search path for commands (see **execution**).
mail	If this variable is set to the name of a mail file then the shell informs the user of the arrival of mail in the specified file.

Command substitution. The standard output from a command enclosed in a pair of grave accents (\`\`) may be used as part or all of a word; trailing newlines are removed.

Blank interpretation. Following parameter and command substitution arguments are scanned for blanks or newlines. Null arguments (other than "" or '') are also removed.

File name generation. Following substitution, each command word is scanned for the characters *, ? and [. If one of these characters appears then the word is regarded as a pattern. The word is replaced with alphabetically sorted file names that match the pattern. If no file name is found that matches the pattern then the word is left unchanged. The character **.** at the start of a file name or immediately following a /, and the character /, must be matched explicitly.

*	Matches any string, including the null string.
?	Matches any single character.
[...]	Matches any one of the characters enclosed. A pair of characters separated by − matches any character lexically between the pair.

Quoting. The following characters have a special meaning to the shell and cause termination of an word unless quoted.

; & ( ) | < > **newline space tab**

A character may be *quoted* by preceding it with a \\. \\newline is ignored. All characters enclosed between a pair of quote marks ('') are quoted. Inside double quotes ("") parameter and command substitution occurs and \\ quotes the characters \\ **$** \` and ".

Prompting. When used interactively the shell issues a prompt, by default **$**, before reading a command. This prompt may be changed using the command **set -p** *prompt.* If at any time a newline is typed and further input is needed to complete a command then the prompt > is issued.

Input output. Before a command is executed its input and output may be redirected using a special notation interpreted by the shell. The following may appear anywhere in a simple-command or may precede or follow a *command* and are not passed on to the invoked command. Substitution occurs before *word* or *digit* is used.

< *word*	Use file *word* as standard input (file descriptor 0).
> *word*	Use file *word* as standard output (file descriptor 1). If the file does not exist then it is created; otherwise it is truncated to zero length.
>> *word*	Use file *word* as standard output. If the file exists then output is appended (by seeking to the end); otherwise the file is created.
<< *word*	The shell input is read up to a line the same as *word*, or end of file. (Notice *word* must be at the beginning of a line, nothing preceding it.) The resulting document becomes the standard input. If any character of *word* is quoted then no interpretation is placed

upon the characters of the document; otherwise, parameter and command substitution occurs and \ is used to quote the characters \ **$** ` and the first character of *word.*

< & *digit* The standard input is duplicated from file descriptor *digit.* (See *dup* (II).)

> & *digit* The standard output is duplicated from file descriptor *digit.*

If one of the above is preceded by a digit then the file descriptor created is that specified by the digit (instead of the default 0 or 1). For example, ... 2>&1 creates file descriptor 2 to be a duplicate of file descriptor 1.

Environment. If a command is followed by **&** then the default standard input for the command is the empty file (/dev/null). Otherwise, the environment for the execution of a command contains the file descriptors of the invoking shell as modified by input output specifications. The INTERRUPT and QUIT signals for an invoked command are ignored if the command is followed by **&**; otherwise signals have the values inherited by the shell from its parent. (But see also **trap.**)

Execution. Each time a command is executed the above substitutions are carried out. Except for the **special commands** listed below a new process is created and an attempt is made to execute the command via an *exec* (II).

The shell parameter **$p** defines the search path for the directory containing the command. In the following, each alternative directory name is separated by −. The default path is **−/bin/−/usr/bin/**. If the command name contains a / then the search path is not used. Otherwise, each directory in the path is searched for an executable file. If the file is not an *a.out* file but has execute permission then it is assumed to be a file containing shell commands. A subshell is created containing a copy of the non local name parameters from the invoking shell. (A name that begins with an underscore is *local.*) A parenthesised command is also executed in a subshell and will not affect the parameters of this shell.

Special commands. The following commands are executed in the shell process and except where specified no input output redirection is permitted for such commands.

: No effect; the command is ignored.

. *file* Read and execute commands from *file* and return. The search path **$p** is used to find the directory containing *file.*

break Exit from the enclosing **for** or **while** loop, if any.

continue Resume the next iteration of the enclosing **for** or **while** loop.

cd [*arg*]
Change the current directory to *arg.* The shell parameter **$h** is the default *arg.*

eval [*arg* ...]
The arguments are read as input to the shell and the resulting command(s) executed.

exec [*arg* ...]
The command specified by the arguments is executed in place of this shell. Input output arguments may appear and if no other arguments are given cause the shell input output to be modified.

exit [*n*]
Causes a non interactive shell to exit with the exit status specified by *n.* If *n* is omitted then the exit status is that of the last command executed. (An end of file will also exit from the shell.)

login [*arg* ...]
Equivalent to *exec login arg ...*

readonly [*name* ...]
The values of the given names may not be changed by assignment.

set −einptvx

−e	If non interactive then exit immediately if a command fails.
−i	This shell behaves as if it were interactive.
−n	Read commands but do not execute them.
−p	Set the shell prompt to the next argument.
−t	Exit after reading and executing one command.
−v	Print shell input lines as they are read.
−x	Print commands and their arguments as they are executed.

set [*arg* ...]
If the first argument is a name then it is set to the concatenation of the remaining arguments. Otherwise all arguments must be of the form *name*=*value* and the shell parameter *name* is set to *value.* If no arguments are given then the values of all names are printed.

shift The positional parameters from **$2...** are renamed **$1...** .

trap [*arg* [*n*] ...]
arg is a command to be executed when the shell receives signal(s) *n.* Trap commands are executed in order of signal number. If no arguments are given then all traps are reset to their original values. If *arg* is the null string then this signal is ignored by the shell and by invoked commands. If *n* is 0 then the command *arg* is executed on exit from the shell.

wait [*n*]
Wait for the specified process and report its termination status. If *n* is not given then all currently active processes are waited for.

Invocation. When the shell is invoked and the first character of argument zero is −, commands are read from the file **.profile**. If the **−c** flag is present then subsequent commands are read from the string *arg.* Otherwise, if the **−s** flag is present or if *arg* is absent then commands are read from the standard input. Shell output is written to file descriptor 2. If the shell input and output are attached to a terminal (as told by *gtty*) then this shell is *interactive.* In this case TERMINATE is ignored (so that **kill 0** does not kill an interactive shell) and INTERRUPT is caught and ignored (so that **wait** is interruptable). In all cases QUIT is ignored by the shell.

The remaining flags are described under the **set** command. Arguments of the form *name*=*value* cause the shell parameter *name* to be initialized to *value.* Any remaining arguments are assigned to **$1,...** .

EXIT STATUS

Errors detected by the shell, such as syntax errors or failure to execute a command (as distinct from failure of the command itself), cause the shell to return a non zero exit status. If the shell is being used non interactively then execution of the shell file is abandoned. Otherwise, the shell returns the exit status of the last command executed (see also **exit**).

FILES

.profile /tmp/sh*

SEE ALSO

test (I) exec (II) fork (II) wait (II) a.out (V)