Linux and Unix sh command

www.computerhope.com/unix/ush.htm

Quick links

About sh

Syntax

```
sh [-acefhikmnprstuvx] [arg] ...
```

Description And History

sh is a command language interpreter that executes commands read from a command line string, the standard input, or a specified file.

The Bourne shell was developed in 1977 by Stephen Bourne at AT&T's Bell Labs in 1977. It was the default shell of Unix Version 7. Most Unix-like systems contain the file **/bin/sh** which is either the Bourne shell, or a symbolic link (or hard link) to a compatible shell.

The Bourne Shell was originally developed as a replacement for the Thompson shell, whose executable file was also named **sh**. Although it is used as an interactive command interpreter, its original purpose was to function as a scripting language.

Features of the Bourne Shell include:

- scripts can be invoked as commands by using their filename
- the shell may be used interactively or non-interactively
- commands may be executed synchronously or asynchronously
- the shell supports input and output redirection, and pipelines
- a robust set of built-in commands
- flow control constructs, quotation facilities, and functions
- typeless variables
- both local and global variable scopes
- scripts can be interpreted, i.e., they do not have to be compiled to be executed
- Command substitution using back quotes, e.g.: `` `command` ``
- "Here documents": the use of **<<** to embed a block of input text within a script
- "**for**/**do**/**done**" loops, in particular the use of **$*** to loop over arguments
- "**case**/**in**/**esac**" selection mechanism, primarily intended to assist argument parsing
- support for environment variables using keyword parameters and exportable variables
- strong provisions for controlling input and output and in its expression-matching facilities.

Use of the Bourne Shell has largely been suerceded by the Bourne-Again Shell (**bash**), which supports more user-friendly interactive features such as job control and a command history.

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**). The value of a *simple-command* is its exit status if it terminates normally or 200+status if it terminates abnormally (see our signals page for a list of status values).

A *pipeline* is a sequence of one or more commands separated by a vertical bar ("**|**"). The standard output of each command but the last is connected by a pipe 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. The value of a pipeline is the exit status of its last command.

A *list* is a sequence of one or more pipelines separated by "**;**", "**&**", "**&&**" or "**||**" and optionally terminated by "**;**" or "**&**". "**;**" and "**&**" have equal precedence which is lower than that of "**&&**" and "**||**", "**&&**" and "**||**" also have equal precedence. 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, instead of semicolons, to delimit commands.

A "**#**" at the beginning of a word starts a comment and causes the rest of the line to be ignored.

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**	For loop. Each time a **for** command is executed, *name* is set to the next *word* in the **for** word list. If '**in** *word* ...' is omitted, then '**in "$@"**' is assumed. Execution ends when there are no more words in the list.
case *word* **in** [*pattern* [**|** *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** ("else if") 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.
(*list* **)**	Execute *list* in a subshell.
{ *list*; **}**	*list* is simply executed.
*name***()** **{** *list*; **}**	Defines the shell function *name*. Each time *name* is recognized as a command, *list* is executed, with the positional parameters **$1**, **$2**... set to the arguments of the command. After the function returns, the previous positional parameters are restored.

The following words are only recognized as the first word of a command, and when not enclosed in quotes:

- **if**
- **then**

- **else**
- **elif**
- **fi**
- **case**
- **in**
- **esac**
- **for**
- **while**
- **until**
- **do**
- **done**
- **{**
- **}**

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. For example, if the executable script **echotest.sh** contained the command:

```
echo "The name of this script is `basename
$0`."
```

Then running the script would display the combined output of **echo** and **basename**:

```
The name of this script is
echotest.sh.
```

Parameter Substitution

The character **$** is used to introduce substitutable parameters. Positional parameters may be assigned values by **set**. Variables may be set in the form "**name=***value* [**name=***value*] ...".

${*parameter***}**	A *parameter* is a sequence of letters, digits or underscores (a name), a digit, or any of the characters *** @ # ? - $!** . The value, if any, of the parameter is substituted. The braces are required only when parameter is followed by a letter, digit, or underscore that is not to be interpreted as part of its name. If *parameter* is a digit then it is a positional parameter. If parameter is ***** or **@** then all the positional parameters, starting with **$1**, are substituted separated by spaces. **$0** is set from argument zero when the shell is invoked.
${*parameter***:-***word***}**	If *parameter* is set and not empty then substitute its value; otherwise substitute *word*.
${*parameter***:=***word***}**	If *parameter* is not set and not empty then set it to *word*; the value of the parameter is then substituted. Positional parameters may not be assigned-to in this way.

${*parameter***:?***word***}**	If *parameter* is set and not empty then substitute its value; otherwise, print *word* and exit from the shell. If *word* is omitted then a standard message is printed.
${*parameter***:+***word***}**	If *parameter* is set and not empty then substitute *word*; otherwise substitute nothing.

If the **:** is omitted, the substitutions are only executed if the parameter is set, even if it is empty.

In the above, *word* is not evaluated unless it is to be used as the substituted string. So, for example, **"echo ${d-`pwd`}"** will only execute **pwd** if **d** is unset.

The following parameters are automatically set by the shell:

#	The number of positional parameters, in decimal.
-	Options supplied to the shell on invocation or by **set**.
?	The value returned by the last executed command, in decimal.
$	The process number of this shell.
!	The process number of the last background command invoked.

The following parameters are used by the shell:

CDPATH	The search path for the **cd** command.
HOME	The default argument (home directory) for the **cd** command.
OPTARG	The value of the last option argument processed by the **getopts** special command.
OPTIND	The index of the last option processed by the **getopts** special command.
PATH	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.
MAILCHECK	If this variable is set, it is interpreted as a value in seconds to wait between checks for new mail. The default is **600** (10 minutes). If the value is zero, mail is checked before each prompt.
MAILPATH	A colon-separated list of files that are checked for new mail. MAIL is ignored if this variable is set.
PS1	Primary prompt string, by default **'$ '**.
PS2	Secondary prompt string, by default **'> '**.
IFS	Internal field separators, normally space, tab, and newline.
LANG, **LC_ALL**	Locale variables.
LC_CTYPE	Affects the mapping of bytes to characters for file name generation, for the interpretation of '\', and for handling **$IFS**.
SHACCT	If this variable is set in the initial environment passed to the shell and points to a file writable by the user, accounting statistics are written to it.

TIMEOUT	The shell exists when prompting for input if no command is entered for more than the given value in seconds. A value of zero means no timeout and is the default.

Blank Interpretation

After parameter and command substitution, any results of substitution are scanned for internal field separator characters (those found in **$IFS**) and split into distinct arguments where such characters are found. Explicit null arguments (**""** or **''**) are retained. Implicit null arguments (those resulting from parameters that have no values) are 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.
[!...]	Matches any character except the enclosed ones.

Quoting

The following characters have a special meaning to the shell and cause termination of a 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 (**''**), except a single quote, are quoted. Inside double quotes (**""**) parameter and command substitution occurs and "\" quotes the characters ****, **`** **"** and **$**.

'**$***' is equivalent to '**$1 $2** ...', whereas '**$@**' is equivalent to '**"$1" "$2"**...'.

Prompting

When used interactively, the shell prompts with the value of **$PS1** before reading a command. If at any time a newline is typed and further input is needed to complete a command then the secondary prompt (**$PS2**) is issued.

Input and 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. 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, \newline is ignored, and \\ is used to quote the characters **\\ $ `** and the first character of *word*. The optional "**-**" causes leading tabulator character to be stripped from the resulting document; *word* may then also be prefixed by a tabulator.
<&*digit*	The standard input is duplicated from file descriptor *digit*. Similarly for the standard output using **>**.
<&-	The standard input is closed. Similarly for the standard output using **>**.

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**. If a command is followed by **&** then the default standard input for the command is the empty file (**/dev/null**), unless job control is enabled. Otherwise, the environment for the execution of a command contains the file descriptors of the invoking shell as modified by input output specifications.

Environment

The *environment* is a list of name-value pairs that is passed to an executed program in the same way as a normal argument list; see **exec** and **environ**. The shell interacts with the environment in several ways. On invocation, the shell scans the environment and creates a parameter for each name found, giving it the corresponding value. Executed commands inherit the same environment. If the user modifies the values of these parameters or creates new ones, none of these affects the environment unless the export command is used to bind the shell's parameter to the environment. The environment seen by any executed command is thus composed of any unmodified name-value pairs originally inherited by the shell, plus any modifications or additions, all of which must be noted in export commands.

The environment for any simple-command may be augmented by prefixing it with one or more assignments to parameters. Thus these two lines are equivalent:

```
TERM=450 cmd         (export TERM; TERM=450; cmd
args                 args)
```

Signals

The INTERRUPT and QUIT signals for an invoked command are ignored if the command is followed by **&** (unless job control is enabled); otherwise signals have the values inherited by the shell from its parent. See also **trap**.

Execution

Each time a command is executed the above substitutions are carried out. The shell then first looks if a function with the command name was defined; if so, it is chosen for execution. Otherwise, except for the 'special commands' listed below a new process is created and an attempt is made to execute the command via an **exec**.

The shell parameter **$PATH** defines the search path for the directory containing the command. Each alternative directory name is separated by a colon ("**:**"). The default path is '**/usr/sbin:/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 has execute permission but is not an **a.out** file, it is assumed to be a file containing shell commands. A subshell (i.e., a separate process) is spawned to read it. A parenthesized command is also executed in a subshell.

Special Commands

:	No effect; the command does nothing.
. *file*	Read and execute commands from file and return. The search path **$PATH** is used to find the directory containing *file*.
break [*n*]	Exit from the enclosing **for** or **while** loop, if any. If *n* is specified then break *n* levels.
continue [*n*]	Resume the next iteration of the enclosing **for** or **while** loop. If *n* is specified then resume at the *n*th enclosing loop.
cd [*arg*]	Change the current directory to *arg*. The shell parameter $HOME is the default arg. If no directory arg is found and the $CDPATH parameter contains a list of directories separated by colons, each of these directories is used as a prefix to arg in the given order, and the current directory is set to the first one that is found. If no suitable directory been found, an interactive shell may try to fix spelling errors and propose an alternative directory name: If the answer is ' ' or anything other than ' ', `cd /usr/lib? y` `cd /usf/lb` `ok` **y** **n** the shell will set the current directory to the one proposed.

echo [*arg* ...]	Each *arg* is printed to standard output; afterwards, a newline is printed. The following escape sequences are recognized in *arg*: **\b** — Prints a backspace character. **\c** — Causes the command to return immediately. Any following characters are ignored, and the terminating newline is not printed. **\f** — Prints a formfeed character. **\n** — Prints a newline character. **\r** — Prints a carriage-return character. **\t** — Prints a tabulator character. **\v** — Prints a vertical tabulator character. \\ — Prints a backslash character. **\0***nnn* — Prints the character (byte) with octal value *nnn*. If **/usr/ucb** precedes **/usr/sbin** or **/usr/bin** in the current setting of the **$PATH** variable and the first argument is **-n**, the terminating newline is not printed, and no escape sequences are recognized. If the **$SYSV3** variable is set in the initial environment passed to the shell, the **-n** argument is also interpreted, but escape sequences are processed as usual.
eval [*arg* ...]	The arguments are read as input to the shell and the resulting command(s) are executed.
exec [*arg* ...]	The command specified by the arguments is executed in place of this shell without creating a new process. Input/output arguments may appear and (if no other arguments are given) cause the shell input/output to be modified.
exit [*n*]	Causes the 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.
export [*name* ...]	The given names are marked for automatic export to the environment of subsequently-executed commands. If no arguments are given then a list of exportable names is printed.
getopts *optstring variable* [*arg* ...]	Retrieves options and option-arguments from *arg* (or the positional parameters) similar to **getopt**. *optstring* is a list of characters (bytes); each character represents an option letter. A character followed by "**:**" indicates that the option has an argument. Calling **getopts** repeatedly causes one option to be retrieved per call. The index of the current option is stored in the variable **OPTIND**; it is initialized to **1** when the shell starts. The option-argument, if any, is stored in the **OPTARG** variable. The option character is stored in the *variable*. When the end of the options is reached, **getopts** returns with a non-zero value. A missing argument or an illegal option also causes a non-zero return value, and an error message is printed to standard error.
hash [*name* ...]	The shell maintains a hash table of the locations of external commands. If name arguments are given, each one is looked up and is inserted into the table if it is found. Otherwise, a list of the commands currently in the table is printed.
newgrp [*arg* ...]	Equivalent to "**exec newgrp** *arg* ...".

pwd	Prints the name of the present working directory.
read [**-r**] *name* ...	One line is read from the standard input; successive words of the input are assigned to the variables name in order, with leftover words to the last variable. The return code is **0** unless end-of-file is encountered. Normally, backslashes escape the following character; this is inhibited if the **-r** option is given.
readonly [*name* ...]	The given names are marked readonly and the values of the these names may not be changed by subsequent assignment. If no arguments are given then a list of all readonly names is printed.
return [*n*]	Return from a shell function to the execution level above. With the argument *n*, the special variable **$?** is set to the given value.

set [**--aefhknptuvx** [*arg* ...]]

--	No effect; useful if the first *arg* begins with **-**.
-a	Export any variables that are modified or created from now on.
-e	If non-interactive, exit immediately if a command fails.
-f	File name generation is disabled.
-h	When a function is defined, look up all external commands it contains as described for the **hash** special command. Normally, these commands are looked up when they are executed.
-k	All keyword arguments are placed in the environment for a command, not just those that precede the command name.
-m	Enables job control (see below).
-n	Read commands but do not execute them.
-p	Makes the shell privileged. A privileged shell does not execute the system and user profiles; if an non-privileged shell (the default) has an effective user or group id different to its real user or group id or if it has an effective user or group id below 100, it resets its effective user or group id, respectively, to the corresponding real id at startup.
-t	Exit after reading and executing one command.
-u	Treat unset variables as an error when substituting.
-v	Print shell input lines as they are read.
-x	Print commands and their arguments as they are executed.
-	Turn off the **-x** and **-v** options.

These flags can also be used upon invocation of the shell. The current set of flags may be found in **$-**.

If **+** is used instead of **-**, the given flags are disabled.

Remaining arguments are positional parameters and are assigned, in order, to **$1**, **$2**, etc. If no arguments are given then the values of all names are printed.

shift [*n*]	The positional parameters from **$2**... are renamed **$1**... The *n* argument causes a shift by the given number, i.e. **$***n+1* is renamed to **$1** and so forth.
times	Print the accumulated user and system times for processes run from the shell.
test [*expr*]	**test** evaluates the expression *expr*, and if its value is true then it returns zero exit status; otherwise, a non-zero exit status is returned. **test** returns a non-zero exit if there are no arguments. The following primitives are used to construct *expr*:

-r *file*	true if the file exists and is readable.
-w *file*	true if the file exists and is writable.
-u *file*	true if the file exists and has the setuid bit set.
-g *file*	true if the file exists and has the setgid bit set.
-k *file*	true if the file exists and has the sticky bit set.
-f *file*	true if the file exists and is a regular file (or any file other than a directory if **/usr/ucb** occurs early in the current **$PATH** parameter).
-d *file*	true if the file exists and is a directory.
-h *file*	true if the file exists and is a symbolic link.
-L *file*	true if the file exists and is a symbolic link.
-p *file*	true if the file exists and is a named pipe.
-b *file*	true if the file exists and is a block device.
-c *file*	true if the file exists and is a character device.
-s *file*	true if the file exists and has a size greater than zero.
-t [*fildes*]	true if the open file whose file descriptor number is *fildes* (**1** by default) is associated with a terminal device.
-z *s1*	true if the length of string *s1* is zero.
-n *s1*	true if the length of the string *s1* is nonzero.
s1 **=** *s2*	true if the strings *s1* and *s2* are equal.
s1 **!=** *s2*	true if the strings *s1* and *s2* are not equal.
s1	true if *s1* is not the null string.
n1 **-eq** *n2*	true if the integers *n1* and *n2* are algebraically equal. Any of the comparisons **-ne**, **-gt**, **-ge**, **-lt**, or **-le** may be used in place of **-eq**.

These primaries may be combined with the following operators:

!	unary negation operator
-a	binary AND operator
-o	binary OR operator
(*expr* **)**	parentheses for grouping

-a has higher precedence than **-o**. Notice that all the operators and flags are separate arguments to **test**. Notice also that parentheses are meaningful as command separators and must be escaped.

trap [*arg*] [*n*|*name*] ...	*arg* is a command to be read and executed when the shell receives signal(s) *n*. Note that *arg* is scanned once when the trap is set and once when the trap is taken. **trap** commands are executed in order of signal number. If *arg* is absent then all trap(s) *n* 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, otherwise upon receipt of signal *n* as numbered in signal. Trap with no arguments prints a list of commands associated with each signal number. A symbolic name can be used instead of the *n* argument; it is formed by the signal name in the C language minus the **SIG** prefix, e.g. **TERM** for **SIGTERM**. **EXIT** is the same as a zero ('**0**') argument.
type *name* ...	For each *name*, prints if it would be executed as a shell function, as a special command, or as an external command. In the last case, the full path name to the command is also printed.
ulimit [**-**[**HS**] [**a**|**cdfmnstuv**]] **ulimit** [**-**[**HS**] [**c**|**d**|**f**|**m**|**n**|**s**|**t**|**u**|**v**]] [*limit*]	Handles resource limits for the shell and processes created by it, as described in **getrlimit**. Without a *limit* argument, the current settings are printed; otherwise, a new limit is set. The following options are accepted: (see options below)

-H	Sets a hard limit. Only the super-user may raise a hard limit.
-S	Sets a soft limit. A soft limit must not exceed the hard limit. If neither **-H** or **-S** is given, the soft limit is printed, or both limits are set, respectively.
-a	Chooses all limits described.
-c	The maximum size of a core dump in 512-byte blocks.
-d	The maximum size of the data segment in kbytes.
-f	The maximum size of a file in 512-byte blocks. This is the default if no limit is explicitly selected.
-l	The maximum size of locked memory in kbytes.
-m	The maximum resident set size in kbytes.
-n	The maximum number of open file descriptors.
-s	The maximum size of the stack segment in kbytes.
-t	The maximum processor time in seconds.
-u	The maximum number of child processes.
-v	The maximum address space size in kbytes.

umask [**-S**] [*nnn*]	The user file creation mask is set to the octal value *nnn* (see umask). Symbolic modes as described in chmod are also accepted. If *nnn* is omitted, the current value of the mask is printed. With the **-S** option, the current mask is printed as a symbolic string.
unset *variable* ...	Unsets each *variable* named.
wait [*n*]	Wait for the specified process and report its termination status. If *n* is not given then all currently active child processes are waited for. The return code from this command is that of the process waited for. If *n* does not refer to a child process of the shell, **wait** returns immediately with code **0**.

Invocation

If the first character of argument zero is **-**, commands are read from **/etc/profile** and **$HOME/.profile**, if the respective file exists. Commands are then read as described below. The following flags are interpreted by the shell when it is invoked:

-c *string*	If the **-c** flag is present then commands are read from *string*.
-s	If the **-s** flag is present or if no arguments remain then commands are read from the standard input. Shell output is written to file descriptor 2.
-i	If the **-i** flag is present or if the shell input and output are attached to a terminal (as told by the function **isatty()**) then this shell is interactive. In this case the terminate signal **SIGTERM** is ignored (so that '**kill 0**' does not kill an interactive shell) and the interrupt signal **SIGINT** is caught and ignored (so that **wait** is interruptable). In all cases **SIGQUIT** is ignored by the shell.

The remaining flags and arguments are described under the **set** command.

Job Control

When an interactive shell is invoked as **jsh**, job control is enabled. Job control allows to stop and resume processes, and to switch between foreground and background jobs. A job consists of the commands of a single pipeline. Each job is placed in a separate process group; a login shell and all jobs created by it form a session. Interrupt, quit, and other terminal control characters only affect the current foreground process group. The foreground job can be stopped pressing the suspend key, typically **^Z**; any job can be stopped by sending the **STOP** signal to it. Jobs are identified by jod ids of the following form:

%, **%%**, or **%+**	The current job.
%-	The job that was previously the current job.
?*string*	The only job whose name contains *string*.
%*number*	The job with the given number.
number	The job with process group id *number*.
string	The only job for which *string* is a prefix of its name.

The following built-in commands are additionally available with job control:

bg [*jobid* ...]	Places each *jobid* in the background. The default job id is the current job.

fg [*jobid* ...]	Sequentially selects each *jobid* as the foreground job. The default job id is the current job.
jobs [**-p**|**-l**] [*jobid* ...] | [**-x** *command* [*arguments* ...]]	Prints information about each *jobid*, or executes *command*: **-l** — Includes the process group id and the starting directory. **-p** — Includes the process group id. **-x** *command* [*arguments* ...] — Executes command with arguments; each argument that forms a job id is replaced by the process group id of the respective job. It is an error if a given job does not exist.
kill [[**-s** *signal* | **-***signal*] *jobid* ... | **-l** [*status*]	A special version of the **kill** command that recognizes job ids in its arguments.
stop *jobid* ...	Stops the given jobs (i.e. sends a **STOP** signal to them).
suspend	Stops the shell itself. This is not allowed if the shell is a session leader.
wait [*jobid*]	The **wait** command (see above) recognizes job ids in its arguments.

Notes

For historical reasons, **^** is a synonym for **|** as pipeline separator. Its use in new applications is therefore discouraged.

If a command other than a simple-command (i.e. '**for** ...', '**case** ...', etc.) is redirected, it is executed in a subshell. If variable assignments must be visible in the parent shell after the input has been redirected, the **exec** special command can be used:

```
exec 5<&0
<input
while read line
do
  ...
 variable=value
  ...
done
exec <&5 5<&-
```

If parameters that have been inherited from the initial environment are modified, they must be explicitly exported to make the change visible to external commands, as described under 'Environment' above.

The **$IFS** parameter is applied to any unquoted word. Thus:

```
IFS=X
echoXfoo
```

executes the '**echo**' command with the argument '**foo**'. The command '**set --**' without further arguments is a no-op (no operation). The shift special command can be used to delete all positional parameters.

There is only one namespace for both functions and parameters. A function definition will delete a parameter with the same name and vice-versa.

Parameter assignments that precede a special command affect the shell itself; parameter assignments that precede the call of a function are ignored.

Files

/etc/profile
$HOME/.profile
/tmp/sh*
/dev/null

Examples

```
sh
```

Invokes the Bourne shell, and places you at a command prompt.

Related commands

bc — A calculator.
init — The parent of all processes on the system.
kill — Send a signal to a process, affecting its behavior or killing it.
ksh — The Korn shell command interpreter.
login — Begin a session on a system.
newgrp — Log into a new group.
ps — Report the status of a process or processes.
pwd — Print the name of the working directory.
stty — Set options for your terminal display.