-- --

IEEE P1003.2 Draft 11.2 – September 1991

3.6.7 Quote Removal

The quote characters

`\ ' "`

(backslash, single-quote, double-quote) that were present in the original word shall be removed unless they have themselves been quoted.

3.7 Redirection

Redirection is used to open and close files for the current shell execution environment (see 3.12) or for any command. *Redirection operators* can be used with numbers representing file descriptors (see the definition in POSIX.1 {8}) as described below. See also 2.9.1. The relationship between these file descriptors and access to them in a programming language is specified in the language binding for that language to this standard.

The overall format used for redirection is:

[*n*]*redir-op* *word*

The number *n* is an optional decimal number designating the file descriptor number; it shall be delimited from any preceding text and immediately precede the redirection operator *redir-op*. If *n* is quoted, the number shall not be recognized as part of the redirection expression. (For example, `echo \2>a` writes the character `2` into file `a`). If any part of *redir-op* is quoted, no redirection expression shall be recognized. (For example, `echo 2\>a` writes the characters `2>a` to standard output.) The optional number, redirection operator, and *word* shall not appear in the arguments provided to the command to be executed (if any).

In this standard, open files are represented by decimal numbers starting with zero. It is implementation defined what the largest value can be; however, all implementations shall support at least 0 through 9 for use by the application. These numbers are called *file descriptors*. The values 0, 1, and 2 have special meaning and conventional uses and are implied by certain redirection operations; they are referred to as *standard input*, *standard output*, and *standard error*, respectively. Programs usually take their input from standard input, and write output on standard output. Error messages are usually written to standard error. The redirection operators can be preceded by one or more digits (with no intervening `<blank>`s allowed) to designate the file descriptor number.

If the redirection operator is `<<` or `<<-`, the word that follows the redirection operator shall be subjected to quote removal; it is unspecified whether any of the other expansions occur. For the other redirection operators, the word that follows the redirection operator shall be subjected to tilde expansion, parameter expansion, command substitution, arithmetic expansion, and quote removal. Pathname expansion shall not be performed on the word by a noninteractive shell; an interactive shell may perform it, but shall do so only when the expansion would result in one word.

If more than one redirection operator is specified with a command, the order of evaluation is from beginning to end.

In the following description of redirections, references are made to opening and creating files. These references shall conform to the requirements in 2.9.1.4. A failure to open or create a file shall cause the redirection to fail.

3.7.1 Redirecting Input

Input redirection shall cause the file whose name results from the expansion of *word* to be opened for reading on the designated file descriptor, or standard input if the file descriptor is not specified.

The general format for redirecting input is:

    [n]<word

where the optional *n* represents the file descriptor number. If the number is omitted, the redirection shall refer to standard input (file descriptor `0`).

3.7.2 Redirecting Output

The two general formats for redirecting output are:

    [n]>word
    [n]>|word

where the optional *n* represents the file descriptor number. If the number is omitted, the redirection shall refer to standard output (file descriptor `1`).

Output redirection using the > format shall fail if the *noclobber* option is set (see 1
the description of `set -C` in 3.14.11) and the file named by the expansion of *word* 1
exists and is a regular file. Otherwise, redirection using the > or >| formats shall 1
cause the file whose name results from the expansion of *word* to be created and opened for ouput on the designated file descriptor, or standard output if none is specified. If the file does not exist, it shall be created; otherwise, it shall be truncated to be an empty file after being opened.

3.7.3 Appending Redirected Output

Appended output redirection shall cause the file whose name results from the expansion of word to be opened for output on the designated file descriptor. The file is opened as if the POSIX.1 {8} *open*() function was called with the O_APPEND flag. If the file does not exist, it shall be created.

The general format for appending redirected output is as follows:

    [n]>>word

where the optional *n* represents the file descriptor number.

3.7.4 Here-Document

The redirection operators << and <<– both allow redirection of lines contained in a shell input file, known as a *here-document*, to the standard input of a command.

The here-document shall be treated as a single word that begins after the next `<newline>` and continues until there is a line containing only the delimiter, with no trailing `<blank>`s. Then the next here-document starts, if there is one. The format is as follows:

```
[n]<<word
        here-document
delimiter
```

If any character in *word* is quoted, the delimiter shall be formed by performing quote removal on *word*, and the here-document lines shall not be expanded. Otherwise, the delimiter shall be the *word* itself.

If no characters in *word* are quoted, all lines of the here-document shall be expanded for parameter expansion, command substitution, and arithmetic expansion. In this case, the backslash in the input shall behave as the backslash inside double-quotes (see 3.2.3). However, the double-quote character (`"`) shall not be treated specially within a here-document, except when the double-quote appears within `$( )`, `‘ ‘`, or `${ }`. 1

If the redirection symbol is <<–, all leading `<tab>` characters shall be stripped from input lines and the line containing the trailing delimiter. If more than one << or <<– operator is specified on a line, the here-document associated with the first operator shall be supplied first by the application and shall be read first by the shell.

3.7.5 Duplicating an Input File Descriptor

The redirection operator

[*n*]<&*word*

is used to duplicate one input file descriptor from another, or to close one. If *word* evaluates to one or more digits, the file descriptor denoted by *n*, or standard input if *n* is not specified, shall be made to be a copy of the file descriptor denoted by *word*; if the digits in *word* do not represent a file descriptor already open for input, a redirection error shall result (see 3.8.1). If *word* evaluates to –, file 1
descriptor *n*, or standard input if *n* is not specified, shall be closed. If *word* evaluates to something else, the behavior is unspecified.

3.7.6 Duplicating an Output File Descriptor

The redirection operator

[*n*]>&*word*

is used to duplicate one output file descriptor from another, or to close one. If *word* evaluates to one or more digits, the file descriptor denoted by *n*, or standard output if *n* is not specified, shall be made to be a copy of the file descriptor denoted by *word*; if the digits in *word* do not represent a file descriptor already open for output, a redirection error shall result (see 3.8.1). If *word* evaluates to −, 1
file descriptor *n*, or standard output if *n* is not specified, shall be closed. If *word* evaluates to something else, the behavior is unspecified.

3.7.7 Open File Descriptors for Reading and Writing.

The redirection operator

[*n*]<>*word*

shall cause the file whose name is the expansion of *word* to be opened for both reading and writing on the file descriptor denoted by *n*, or standard input if *n* is not specified. If the file does not exist, it shall be created.

3.7.8 Redirection Rationale. *(This subclause is not a part of P1003.2)*

In the C binding for POSIX.1 {8}, file descriptors are integers in the range 0 − ({OPEN_MAX}−1). The file descriptors discussed in Redirection are that same set of small integers.

As POSIX.2 is being finalized, it is not known how file descriptors will be represented in the language-independent description of POSIX.1 {8}. The current consensus appears to be that they will remain as small integers, but it is still possible that they will be defined as an opaque type. If they remain as integers, then the current POSIX.2 wording is acceptable. If they become an opaque type, then the C binding to POSIX.1 {8} will have to define the mapping from the binding's small integers to the opaque type, and the Redirection clause in POSIX.2 will have to be modified to specify that same mapping.

Having multidigit file descriptor numbers for I/O redirection can cause some obscure compatibility problems. Specifically, scripts that depend on an example command:

```
echo 22>/dev/null
```

echoing `"2"` are somewhat broken to begin with. However, the file descriptor number still must be delimited from the preceding text. For example,

```
cat file2>foo
```

will write the contents of `file2`, not the contents of `file`.

The >| format of output redirection was adopted from the KornShell. Along with the *noclobber* option, `set −C`, it provides a safety feature to prevent inadvertent overwriting of existing files. (See the rationale with the `pathchk` utility for why this step was taken.) The restriction on regular files is historical practice.

The System V shell and the KornShell have differed historically on pathname expansion of *word* ; the former never performed it, the latter only when the result was a single field (file). As a compromise, it was decided that the KornShell functionality was useful, but only as a shorthand device for interactive users. No reasonable shell script would be written with a command such as:

```
cat foo > a*
```

Thus, shell scripts are prohibited from doing it, while interactive users can select the shell with which they are most comfortable.

The construct `2>&1` is often used to redirect standard error to the same file as standard output. Since the redirections take place beginning to end, the order of redirections is significant. For example:

```
ls > foo 2>&1
```

directs both standard output and standard error to file `foo`. However

```
ls 2>&1 > foo
```

only directs standard output to file `foo` because standard error was duplicated as standard output before standard output was directed to file `foo`.

The <> operator is a feature first documented in the KornShell, but it has been silently present in both System V and BSD shells. It could be useful in writing an application that worked with several terminals, and occasionally wanted to start up a shell. That shell would in turn be unable to run applications that run from an ordinary controlling terminal unless it could make use of <> redirection. The 1
specific example is a historical version of the pager `more`, which reads from standard error to get its commands, so standard input and standard output are both available for their usual usage. There is no way of saying the following in the shell without <>:

```
cat food | more - >/dev/tty03 2<>/dev/tty03
```

Another example of <> is one that opens `/dev/tty` on file descriptor 3 for reading and writing:

```
exec 3<> /dev/tty
```

An example of creating a lock file for a critical code region:

```
set -C
until   2> /dev/null > lockfile
do      sleep 30
done
set +C
perform critical function
rm lockfile
```

Since `/dev/null` is not a regular file, no error is generated by redirecting to it in *noclobber* mode.

The case of a missing delimiter at the end of a here-document is not specified. This is considered an error in the script (one that sometimes can be difficult to diagnose), although some systems have treated end-of-file as an implicit delimiter.

Tilde expansion is not performed on a here-document because the data is treated 1
as if it were enclosed in double-quotes. 1

3.8 Exit Status and Errors

3.8.1 Consequences of Shell Errors

For a noninteractive shell, an error condition encountered by a special built-in (see 3.14) or other type of utility shall cause the shell to write a diagnostic message to standard error and exit as shown in the following table:

	Special Built-in	**Other Utilities**
Shell language syntax error	shall exit	shall exit
Utility syntax error (option or operand error)	shall exit	shall not exit
Redirection error	shall exit	shall not exit
Variable assignment error	shall exit	shall not exit
Expansion error	shall exit	shall exit
Command not found	n/a	may exit
`dot` script not found	shall exit	n/a

An ‘‘expansion error’’ is one that occurs when the shell expansions defined in 3.6 are carried out (e.g., `${x!y}`, because `!` is not a valid operator); an implementation may treat these as syntax errors if it is able to detect them during tokenization, rather than during expansion.

If any of the errors shown as ‘‘shall (may) exit’’ occur in a subshell, the subshell shall (may) exit with a nonzero status, but the script containing the subshell shall not exit because of the error.

In all of the cases shown in the table, an interactive shell shall write a diagnostic message to standard error without exiting.

3.8.2 Exit Status for Commands

Each command has an exit status that can influence the behavior of other shell commands. The exit status of commands that are not utilities are documented in this subclause. The exit status of the standard utilities are documented in their respective clauses.