-- --

IEEE P1003.2 Draft 11.2 – September 1991

redirection operator is handled as part of the grammar.)

The start symbol of the grammar (`complete_command`) represents either input from the command line or a shell script. It is repeatedly applied by the interpreter to its input, and represents a single ‘‘chunk’’ of that input as seen by the interpreter.

The processing of here-documents is handled as part of token recognition (see 3.3) rather than as part of the grammar.

3.11 Signals and Error Handling

When a command is in an asynchronous list, the shell shall prevent SIGQUIT and SIGINT signals from the keyboard from interrupting the command. Otherwise, signals shall have the values inherited by the shell from its parent (see also 3.14.13).

When a signal for which a trap has been set is received while the shell is waiting for the completion of a utility executing a foreground command, the trap associated with that signal shall not be executed until after the foreground command has completed. When the shell is waiting, by means of the `wait` utility, for asynchronous commands to complete, the reception of a signal for which a trap has been set shall cause the `wait` utility to return immediately with an exit status >128, immediately after which the trap associated with that signal shall be taken.

If multiple signals are pending for the shell for which there are associated trap actions (see 3.14.13), the order of execution of trap actions is unspecified.

3.12 Shell Execution Environment

A shell execution environment consists of the following:

— Open files inherited upon invocation of the shell, plus open files controlled by `exec`.

— Working Directory as set by `cd` (see 4.5).

— File Creation Mask set by `umask` (see 4.67).

— Current traps set by `trap` (see 3.14.13).

— Shell parameters that are set by variable assignment (see `set` in 3.14.11) or from the POSIX.1 {8} environment inherited by the shell when it begins (see `export` in 3.14.8).

— Shell functions (see 3.9.5.)

— Options turned on at invocation or by `set`.

— Process IDs of the last commands in asynchronous lists known to this shell environment; see 3.9.3.1.

Utilities other than the special built-ins (see 3.14) shall be invoked in a separate environment that consists of the following. The initial value of these objects shall be the same as that for the parent shell, except as noted below.

- Open files inherited on invocation of the shell, open files controlled by the `exec` special built-in (see 3.14.6), plus any modifications and additions specified by any redirections to the utility.
- Current working directory.
- File creation mask.
- If the utility is a shell script, traps caught by the shell shall be set to the default values and traps ignored by the shell shall be set to be ignored by the utility. If the utility is not a shell script, the trap actions (default or ignore) shall be mapped into the appropriate signal handling actions for the utility.
- Variables with the `export` attribute, along with those explicitly exported for the duration of the command, shall be passed to the utility as POSIX.1 {8} environment variables.

The environment of the shell process shall not be changed by the utility unless explicitly specified by the utility description (for example, `cd` and `umask`).

A subshell environment shall be created as a duplicate of the shell environment, except that signal traps set by that shell environment shall be set to the default values. Changes made to the subshell environment shall not affect the shell environment. Command substitution, commands that are grouped with parentheses, and asynchronous lists shall be executed in a subshell environment. Additionally, each command of a multicommand pipeline is in a subshell environment; as an extension, however, any or all commands in a pipeline may be executed in the current environment. All other commands shall be executed in the current shell environment.

3.12.0.1 Shell Execution Environment Rationale. *(This subclause is not a part of P1003.2)*

Some systems have implemented the last stage of a pipeline in the current environment so that commands such as

command `| read foo`

set variable `foo` in the current environment. It was decided to allow this extension, but not require it; therefore, a shell programmer should consider a pipeline to be in a subshell environment, but not depend on it.

The previous description of execution environment failed to mention that each command in a multiple command pipeline could be in a subshell execution environment. For compatibility with some existing shells, the wording was phrased to allow an implementation to place any or all commands of a pipeline in the current environment. However, this means that a POSIX application must assume each command is in a subshell environment, but not depend on it.

The wording about shell scripts is meant to convey the fact that describing ‘‘trap actions’’ can only be understood in the context of the shell command language. Outside this context, such as in a C-language program, signals are the operative condition, not traps.

3.13 Pattern Matching Notation

The pattern matching notation described in this clause is used to specify patterns for matching strings in the shell. Historically, pattern matching notation is related to, but slightly different from, the regular expression notation described in 2.8. For this reason, the description of the rules for this pattern matching notation are based on the description of regular expression notation.

3.13.0.1 Pattern Matching Notation Rationale. *(This subclause is not a part of P1003.2)*

Pattern matching is a simpler concept and has a simpler syntax than regular expressions, as the former is generally used for the manipulation of file names, which are relatively simple collections of characters, while the latter is generally used to manipulate arbitrary text strings of potentially greater complexity. However, some of the basic concepts are the same, so this clause points liberally to the detailed descriptions in 2.8.

3.13.1 Patterns Matching a Single Character

The following *patterns matching a single-character* match a single character: *ordinary characters*, *special pattern characters*, and *pattern bracket expressions*. The pattern bracket expression also shall match a single collating element.

An ordinary character is a pattern that shall match itself. It can be any character
in the supported character set except for NUL, those special shell characters in 3.2 1
that require quoting, and the following three special pattern characters. Match- 1
ing shall be based on the bit pattern used for encoding the character, not on the 1
graphic representation of the character. If any character (ordinary, shell special, 1
or pattern special) is quoted, that pattern shall match the character itself. The 1
shell special characters always require quoting. 1

When unquoted and outside a bracket expression, the following three characters 1
shall have special meaning in the specification of patterns: 1

- `?` A question-mark is a pattern that shall match any character.
- `*` An asterisk is a pattern that shall match multiple characters, as described in 3.13.2.
- `[` The open bracket shall introduce a pattern bracket expression.

The description of basic regular expression bracket expressions in 2.8.3.2 also shall apply to the pattern bracket expression, except that the exclamation-mark