-- --

IEEE P1003.2 Draft 11.2 − September 1991

achieve the desired result:

```
unset bar
foo="${bar-\}}"
```

The only alternative to this compromise between shells would be to make the behavior unspecified whenever the literal characters ', {, }, and " appear within `${...}`. To write a portable script that uses these values, a user would have to assign variables, say,

```
squote=\' dquote=\" lbrace='{' rbrace='}'
${foo-$squote$rbrace$squote}
```

rather than

```
${foo-"'}'"}
```

Some systems have allowed the end of the word to terminate the backquoted command substitution, such as in

```
"`echo hello"
```

This usage is undefined in POSIX.2, where the matching backquote is required. The other undefined usage can be illustrated by the example:

```
sh -c '` echo "foo`'
```

The description of the recursive actions involving command substitution can be illustrated with an example. Upon recognizing the introduction of command substitution, the shell must parse input (in a new context), gathering the ‘‘source’’ for the command substitution until an unbalanced `)` or `` ` `` is located. For example, in the following

```
echo "$(date; echo "
        one" )"
```

the double-quote following the `echo` does not terminate the first double-quote; it is part of the command substitution ‘‘script.’’ Similarly, in

```
echo "$(echo *)"
```

the asterisk is not quoted since it is inside command substitution; however,

```
echo "$(echo "*")"
```

is quoted (and represents the asterisk character itself).

3.3 Token Recognition

The shell reads its input in terms of lines from a file, from a terminal in the case of an interactive shell, or from a string in the case of `sh -c` or *system* (). The 1
input lines can be of unlimited length. These lines are parsed using two major 1
modes: ordinary token recognition and processing of here-documents.

When an `io_here` token has been recognized by the grammar (see 3.10), one or more of the immediately subsequent lines form the body of one or more here-documents and shall be parsed according to the rules of 3.7.4.

When it is not processing an io_here, the shell shall break its input into tokens by applying the first applicable rule below to the next character in its input. The token shall be from the current position in the input until a token is delimited according to one of the rules below; the characters forming the token are exactly those in the input, including any quoting characters. If it is indicated that a token is delimited, and no characters have been included in a token, processing shall continue until an actual token is delimited.

(1) If the end of input is recognized, the current token shall be delimited. If there is no current token, the end-of-input indicator shall be returned as the token.

(2) If the previous character was used as part of an operator and the current character is not quoted and can be used with the current characters to form an operator, it shall be used as part of that (operator) token.

(3) If the previous character was used as part of an operator and the current character cannot be used with the current characters to form an operator, the operator containing the previous character shall be delimited.

(4) If the current character is backslash, single-quote, or double-quote (\, ', or ") and it is not quoted, it shall affect quoting for subsequent character(s) up to the end of the quoted text. The rules for quoting are as described in 3.2. During token recognition no substitutions shall be actually performed, and the result token shall contain exactly the characters that appear in the input (except for <newline> joining), unmodified, including any embedded or enclosing quotes or substitution operators, between the quote mark and the end of the quoted text. The token shall not be delimited by the end of the quoted field.

(5) If the current character is an unquoted $ or `, the shell shall identify the start of any candidates for parameter expansion (3.6.2), command substitution (3.6.3), or arithmetic expansion (3.6.4) from their introductory unquoted character sequences: $ or ${, $( or `, and $((, respectively. The shell shall read sufficient input to determine the end of the unit to be expanded (as explained in the cited subclauses). While processing the characters, if instances of expansions or quoting are found nested within the substitution, the shell shall recursively process them in the manner specified for the construct that is found. The characters found from the beginning of the substitution to its end, allowing for any recursion necessary to recognize embedded constructs, shall be included unmodified in the result token, including any embedded or enclosing substitution operators or quotes. The token shall not be delimited by the end of the substitution.

(6) If the current character is not quoted and can be used as the first character of a new operator, the current token (if any) shall be delimited. The current character shall be used as the beginning of the next (operator) token.

(7) If the current character is an unquoted `<newline>`, the current token shall be delimited.

(8) If the current character is an unquoted `<blank>`, any token containing the previous character is delimited and the current character is discarded.

(9) If the previous character was part of a word, the current character is appended to that word.

(10) If the current character is a #, it and all subsequent characters up to, but excluding, the next `<newline>` are discarded as a comment. The `<newline>` that ends the line is not considered part of the comment.

(11) The current character is used as the start of a new word.

Once a token is delimited, it shall be categorized as required by the grammar in 3.10.

3.3.1 Token Recognition Rationale. *(This subclause is not a part of P1003.2)*

The (3) rule about combining characters to form operators is not meant to pre- 1
clude systems from extending the shell language when characters are combined in 1
otherwise invalid ways. Portable applications cannot use invalid combinations 1
and test suites should not penalize systems that take advantage of this fact. For 1
example, the unquoted combination |& is not valid in a POSIX.2 script, but has a 1
specific KornShell meaning. 1

The (10) rule about # as the current character is the first in the sequence in which a new token is being assembled. The # starts a comment only when it is at the beginning of a token. This rule is also written to indicate that the search for the end-of-comment does not consider escaped `<newline>` specially, so that a comment cannot be continued to the next line.

3.4 Reserved Words

Reserved words are words that have special meaning to the shell. (See 3.9.) The following words shall be recognized as reserved words:

`!`	`elif`	`fi`	`in`	`while`
`case`	`else`	`for`	`then`	`{`[4]
`do`	`esac`	`if`	`until`	`}`
`done`				

4) In some historical systems, the curly braces are treated as control operators. To assist in future standardization activities, portable applications should avoid using unquoted braces to represent the characters themselves. It is possible that a future version of POSIX.2 may require this, although probably not for the often-used `find {}` construct.