IEEE P1003.2 Draft 11.2 – September 1991

Other invalid constructs might be deemed worthy of a warning but otherwise cause some reasonable behavior. Still other constructs may be very difficult to detect in some implementations. Also, different implementations might detect a given error during an initial parsing of the program (before reading any input files) while others might detect it when executing the program after reading some input. Implementors should be aware that diagnosing errors as early as possible and producing useful diagnostics can ease debugging of applications, and thus make an implementation more usable.

The unspecified behavior from using multicharacter `RS` values is to allow possible future extensions based on regular expressions used for record separators. Historical implementations take the first character of the string and ignore the others.

The undefined behavior resulting from NULs in regular expressions allows future extensions for the GNU `gawk` program to process binary data.

Unspecified behavior when `split(string,array,<null>)` is used is to allow a proposed future extension that would split up a string into an array of individual characters.

4.2 `basename` — Return nondirectory portion of pathname

4.2.1 Synopsis

`basename` *string* [*suffix*]

4.2.2 Description

The *string* operand shall be treated as a pathname, as defined in 2.2.2.102. The string *string* shall be converted to the filename corresponding to the last pathname component in *string* and then the suffix string *suffix* , if present, shall be removed. This shall be done by performing actions equivalent to the following steps in order:

(1) If *string* is `//`, it is implementation defined whether steps (2) through (5) are skipped or processed.

(2) If *string* consists entirely of slash characters, *string* shall be set to a single slash character. In this case, skip steps (3) through (5).

(3) If there are any trailing slash characters in *string* , they shall be removed.

(4) If there are any slash characters remaining in *string* , the prefix of *string* up to and including the last slash character in *string* shall be removed.

(5) If the *suffix* operand is present, is not identical to the characters remaining in *string* , and is identical to a suffix of the characters remaining in *string* , the suffix *suffix* shall be removed from *string* . Otherwise, *string*

shall not be modified by this step. It shall not be considered an error if *suffix* is not found in *string* .

The resulting string shall be written to standard output.

4.2.3 Options

None.

4.2.4 Operands

The following operands shall be supported by the implementation:

string A string.

suffix A string.

4.2.5 External Influences

4.2.5.1 Standard Input

None.

4.2.5.2 Input Files

None.

4.2.5.3 Environment Variables

The following environment variables shall affect the execution of `basename`:

LANG This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6.

LC_ALL This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.

LC_CTYPE This variable shall determine the locale for the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments).

LC_MESSAGES This variable shall determine the language in which messages should be written.

4.2.5.4 Asynchronous Events

Default.

4.2.6 External Effects

4.2.6.1 Standard Output

The `basename` utility shall write a line to the standard output in the following format:

`"%s\n"`, *<resulting string>*

4.2.6.2 Standard Error

Used only for diagnostic messages.

4.2.6.3 Output Files

None.

4.2.7 Extended Description

None.

4.2.8 Exit Status

The `basename` utility shall exit with one of the following values:

- 0 Successful completion.
- >0 An error occurred.

4.2.9 Consequences of Errors

Default.

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

Examples, Usage

If the string *string* is a valid pathname,

```
$(basename "string")
```

produces a filename that could be used to open the file named by *string* in the directory returned by

```
$(dirname "string")
```

If the string *string* is not a valid pathname, the same algorithm is used, but the result need not be a valid filename. The `basename` utility is not expected to make any judgements about the validity of *string* as a pathname; it just follows the specified algorithm to produce a result string.

The following shell script compiles `/usr/src/cmd/cat.c` and moves the output to a file named `cat` in the current directory when invoked with the argument `/usr/src/cmd/cat` or with the argument `/usr/src/cmd/cat.c`:

```
c89 $(dirname "$1")/$(basename "$1" .c).c
mv a.out $(basename "$1" .c)
```

History of Decisions Made

The POSIX.1 {8} definition of pathname allows trailing slashes on a pathname naming a directory. Some historical implementations have not allowed trailing slashes and thus treated pathnames of this form in other ways. Existing implementations also differ in their handling of *suffix* when *suffix* matches the entire string left after removing the directory part of *string*.

The behaviors of `basename` and `dirname` in this standard have been coordinated so that when *string* is a valid pathname

```
$(basename "string")
```

would be a valid filename for the file in the directory

```
$(dirname "string")
```

This would not work for the versions of these utilities in earlier drafts due to the way it specified handling of trailing slashes.

Since the definition of *pathname* in 2.2.2.102 specifies implementation-defined behavior for pathnames starting with two slash characters, Draft 11 has been changed to specify similar implementation-defined behavior for the `basename` and `dirname` utilities. On implementations where the pathname `//` is always treated the same as the pathname `/`, the functionality required by Draft 10 meets all of the Draft 11 requirements.

4.3 `bc` — Arbitrary-precision arithmetic language

4.3.1 Synopsis

`bc [−l]` [*file* . . .]

4.3.2 Description

The `bc` utility shall implement an arbitrary precision calculator. It shall take input from any files given, then read from the standard input. If the standard input and standard output to `bc` are attached to a terminal, the invocation of `bc` shall be considered to be *interactive* , causing behavioral constraints described in the following subclauses.

4.3.3 Options

The `bc` utility shall conform to the utility argument syntax guidelines described in 2.10.2.

The following option shall be supported by the implementation:

`−l` (The letter ell.) Define the math functions and initialize `scale` to 20, instead of the default zero. See 4.3.7.

4.3.4 Operands

The following operands shall be supported by the implementation:

file A pathname of a text file containing `bc` program statements. After all *file* s have been read, `bc` shall read the standard input.

4.3.5 External Influences

4.3.5.1 Standard Input

See Input Files.

4.3.5.2 Input Files

Input files shall be text files containing a sequence of comments, statements, and function definitions that shall be executed as they are read.