IEEE P1003.2 Draft 11.2 – September 1991

4.12 `command` — Execute a simple command

4.12.1 Synopsis

`command [−p]` *command_name* [*argument* . . .]

4.12.2 Description

The `command` utility shall cause the shell to treat the arguments as a simple com-
mand, suppressing the shell function lookup that is described in 3.9.1.1 item 1
(1)(b). 1

If the *command_name* is the same as the name of one of the special built-in utilities, the special properties in the enumerated list at the beginning of 3.14 shall not occur. In every other respect, if *command_name* is not the name of a function, the effect of `command` shall be the same as omitting `command`.

4.12.3 Options

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

The following option shall be supported by the implementation:

`−p` Perform the command search using a default value for **PATH** that is guaranteed to find all of the standard utilities.

4.12.4 Operands

The following operands shall be supported by the implementation:

argument One of the strings treated as an argument to *command_name* .

command_name
The name of a utility or a special built-in utility.

4.12.5 External Influences

4.12.5.1 Standard Input

None.

4.12.5.2 Input Files

None.

4.12.5.3 Environment Variables

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

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).

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

PATH This variable shall determine the search path used during the command search described in 3.9.1.1, except as described under the `−p` option.

4.12.5.4 Asynchronous Events

Default.

4.12.6 External Effects

4.12.6.1 Standard Output

None.

4.12.6.2 Standard Error

Used only for diagnostic messages.

4.12.6.3 Output Files

None.

4.12.7 Extended Description

None.

4.12.8 Exit Status

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

126 The utility specified by *command_name* was found but could not be 1
invoked. 1

127 An error occurred in the `command` utility or the utility specified by 1
command_name could not be found. 1

Otherwise, the exit status of `command` shall be that of the simple command specified by the arguments to `command`.

4.12.9 Consequences of Errors

Default.

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

Examples, Usage

The order for command search in POSIX.2 allows functions to override regular built-ins and path searches. This utility is necessary to allow functions that have the same name as a utility to call the utility (instead of a recursive call to the function).

The system default path is available using `getconf`; however, since `getconf` may need to have the **PATH** set up before it can be called itself, the following can be used:

```
command -p getconf _CS_PATH
```

Since `command` appears in Table 2-2, it will always be found prior to the **PATH** search.

There is nothing in the description of `command` that implies the command line is parsed any differently than for any other simple command. For example,

```
command a | b ; c
```

is not parsed in any special way that causes `|` or `;` to be treated other than a pipe operator or semicolon or that prevents function lookup on `b` or `c`.

Examples : Make a version of `cd` that always prints out the new working directory exactly once:

```
cd() {
        command cd "$@" >/dev/null
        pwd
}
```

Start off a ‘‘secure shell script’’ in which the script avoids being spoofed by its parent:

```
IFS='
'
#       The preceding value should be <space><tab><newline>.
#       Set IFS to its default value.                          1

\unset -f command
#       Ensure command is not a user function.
#       Note that unset is escaped to prevent an alias being used
#       for unset on implementations that support aliases.

PATH="$(\command -p getconf _CS_PATH):$PATH"
#       Put on a reliable PATH prefix.

#       Now, unset all utility names that will be used (or
#       invoke them with \command each time).
#       . . .
```

At this point, given correct permissions on the directories called by **PATH** , the script has the ability to ensure that any utility it calls is the intended one. It is being very cautious because it assumes that implementation extensions may be present that would allow user aliases and/or functions to exist when it is invoked; neither capability is specified by POSIX.2, but neither is prohibited as an extension. For example, the proposed UPE supplement to POSIX.2 introduces a **ENV** variable that precedes the invocation of the script with a user startup script. Such a script could have used the aliasing facility from the UPE or the functions in POSIX.2 to spoof the application.

The `command`, `env`, `nohup`, and `xargs` utilities have been specified to use exit
code 127 if an error occurs so that applications can distinguish ‘‘failure to find a 1
utility’’ from ‘‘invoked utility exited with an error indication.’’ The value 127 was 1
chosen because it is not commonly used for other meanings; most utilities use
small values for ‘‘normal error conditions’’ and the values above 128 can be con-
fused with termination due to receipt of a signal. The value 126 was chosen in a 1
similar manner to indicate that the utility could be found, but not invoked. Some 1
scripts produce meaningful error messages differentiating the 126 and 127 cases. 1
The distinction between exit codes 126 and 127 is based on KornShell practice 2
that uses 127 when all attempts to *exec* the utility fail with [ENOENT], and uses 2
126 when any attempt to *exec* the utility fails for any other reason. 2

History of Decisions Made

The `command` utility is somewhat similar to the Eighth Edition `builtin` command, but since `command` also goes to the file system to search for utilities, the name `builtin` would not be intuitive.

The `command` utility will most likely be provided as a regular built-in. In an earlier draft, it was a special built-in. This was changed for the following reasons:

— The removal of exportable functions made the special precedence of a special built-in unnecessary.

— A special built-in has special properties (see the enumerated list at the beginning of 3.14) that were inappropriate for invoking other utilities. For example, two commands such as

`date >` *unwritable-file*

`command date >` *unwritable-file*

would have entirely different results; in a noninteractive script, the former would continue to execute the next command, the latter would abort. Introducing this semantic difference along with suppressing functions was seen to be nonintuitive.

— There are some advantages of suppressing the special characteristics of special built-ins on occasion. For example:

`command exec >` *unwritable-file*

will not cause a noninteractive script to abort, so that the output status can be checked by the script.

An earlier draft presented a larger number of options. Most were removed because they were not useful to real portable applications, given the new command search order.

The `−p` option is present because it is useful to be able to ensure a safe path search that will find all the POSIX.2 standard utilities. This search might not be identical to the one that occurs through one of the POSIX.1 {8} *exec* functions when **PATH** is unset, as explained in 2.6.1. At the very least, this feature is required to allow the script to access the correct version of `getconf` so that the value of the default path can be accurately retrieved.

4.13 cp — Copy files

4.13.1 Synopsis

`cp [−fip]` *source_file target_file* 2

`cp [−fip]` *source_file ... target* 2

`cp −R [−fip]` *source_file ... target* 2

`cp −r [−fip]` *source_file ... target* 2

4.13.2 Description

The first synopsis form is denoted by two operands, neither of which are existing files of type directory. The `cp` utility shall copy the contents of *source_file* to the destination path named by *target_file.*

The second synopsis form is denoted by two or more operands where the `−R` or `−r` options are not specified and the first synopsis form is not applicable. It shall be an error if any *source_file* is a file of type directory, if *target* does not exist, or if *target* is a file of a type defined by POSIX.1 {8}, but is not a file of type directory. The `cp` utility shall copy the contents of each *source_file* to the destination path named by the concatenation of *target,* a slash character, and the last component of *source_file.*

The third and fourth synopsis forms are denoted by two or more operands where the `−R` or `−r` options are specified. The `cp` utility shall copy each file in the file hierarchy rooted in each *source_file* to a destination path named as follows.

If *target* exists and is a file of type directory, the name of the corresponding destination path for each file in the file hierarchy shall be the concatenation of *target,* a slash character, and the pathname of the file relative to the directory containing *source_file.*

If *target* does not exist, and two operands are specified, the name of the corresponding destination path for *source_file* shall be *target;* the name of the corresponding destination path for all other files in the file hierarchy shall be the concatenation of *target,* a slash character, and the pathname of the file relative to *source_file.*

It shall be an error if *target* does not exist and more than two operands are specified, or if *target* exists and is a file of a type defined by POSIX.1 {8}, but is not a file of type directory.

In the following description, *source_file* refers to the file that is being copied, whether specified as an operand or a file in a file hierarchy rooted in a *source_file* operand. The term *dest_file* refers to the file named by the destination path.

For each *source_file,* the following steps shall be taken: