Full credit is given to the above companies including the OS that this PDF file was generated!

Red Hat Enterprise Linux Release 9.2 Manual Pages on 'command.1p' command

$ man command.1p

COMMAND(1P) POSIX Programmer's Manual COMMAND(1P)

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

command ? execute a simple command

SYNOPSIS

command [-p] command_name [argument...]

command [-p][-v|-V] command_name

DESCRIPTION

The command utility shall cause the shell to treat the arguments as a simple command, suppressing the shell function lookup that is described in Section 2.9.1.1, Command Search and Execution, item 1b.

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 Section 2.14, Special Built-In Utilities shall not oc? cur. In every other respect, if command_name is not the name of a func? tion, the effect of command (with no options) shall be the same as omitting command.

When the -v or -V option is used, the command utility shall provide in? formation concerning how a command name is interpreted by the shell.

OPTIONS

The command utility shall conform to the Base Definitions volume of POSIX.1?2017, Section 12.2, Utility Syntax Guidelines.

The following options shall be supported:

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

-v Write a string to standard output that indicates the pathname or command that will be used by the shell, in the current shell execution environment (see Section 2.12, Shell Execu? tion Environment), to invoke command_name, but do not invoke command_name.

* Utilities, regular built-in utilities, command_names in? cluding a <slash> character, and any implementation-de? fined functions that are found using the PATH variable (as described in Section 2.9.1.1, Command Search and Exe? cution), shall be written as absolute pathnames.
* Shell functions, special built-in utilities, regular built-in utilities not associated with a PATH search, and shell reserved words shall be written as just their names.
* An alias shall be written as a command line that repre? sents its alias definition.
* Otherwise, no output shall be written and the exit status shall reflect that the name was not found.

-V Write a string to standard output that indicates how the name given in the command_name operand will be interpreted by the shell, in the current shell execution environment (see Sec? tion 2.12, Shell Execution Environment), but do not invoke command_name. Although the format of this string is unspeci? fied, it shall indicate in which of the following categories command_name falls and shall include the information stated:

* Utilities, regular built-in utilities, and any implemen? tation-defined functions that are found using the PATH

variable (as described in Section 2.9.1.1, Command Search and Execution), shall be identified as such and include the absolute pathname in the string.

* Other shell functions shall be identified as functions.
* Aliases shall be identified as aliases and their defini? tions included in the string.
* Special built-in utilities shall be identified as special built-in utilities.
* Regular built-in utilities not associated with a PATH search shall be identified as regular built-in utilities. (The term ``regular'' need not be used.)
* Shell reserved words shall be identified as reserved words.

OPERANDS

The following operands shall be supported:

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.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of com? mand:

LANG Provide a default value for the internationalization vari? ables that are unset or null. (See the Base Definitions vol? ume of POSIX.1?2017, Section 8.2, Internationalization Vari? ables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of

bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES

Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error and informative messages written to standard output.

NLSPATH Determine the location of message catalogs for the processing of LC_MESSAGES.

PATH Determine the search path used during the command search de? scribed in Section 2.9.1.1, Command Search and Execution, ex? cept as described under the -p option.

ASYNCHRONOUS EVENTS

Default.

STDOUT

When the -v option is specified, standard output shall be formatted as:

"%s\n", <pathname or command>

When the -V option is specified, standard output shall be formatted as:

"%s\n", <unspecified>

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

When the -v or -V options are specified, the following exit values shall be returned:

0 Successful completion.

>0 The command_name could not be found or an error occurred.

Otherwise, the following exit values shall be returned:

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

127 An error occurred in the command utility or the utility specified

by command_name could not be found.

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

CONSEQUENCES OF ERRORS

Default.

The following sections are informative.

APPLICATION USAGE

The order for command search allows functions to override regular built-ins and path searches. This utility is necessary to allow func? tions 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 get? conf may need to have the PATH set up before it can be called itself, the following can be used:

```
command -p getconf PATH
```

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

```
command exec > unwritable-file
```

does not cause a non-interactive script to abort, so that the output status can be checked by the script.

The command, env, nohup, time, and xargs utilities have been specified to use exit code 127 if an error occurs so that applications can dis? tinguish ``failure to find a utility'' from ``invoked utility exited with an error indication''. The value 127 was 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 confused with termination due to receipt of a signal. The value 126 was chosen in a similar manner to indicate that the utility could be found, but not invoked. Some scripts produce meaningful error messages differenti? ating the 126 and 127 cases. The distinction between exit codes 126 and 127 is based on KornShell practice that uses 127 when all attempts to exec the utility fail with [ENOENT], and uses 126 when any attempt to exec the utility fails for any other reason.

Since the -v and -V options of command produce output in relation to the current shell execution environment, command is generally provided as a shell regular built-in. If it is called in a subshell or separate utility execution environment, such as one of the following:

(PATH=foo command -v)

nohup command -v

it does not necessarily produce correct results. For example, when called with nohup or an exec function, in a separate utility execution environment, most implementations are not able to identify aliases, functions, or special built-ins.

Two types of regular built-ins could be encountered on a system and these are described separately by command. The description of command search in Section 2.9.1.1, Command Search and Execution allows for a standard utility to be implemented as a regular built-in as long as it is found in the appropriate place in a PATH search. So, for example, command -v true might yield /bin/true or some similar pathname. Other implementation-defined utilities that are not defined by this volume of POSIX.1?2017 might exist only as built-ins and have no pathname associ? ated with them. These produce output identified as (regular) built-ins. Applications encountering these are not able to count on execing them, using them with nohup, overriding them with a different PATH, and so on.

EXAMPLES

1. Make a version of cd that always prints out the new working direc? tory exactly once:

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

2. Start off a ``secure shell script'' in which the script avoids be? ing spoofed by its parent:

```
IFS='
'
```

```
    #   The preceding value should be <space><tab><newline>.
    #   Set IFS to its default value.
    \unalias -a
    #   Unset all possible aliases.
    #   Note that unalias is escaped to prevent an alias
    #   being used for unalias.
    unset -f command
    #   Ensure command is not a user function.
    PATH="$(command -p getconf PATH):$PATH"
    #   Put on a reliable PATH prefix.
    #   ...
```

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 as? sumes that implementation extensions may be present that would al? low user functions to exist when it is invoked; this capability is not specified by this volume of POSIX.1?2017, but it is not prohib? ited as an extension. For example, the ENV variable precedes the invocation of the script with a user start-up script. Such a script could define functions to spoof the application.

RATIONALE

Since command is a regular built-in utility it is always found prior to the PATH search.

There is nothing in the description of command that implies the command line is parsed any differently from that of 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.

The command utility is somewhat similar to the Eighth Edition shell 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 is most likely to be provided as a regular built-in. It is not listed as a special built-in 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 Section 2.14, Special Built-In Utilities) 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 non-interactive 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 non-intuitive.

The -p option is present because it is useful to be able to ensure a safe path search that finds all the standard utilities. This search might not be identical to the one that occurs through one of the exec functions (as defined in the System Interfaces volume of POSIX.1?2017) when PATH is unset. 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.

The command -v and -V options were added to satisfy requirements from users that are currently accomplished by three different historical utilities: type in the System V shell, whence in the KornShell, and which in the C shell. Since there is no historical agreement on how and what to accomplish here, the POSIX command utility was enhanced and the historical utilities were left unmodified. The C shell which merely conducts a path search. The KornShell whence is more elaborate?in addition to the categories required by POSIX, it also reports on tracked aliases, exported aliases, and undefined functions.

The output format of -V was left mostly unspecified because human users are its only audience. Applications should not be written to care about this information; they can use the output of -v to differentiate between various types of commands, but the additional information that

may be emitted by the more verbose -V is not needed and should not be arbitrarily constrained in its verbosity or localization for applica? tion parsing reasons.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.9.1.1, Command Search and Execution, Section 2.12, Shell Exe? cution Environment, Section 2.14, Special Built-In Utilities, sh, type

The Base Definitions volume of POSIX.1?2017, Chapter 8, Environment Variables, Section 12.2, Utility Syntax Guidelines

The System Interfaces volume of POSIX.1?2017, exec

COPYRIGHT