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 'set.1p' command

$ man set.1p

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

set ? set or unset options and positional parameters

SYNOPSIS

set [-abCefhmnuvx] [-o option] [argument...]

set [+abCefhmnuvx] [+o option] [argument...]

set -- [argument...]

set -o

set +o

DESCRIPTION

If no options or arguments are specified, set shall write the names and values of all shell variables in the collation sequence of the current locale. Each name shall start on a separate line, using the format:

"%s=%s\n", <name>, <value>

The value string shall be written with appropriate quoting; see the de? scription of shell quoting in Section 2.2, Quoting. The output shall be suitable for reinput to the shell, setting or resetting, as far as possible, the variables that are currently set; read-only variables

cannot be reset.

When options are specified, they shall set or unset attributes of the shell, as described below. When arguments are specified, they cause po? sitional parameters to be set or unset, as described below. Setting or unsetting attributes and positional parameters are not necessarily re? lated actions, but they can be combined in a single invocation of set.

The set special built-in shall support the Base Definitions volume of POSIX.1?2017, Section 12.2, Utility Syntax Guidelines except that op? tions can be specified with either a leading <hyphen-minus> (meaning enable the option) or <plus-sign> (meaning disable it) unless otherwise specified.

Implementations shall support the options in the following list in both their <hyphen-minus> and <plus-sign> forms. These options can also be specified as options to sh.

-a When this option is on, the export attribute shall be set for each variable to which an assignment is performed; see the Base Definitions volume of POSIX.1?2017, Section 4.23, Variable As? signment. If the assignment precedes a utility name in a com? mand, the export attribute shall not persist in the current exe? cution environment after the utility completes, with the excep? tion that preceding one of the special built-in utilities causes the export attribute to persist after the built-in has completed. If the assignment does not precede a utility name in the command, or if the assignment is a result of the operation of the getopts or read utilities, the export attribute shall persist until the variable is unset.

-b This option shall be supported if the implementation supports the User Portability Utilities option. It shall cause the shell to notify the user asynchronously of background job completions. The following message is written to standard error:

"[%d]%c %s%s\n", <job-number>, <current>, <status>, <job-name>

where the fields shall be as follows:

<current> The character '+' identifies the job that would be

used as a default for the fg or bg utilities; this job can also be specified using the job_id "%+" or "%%". The character '-' identifies the job that would become the default if the current default job were to exit; this job can also be specified using the job_id "%-". For other jobs, this field is a <space>. At most one job can be identified with '+' and at most one job can be identified with '-'. If there is any suspended job, then the current job shall be a suspended job. If there are at least two suspended jobs, then the previous job also shall be a suspended job.

<job-number>

A number that can be used to identify the process group to the wait, fg, bg, and kill utilities. Using these utilities, the job can be identified by prefix?ing the job number with '%'.

<status> Unspecified.

<job-name> Unspecified.

When the shell notifies the user a job has been completed, it may remove the job's process ID from the list of those known in the current shell execution environment; see Section 2.9.3.1, Exam?ples. Asynchronous notification shall not be enabled by default.

-C (Uppercase C.) Prevent existing files from being overwritten by the shell's '>' redirection operator (see Section 2.7.2, Redi?recting Output); the ">|" redirection operator shall override this noclobber option for an individual file.

-e When this option is on, when any command fails (for any of the reasons listed in Section 2.8.1, Consequences of Shell Errors or by returning an exit status greater than zero), the shell immedi?ately shall exit, as if by executing the exit special built-in utility with no arguments, with the following exceptions:

1. The failure of any individual command in a multi-command

pipeline shall not cause the shell to exit. Only the failure of the pipeline itself shall be considered.

2. The -e setting shall be ignored when executing the compound list following the while, until, if, or elif reserved word, a pipeline beginning with the ! reserved word, or any command of an AND-OR list other than the last.

3. If the exit status of a compound command other than a sub? shell command was the result of a failure while -e was being ignored, then -e shall not apply to this command.

This requirement applies to the shell environment and each sub? shell environment separately. For example, in:

set -e; (false; echo one) | cat; echo two

the false command causes the subshell to exit without executing echo one; however, echo two is executed because the exit status of the pipeline (false; echo one) | cat is zero.

-f The shell shall disable pathname expansion.

-h Locate and remember utilities invoked by functions as those func? tions are defined (the utilities are normally located when the function is executed).

-m This option shall be supported if the implementation supports the User Portability Utilities option. All jobs shall be run in their own process groups. Immediately before the shell issues a prompt after completion of the background job, a message reporting the exit status of the background job shall be written to standard error. If a foreground job stops, the shell shall write a message to standard error to that effect, formatted as described by the jobs utility. In addition, if a job changes status other than ex? iting (for example, if it stops for input or output or is stopped by a SIGSTOP signal), the shell shall write a similar message im? mediately prior to writing the next prompt. This option is en? abled by default for interactive shells.

-n The shell shall read commands but does not execute them; this can be used to check for shell script syntax errors. An interactive

shell may ignore this option.

-o Write the current settings of the options to standard output in an unspecified format.

+o Write the current option settings to standard output in a format that is suitable for reinput to the shell as commands that achieve the same options settings.

-o option

This option is supported if the system supports the User Porta? bility Utilities option. It shall set various options, many of which shall be equivalent to the single option letters. The fol? lowing values of option shall be supported:

allexport Equivalent to -a.

errexit Equivalent to -e.

ignoreeof Prevent an interactive shell from exiting on end-of-file. This setting prevents accidental logouts when <control>?D is entered. A user shall explicitly exit to leave the interactive shell.

monitor Equivalent to -m. This option is supported if the sys? tem supports the User Portability Utilities option.

noclobber Equivalent to -C (uppercase C).

noglob Equivalent to -f.

noexec Equivalent to -n.

nolog Prevent the entry of function definitions into the com? mand history; see Command History List.

notify Equivalent to -b.

nounset Equivalent to -u.

verbose Equivalent to -v.

vi Allow shell command line editing using the built-in vi editor. Enabling vi mode shall disable any other com? mand line editing mode provided as an implementation extension.

It need not be possible to set vi mode on for certain block-mode terminals.

xtrace Equivalent to -x.

-u When the shell tries to expand an unset parameter other than the '@' and '*' special parameters, it shall write a message to stan? dard error and the expansion shall fail with the consequences specified in Section 2.8.1, Consequences of Shell Errors.

-v The shell shall write its input to standard error as it is read.

-x The shell shall write to standard error a trace for each command after it expands the command and before it executes it. It is un? specified whether the command that turns tracing off is traced.

The default for all these options shall be off (unset) unless stated otherwise in the description of the option or unless the shell was in? voked with them on; see sh.

The remaining arguments shall be assigned in order to the positional parameters. The special parameter '#' shall be set to reflect the num? ber of positional parameters. All positional parameters shall be unset before any new values are assigned.

If the first argument is '-', the results are unspecified.

The special argument "--" immediately following the set command name can be used to delimit the arguments if the first argument begins with '+' or '-', or to prevent inadvertent listing of all shell variables when there are no arguments. The command set -- without argument shall unset all positional parameters and set the special parameter '#' to zero.

OPTIONS

See the DESCRIPTION.

OPERANDS

See the DESCRIPTION.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

None.

ASYNCHRONOUS EVENTS

Default.

STDOUT

See the DESCRIPTION.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

0 Successful completion.

>0 An invalid option was specified, or an error occurred.

CONSEQUENCES OF ERRORS

Default.

The following sections are informative.

APPLICATION USAGE

Application writers should avoid relying on set -e within functions.

For example, in the following script:

```
set -e
start() {
    some_server
    echo some_server started successfully
}
start || echo >&2 some_server failed
```

the -e setting is ignored within the function body (because the func?

tion is a command in an AND-OR list other than the last). Therefore, if

some_server fails, the function carries on to echo "some_serverstarted?

successfully", and the exit status of the function is zero (which means

"some_serverfailed" is not output).

EXAMPLES

Write out all variables and their values:

```
set
```

Set $1, $2, and $3 and set "$#" to 3:

```
set c a b
```

Turn on the -x and -v options:

```
set -xv
```

Unset all positional parameters:

```
set --
```

Set $1 to the value of x, even if it begins with '-' or '+':

```
set -- "$x"
```

Set the positional parameters to the expansion of x, even if x expands with a leading '-' or '+':

```
set -- $x
```

RATIONALE

The set -- form is listed specifically in the SYNOPSIS even though this usage is implied by the Utility Syntax Guidelines. The explanation of this feature removes any ambiguity about whether the set -- form might be misinterpreted as being equivalent to set without any options or ar? guments. The functionality of this form has been adopted from the Korn? Shell. In System V, set -- only unsets parameters if there is at least one argument; the only way to unset all parameters is to use shift. Using the KornShell version should not affect System V scripts because there should be no reason to issue it without arguments deliberately; if it were issued as, for example:

```
set -- "$@"
```

and there were in fact no arguments resulting from "$@", unsetting the parameters would have no result.

The set + form in early proposals was omitted as being an unnecessary duplication of set alone and not widespread historical practice.

The noclobber option was changed to allow set -C as well as the set -o noclobber option. The single-letter version was added so that the his? torical "$-" paradigm would not be broken; see Section 2.5.2, Special Parameters.

The description of the -e option is intended to match the behavior of the 1988 version of the KornShell.

The -h flag is related to command name hashing. See hash.

The following set flags were omitted intentionally with the following rationale:

-k The -k flag was originally added by the author of the Bourne shell to make it easier for users of pre-release versions of the shell. In early versions of the Bourne shell the construct set name=value had to be used to assign values to shell variables. The problem with -k is that the behavior affects parsing, virtu? ally precluding writing any compilers. To explain the behavior of -k, it is necessary to describe the parsing algorithm, which is implementation-defined. For example:

```
set -k; echo name=value
```

and:

```
set -k
echo name=value
```

behave differently. The interaction with functions is even more complex. What is more, the -k flag is never needed, since the command line could have been reordered.

-t The -t flag is hard to specify and almost never used. The only known use could be done with here-documents. Moreover, the behav? ior with ksh and sh differs. The reference page says that it ex? its after reading and executing one command. What is one command? If the input is date;date, sh executes both date commands while ksh does only the first.

Consideration was given to rewriting set to simplify its confusing syn? tax. A specific suggestion was that the unset utility should be used to unset options instead of using the non-getopt()-able +option syntax. However, the conclusion was reached that the historical practice of us? ing +option was satisfactory and that there was no compelling reason to modify such widespread historical practice.

The -o option was adopted from the KornShell to address user needs. In addition to its generally friendly interface, -o is needed to provide the vi command line editing mode, for which historical practice yields

no single-letter option name. (Although it might have been possible to invent such a letter, it was recognized that other editing modes would be developed and -o provides ample name space for describing such ex? tensions.)

Historical implementations are inconsistent in the format used for -o option status reporting. The +o format without an option-argument was added to allow portable access to the options that can be saved and then later restored using, for instance, a dot script.

Historically, sh did trace the command set +x, but ksh did not.

The ignoreeof setting prevents accidental logouts when the end-of-file character (typically <control>?D) is entered. A user shall explicitly exit to leave the interactive shell.

The set -m option was added to apply only to the UPE because it applies primarily to interactive use, not shell script applications.

The ability to do asynchronous notification became available in the 1988 version of the KornShell. To have it occur, the user had to issue the command:

```
trap "jobs -n" CLD
```

The C shell provides two different levels of an asynchronous notifica? tion capability. The environment variable notify is analogous to what is done in set -b or set -o notify. When set, it notifies the user im? mediately of background job completions. When unset, this capability is turned off.

The other notification ability comes through the built-in utility no? tify. The syntax is:

```
notify [%job ... ]
```

By issuing notify with no operands, it causes the C shell to notify the user asynchronously when the state of the current job changes. If given operands, notify asynchronously informs the user of changes in the states of the specified jobs.

To add asynchronous notification to the POSIX shell, neither the Korn? Shell extensions to trap, nor the C shell notify environment variable seemed appropriate (notify is not a proper POSIX environment variable

name).

The set -b option was selected as a compromise.

The notify built-in was considered to have more functionality than was required for simple asynchronous notification.

Historically, some shells applied the -u option to all parameters in? cluding $@ and $*. The standard developers felt that this was a mis? feature since it is normal and common for $@ and $* to be used in shell scripts regardless of whether they were passed any arguments. Treating these uses as an error when no arguments are passed reduces the value of -u for its intended purpose of finding spelling mistakes in variable names and uses of unset positional parameters.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.14, Special Built-In Utilities, hash

The Base Definitions volume of POSIX.1?2017, Section 4.23, Variable As? signment, Section 12.2, Utility Syntax Guidelines

COPYRIGHT