**NAME**

getoptcvt – convert to getopts to parse command options

**SYNOPSIS**

**/usr/lib/getoptcvt** [ –**b** ] *filename*

**/usr/lib/getoptcvt**

**DESCRIPTION**

**/usr/lib/getoptcvt** reads the shell script in *filename*, converts it to use **getopts** instead of **getopt**, and writes the results on the standard output.

**getopts** is a built-in Bourne shell command used to parse positional parameters and to check for valid options. See **sh**(1). It supports all applicable rules of the command syntax standard (see Rules 3-10, **intro**(1)). It should be used in place of the **getopt** command. (See the **NOTES** section below.) The syntax for the shell's built-in **getopts** command is:

**getopts** *optstring name* [ *argument* . . . ]

*optstring* must contain the option letters the command using **getopts** will recognize; if a letter is followed by a colon, the option is expected to have an argument, or group of arguments, which must be separated from it by white space.

Each time it is invoked, **getopts** places the next option in the shell variable *name* and the index of the next argument to be processed in the shell variable **OPTIND**. Whenever the shell or a shell script is invoked, **OPTIND** is initialized to **1**.

When an option requires an option-argument, **getopts** places it in the shell variable **OPTARG**.

If an illegal option is encountered, **?** will be placed in *name* .

When the end of options is encountered, **getopts** exits with a non-zero exit status. The special option —— may be used to delimit the end of the options.

By default, **getopts** parses the positional parameters. If extra arguments (*argument* . . .) are given on the **getopts** command line, **getopts** parses them instead.

So that all new commands will adhere to the command syntax standard described in **intro**(1), they should use **getopts** or **getopt** to parse positional parameters and check for options that are valid for that command (see the **NOTES** section below).

**OPTIONS**

–**b** Make the converted script portable to earlier releases of the UNIX system. **/usr/lib/getoptcvt** modifies the shell script in *filename* so that when the resulting shell script is executed, it determines at run time whether to invoke **getopts** or **getopt**.

**EXAMPLES**

The following fragment of a shell program shows how one might process the arguments for a command that can take the options **a** or **b**, as well as the option **o**, which requires an option-argument:

```
while getopts abo: c
do
        case $c in
        a | b)          FLAG=$c;;
        o)              OARG=$OPTARG;;
        \?)             echo $USAGE
                        exit 2;;
        esac
done
shift `expr $OPTIND − 1`
```

This code accepts any of the following as equivalent:

```
cmd −a −b −o "xxx z yy" filename
cmd −a −b −o "xxx z yy" −− filename
cmd −ab −o xxx,z,yy filename
cmd −ab −o "xxx z yy" filename
cmd −o xxx,z,yy −b −a filename
```

## SEE ALSO

**intro**(1), **sh**(1), **shell_builtins**(1), **getopt**(3C)

## DIAGNOSTICS

**getopts** prints an error message on the standard error when it encounters an option letter not included in *optstring*.

## NOTES

Although the following command syntax rule (see **intro**(1)) relaxations are permitted under the current implementation, they should not be used because they may not be supported in future releases of the system. As in the **EXAMPLES** section above, **a** and **b** are options, and the option **o** requires an option-argument. The following example violates Rule 5: options with option-arguments must not be grouped with other options:

**example% cmd −aboxxx filename**

The following example violates Rule 6: there must be white space after an option that takes an option-argument:

**example% cmd −ab −oxxx filename**

Changing the value of the shell variable **OPTIND** or parsing different sets of arguments may lead to unexpected results.