Korn Shell DEBUG Trap

Finnbarr P. Murphy

(fpm@fpmurphy.com)

The *trap* shell builtin is used to change the way signals are handled by the Korn Shell 93 (*ksh93*) shell. In addition, a *trap* may be set for three *ksh93* pseudo-signals: *EXIT, ERR,* and *DEBUG*. In this post we demonstrate how to use the *DEBUG* pseudo-signal to trap changes in the value of a variable for debugging purposes.

```
trap [  -p  ] [  action   ] [  sig   ] . . .

    The -p option causes the trap action associated with each trap as specified by the
    arguments to be printed with appropriate quoting. Otherwise, action  will be processed
    as if it were an argument to eval when the shell receives signal(s) sig. Each sig can
    be given as a number or as the name of the signal. Trap commands are executed in order
    of signal number. Any attempt to set a trap on a signal that was ignored on entry to
    the current shell is ineffective. If action  is omitted and the first sig  is a number,
    or if action  is -, then the trap(s) for each sig  are reset to their original values.
    If action  is the null string then this signal is ignored by the shell and by the
    commands it invokes. If sig  is ERR then action  will be executed whenever a command
    has a non-zero exit status. If sig  is DEBUG then action  will be executed before each
    command. The variable .sh.command will contain the contents of the current command lin
e
    when action  is running. If the exit status of the trap is 2 the command will not be
    executed. If the exit status of the trap is 255 and inside a function or a dot script,
    the function or dot script will return. If sig  is 0 or EXIT and the trap statement is
    executed inside the body of a function defined with the function name  syntax, then th
e
    command action is executed after the function completes. If sig  is 0 or EXIT for a tr
ap
    set outside any function then the command action  is executed on exit from the shell.
If
    sig is KEYBD, then action  will be executed whenever a key is read while in emacs,
    gmacs, or vi  mode. The trap command with no arguments prints a list of commands
    associated with each signal number.

    An exit or return without an argument in a trap action will preserve the exit status
    of the command that invoked the trap.
```

Consider the following simple shell script (*demo.ksh*) and its output when invoked:

```
#!/bin/ksh

let LIMIT=50

function debugprint {
    (( LIMIT > 50 )) &&  {
        printf "   >>> Line No: %dn" ${.sh.lineno}
        printf "   >>>   Level: %dn" "${.sh.level}"
        printf "   >>> Command: %sn" "${.sh.command}"
        printf "   >>>  Notice: LIMIT > 50n"
    }
}

trap debugprint DEBUG

printf "Value of LIMIT variable is %dn"  $LIMIT
(( LIMIT += 30 ))
```



```
printf "Value of LIMIT variable is %dn"  $LIMIT
```

```
$ ./demo.ksh
Value of LIMIT variable is 50
   >>> Line No: 17
   >>>   Level: 1
   >>> Command: printf 'Value of LIMIT variable is %dn' 80
   >>>  Notice: LIMIT > 50
Value of LIMIT variable is 80
$
```

As you can seem the script outputs detailed debugging information whenever the value of *LIMIT* is greater than 50.

Here is an explanation of the special *ksh93* shell variables used in the above script.

```
.sh.command

    When processing a DEBUG trap, this variable contains the current command line that
    is about to run.

sh.lineno

    Set during a DEBUG trap to the line number for the caller of each function.

.sh.level

    Set to the current function depth. This can be changed inside a DEBUG trap and will
    set the context to the specified level.
```

As you can see using the *TRAP* pseudo-signal is a much more targeted and detailed method of debugging a Korn Shell script than if you use the standard *set -x* shell debugging option.

