

Re: difference between the -o and || conditionals.

Source: <http://unix.derkeiler.com/Newsgroups/comp.unix.shell/2004-12/0047.html>

From: Stephane CHAZELAS (*this.address_at_is.invalid*)

Date: 12/01/04

Date: Wed, 1 Dec 2004 17:29:46 +0000

2004-12-1, 08:42(-08), Dexter:

[...]

> *Can someone please tell me the difference between the following constructs.*

>

> *if [\$? -ne 0] -o ["`cat \$tmp_file | grep -i error`"]; then*

> *print "error"*

> *fi*

> *==OR==*

>

> *if [\$? -ne 0] || ["`cat \$tmp_file | grep -i error`"]; then*

> *print "error"*

> *fi*

> *==OR==*

>

> *if [\$? -ne 0 -o "`cat \$tmp_file | grep -i error`"]; then*

> *print "error"*

> *fi*

[...]

None is correct.

If should be:

```
if [ "$?" -ne 0 ] || grep -li error < "$tmp_file" > /dev/null; then
```

"[" is an ordinary (even though generally built in the shell, but that makes no difference from a functional point of view) command.

In the first case, you are passing it those arguments:

0: "["

1 (or 0 or more): The expansion of \$? possibly splitted as you forgot to quote it and I don't know the current value of IFS.

2: "-ne"

3: "0"

4: "]"

5: "-o"

comp.unix.shell: Re: difference between the -o and || conditionals.

6: "["
7: the output of `cat $tmp_file | grep -i error` minus the trailing newline characters as a single argument
8: "]"

Given all those arguments, "[" will have a hard time guessing what kind of test you ask it to perform (and will probably fail as the above syntax is not correct).

In the second case, you are running a first "[" command with those arguments:

0: "["
1 (or 0 or more): The expansion of \$? possibly splitted as you forgot to quote it and I don't know the current value of IFS.
2: "-ne"
3: "0"
4: "]"

If \$? expands to a single argument, then "[" will be able to parse that command and will check if its first argument is the string representation of an integer number that is not equal to the number whose string representation is in its third argument. If true, it will return with a 0 exit status and with a non-0 exit status otherwise.

Then if (and only if) the exit status of that first command is non-zero, then the shell will run the second "[" command with those arguments:

0: "["
1: the output of `cat $tmp_file | grep -i error` minus the trailing newline characters as a single argument
2: "]"

Some "[" implementations when given only one argument beside the closing "]" will test whether that argument is empty or not (and report the corresponding exit status). Some other "[" implementations will first check if that argument looks like a test operator (such as "-f", "!"...), so it's wiser not to use that at all and use the more obvious:

```
[ -n "..." ]
```

In the third case, one "[" command is run with those arguments

0: "["
1 (or 0 or more): The expansion of \$? possibly splitted as you forgot to quote it and I don't know the current value of IFS.
2: "-ne"
3: "0"

Re: difference between the -o and || conditionals.

comp.unix.shell: Re: difference between the -o and || conditionals.

- 4: "-o"
- 5: the output of `cat $tmp_file | grep -i error` minus the trailing newline characters as a single argument
- 6: "]"

Generally, when given more than 3 arguments (beside the "]"), "[" implementations get very confused about what you are trying to test, especially if operands look like test operators. That's why the second case is preferred. In that particular case, moreover, the `cat` and `grep` commands are run and its (possibly huge) output is stored in memory and passed to the "[" command even if `$? -ne 0`

Now, let's compare with my suggested solution.

```
if [ "$?" -ne 0 ] || grep -li error < "$tmp_file" > /dev/null; then
```

The first "[" tests whether the expansion of `$?` is a the string representation of a number that is not-equal to 0.

Then, if that command returns false (a non-zero exit status), then the shell runs the `grep` command, with those arguments:

- 0: "grep"
 - 1: "-li"
 - 2: "error"
- (and that's all)

with its stdin connected to the file whose path is stored in `$tmp_file` and its stdout connected to `/dev/null`.

`grep` will then read one line at a time from its stdin, and as soon as it finds a line that contains "error" (regardless of case), it will output something like "(stdin)" to its stdout and exit immediately with a 0 (true) exit status. On the other hand, if it reaches the end of the file without finding "error", then it will return with a non-zero exit status.

Compare that with

```
[ "`cat $tmp_file | grep -i error`" ]
```

which creates two pipes, forks at least two processes, runs two commands, `cat` reading the whole file, then writing it without even doing any modification on it, `grep` reading from that pipe, outputting every line (til the end) containing "error", the shell storing all that output in memory, and then, when `grep` is finished, calls "[" with that string stored in memory as argument and then the "[" command in turns tests whether its argument is empty or not and return the corresponding exit status.

--

Re: difference between the -o and || conditionals.

comp.unix.shell: Re: difference between the -o and || conditionals.

Stephane