Confusing use of && and || operators

I was skimming through an `/etc/rc.d/init.d/sendmail` file (I know this is hardly ever used, but I'm studying for an exam), and I've become a bit confused about the `&&` and the `||` operators. I've read where they can be used in statements such as:

```
if [ test1 ] && [ test2 ]; then
     echo "both tests are true"
elif [ test1 ] || [ test2 ]; then
     echo "one test is true"
fi
```

However, this script shows single line statements such as:

```
[ -z "$SMQUEUE" ] && SMQUEUE="QUEUE"
[ -f /usr/sbin/sendmail ] || exit 0
```

These seem to be using the `&&` and `||` operators to elicit responses based on tests, but I haven't been able to dig up documenation regarding this particular use of these operators. Can anyone explain what these do in this particular context?

6 Answers

The right side of `&&` will only be evaluated if the exit status of the left side is zero. `||` is the opposite: it will evaluate the right side only if the left side exit status is nonzero. You can consider `[ ... ]` to be a program with a return value. If the test inside evaluates to true, it returns zero; it returns nonzero otherwise.

Examples:

```
$ false && echo howdy!

$ true && echo howdy!
howdy!
$ true || echo howdy!

$ false || echo howdy!
howdy!
```

Extra notes:

If you do `which [`, you might see that `[` actually does point to a program! It's usually not actually the one that runs in scripts, though; run `type [` to see what actually gets run. If you wan to try using the program, just give the full path like so: `/bin/[ 1 = 1`.

edited Nov 16 '11 at 21:10 answered Nov 16 '11 at 2:36
Shawn J. Goff
26.3k ● 16 ● 100 ● 122

4 When you see "X or Y", you test X. If it's true, you already know the answer to "X or Y" (it's true), so no need to test Y. If it's true, you don't know the answer to "X or Y", so you do need to test Y. When you see "X and Y", you test X. If it's false, you already know the answer to "X and Y" (it's false), so no need to test Y. If X is true, then you do need to test Y to see if "X and Y" is true. – David Schwartz Nov 16 '11 at 3:41

2 Instead of "if the left side returns zero", I would write "if the left side command's exit status is zero". I find "return" a bit ambiguous as the output and the exit status can both be considered as "return values" – glenn jackman Nov 16 '11 at 14:36

Not only can you "*consider* `[ ... ]` to be a program", in many cases it ***is***. It's commonly in the file `/bin/[` or `/usr/bin/[`. – L S Oct 28 '16 at 16:39

3 C programmers will find this super confusing. There 0 means false... While in shellscripting 0 means true... Mind blown. – Calmarius Aug 15 '17 at 9:37

Another one that you could mention is `test` instead of `if` or `[`. And `if [` and `if test` seem to be interspersed with `[` and `test` with no apparent rhyme or reason. `test` shows up on occasion, like at config.site for vendor libs on Fedora x86_64. – jww Oct 21 '17 at 4:45

Here's my cheat sheet:

- "A ; B" Run A and then B, regardless of success of A
- "A && B" Run B if A succeeded
- "A || B" Run B if A failed
- "A &" Run A in background.

I like the cheat answers like yours. Thanks! – Tarik May 14 '17 at 2:34

There's some interesting lambda calculus parallels going on here demonstrated in working JavaScript(define the variables in order); "left(aka true)": `(a => b => a)`. "right(aka false)": `(a => b => b)`. "A; B" "then" `(a => b => (() => b())(a()))`. "A && B" "if without else(when)" `(a => b => a()(() => right)(b))`. "A || B" "else without if(unless)" `(a => b => a()(b)(() => right))`. "a && b || c" "ifThenElse" `(a => b => c => (unless(when(a)(b))(c))())`. – Dmitry Jul 15 '17 at 4:53

note that in bash, left is analogous 0/empty string, right analogous is everything else. – Dmitry Jul 15 '17 at 5:00

---

to expand on @Shawn-j-Goff's answer from above, `&&` is a logical AND, and `||` is a logical OR.

See this part of the Advanced Bash Scripting Guide. Some of the contents from the link for user reference as below.

&& AND

```
if [ $condition1 ] && [ $condition2 ]
#  Same as:  if [ $condition1 -a $condition2 ]
#  Returns true if both condition1 and condition2 hold true...

if [[ $condition1 && $condition2 ]]    # Also works.
#  Note that && operator not permitted inside brackets
#+ of [ ... ] construct.
```

|| OR

```
if [ $condition1 ] || [ $condition2 ]
# Same as:  if [ $condition1 -o $condition2 ]
# Returns true if either condition1 or condition2 holds true...

if [[ $condition1 || $condition2 ]]    # Also works.
#  Note that || operator not permitted inside brackets
#+ of a [ ... ] construct.
```

edited Aug 29 '17 at 5:27
GeneCode
103 ● 4

answered Nov 16 '11 at 6:08
Tim Kennedy
12k ● 2 ● 25 ● 45

From my experience I use the && and || to reduce an if statement to a single line.

Say we are looking for a file called /root/Sample.txt then the traditional iteration would be as follows in shell:

```
if [ -f /root/Sample.txt ]
then
    echo "file found"
else
    echo "file not found"
fi
```

These 6 lines can be reduced to a single line:

```
[[ -f /root/Sample.txt ]] && echo "file found" || echo "file not found"
```

When running a few iterations to set variables or to create files etc., life is easier and the script looks slicker using the single line if function, it's only drawback is that it becomes a bit more difficult to implement multiple commands from a single iteration however you can make use of functions.

edited Nov 10 '17 at 8:48
Community ♦
1

answered Jun 21 '16 at 0:15
syme89
61 ● 1 ● 1

This is cool. Reminds me of objc code (a==b)?val=1:val=2; – GeneCode Aug 29 '17 at 4:13

There is a notion of "short cutting".

When `(expr1 && expr2)` is evaluated - `expr2` is only evaluated if `exp1` evaluates to "true". This is because both `expr1` AND `expr2` have to be true for `(expr1 && expr2)` to be true. If `expr1` evaluates to "false" `expr2` is NOT evalued (short cut) because `(expr1 && expr2)` is already "flase".

Try the following - assume file `F1` exists & file `F2` does not exist:

```
( [ -s F1 ] && echo "File Exists" )  # will print "File Exists" - no short cut
( [ -s F2 ] && echo "File Exists" )  # will NOT print "File Exists" - short cut
```

Similarly for `||` (or) - but short cutting is reversed.

edited Jan 8 '15 at 19:47
jasonwryan
42.7k ● 14 ● 110 ● 160

answered Jan 8 '15 at 18:47
Larry
21 ● 1

4 The logic is usually referred to as "short circuiting". I've never seen the phrase "short cutting" used. I mention this to help with finding references. – L S Oct 28 '16 at 16:42