Tip 35: Use the shell to test for files

Ben Klemens

10 December 2011

level: you want some automation out of your shell
purpose: look before you leap

Last time (Entry #084), I discussed how the shell can be used as a Turing-complete programming language (and advised you to not use it as such). For example, you could automate the sort of thing you'd type at a command line, like setting up a sequence of runs of a program.

Now let's say that your program relies on a data set that has to be read in from a text file to a database. You only want to do the read-in once; in pseudocode: if database exists do nothing, else generate database from text.

On the command line, you would use `test`, a versatile command typically built into the shell. Run a quick `ls`, get a file name you know is there, and use `test` like this:

```
test -e a_file_i_know
echo $?
```

By itself `test` outputs nothing, but since you're a C programmer, you know that every program has a `main` function that returns an integer, and we will use only that return value here. Custom is to read the return value as a problem number, so 0=no problem, and in this case 1=file does not exist. [Which is why, as per Tip #4 (Entry #053), the default is that `main` returns zero.] The shell doesn't print the return value to the screen, but stores it in a variable, `$?`, which you can print via `echo`.

OK, now let us use it in an `if` statement to act only if a file does not exist. As in C, `!` means *not*.

```
if test ! -e a_test_file; then
    echo test file had not existed
    touch a_test_file
else
    echo test file existed
    rm a_test_file
fi
```

Notice that, as with the `for` loops from last time, the semicolon is in what I consider an awkward position, and we have the super-cute rule that we end `if` blocks with fi. To make it easier for you to run this repeatedly using !!, let's cram it onto one margin-busting line. The keywords [and] are equivalent to test, so when you see this form in other people's scripts and want to know what's going on, the answer is in man test.

```
if [ ! -e a_test_file ]; then echo test file had not existed; touch a_test_file;
```

The multi-line version would make a fine header for the script from last time: start with some if statements to check that everything is in place, then run a for loop to run your program a few thousand times.

The condition is considered to be true when the evaluated expression is zero (=no problem), and false when it is nonzero (=problem). So outside of the test command you can think of the typical if statement as *if the program ran OK, then...*, which makes it perfect for error checking:

```
#generate some test files
mkdir a_test_dir
echo testing ... testing > a_test_dir/tt

#Remove the test files iff they were archived right
if tar cz a_test_dir > archived.tgz; then
    echo Compression went OK. Removing directory.
    rm -r a_test_dir
else
    echo Compression failed. Doing nothing.
fi
```

[If you want to see this fail after running once, try `chmod 000 archived.tgz` to make the destination archive unwriteable, then re-run.]