Tip 9: Compile C programs via here script

Ben Klemens

19 October 2011

level: amused beginner
purpose: Compile via cut/paste

Today's tip compiles C code pasted onto the command line. If you are a kinetic learner who picked up scripting languages by cutting and pasting snippets of code into the interpreter, you'll be able to do the same with C and your shell.

Further, a shell is glue that does traffic control among the many programs that do the real work. This tip will let you put C into that flow, so C code can be sandwiched into a single file that also has plain shell instructions or steps in Perl or R.

We're not going to use the makefile, so we need a single compilation command. To make life less painful, let us alias it. Paste this onto your command line, or add it to your `.bashrc`, `.cshrc`, &c.:

```
go_libs="-lm"
go_flags="-g -Wall -include allheads.h --std=gnu99 -O3"
alias go_c="gcc -xc '-' $go_libs $go_flags"

#C shell users do this a little differently:
set go_libs="-lm"
set go_flags="-g -Wall -include allheads.h --std=gnu99 -O3"
alias go_c "gcc -xc '-' $go_libs $go_flags"
```

where `allheads.h` is the aggregate header you'd put together in tip #6. Using this `-include` switch means one less thing to think about when writing the C code, and I've found that bash's history gets wonky when there are `#`s in the C code. [The `zsh` is a big winner when it comes to interactively editing histories with long here scripts, and has no problem with the #s.]

On the gcc line, you'll recognize the `'-'` to mean that instead of reading from a named file, use stdin. The `-xc` flags this as C code, because *GCC* stands for GNU Compiler Collection, not GNU C Compiler, and with no input filename ending in `.c` to tip it off, we have to be clear that this is not Java, Fortran, Objective C, Ada, or C++.

Whatever you did to customize the `LIBS` and `CFLAGS` in your makefile, do here. For example, here are my versions of `go_libs` and `go_flags`, because I have pkg-config installed and use GLib and Apophenia (and by implication, the GSL and SQLite) for everything:

```
go_libs="`pkg-config --libs glib-2.0 apophenia`"
go_flags="-g -Wall -include apop.h --std=gnu99 -O3"
```

Now we can use a here document to paste short scripts onto the command line. Not only do you not need a makefile, you don't even need an input file or command interpreter.

To do:
After defining the right aliases for your setup (including a reference to your aggregate header), paste this onto your command line:

```
go_c <<"EOF"
int main(){
    long int testme = 2, ct =0;
    long int *primes   = NULL;
    while(1){
        int isprime = 1;
        for (long int i=0; isprime && i< sqrt(testme) && i<ct; i++)
            isprime = testme % primes[i];
        if (isprime){
            printf("%li  \r", testme); fflush(NULL);
            primes       = realloc(primes, sizeof(long int)*(ct+1));
            primes[ct++] = testme;
        }
        testme  ++;
    }
}
EOF

./a.out #This line will wait for you to hit <ctrl>-C
```

Now you have a program that generates prime numbers faster than you can read them [stop via <ctrl>-C]. You can change `\r` to `\t` or `\n` if you want to keep a record. Until I got bored and stopped, it was testing about 100,000 numbers/second on my netbook, but you can see how it does on your machine.

Managing lines of code on the command line is not fun, so don't expect this sort of thing to be your primary mode of working. But cutting and pasting code snippets onto the command line *is* fun, and being able to have a single step in C within a longer shell script is pretty fabulous.