Tip 83: Use m4 in the middle of your documents

Ben Klemens

16 March 2012

level: macro hacker
purpose: eliminate any and all repetition

The m4 macro language is mostly interesting because its macros are intended to be put anywhere in any text file. We'll bulletproof the example in a little bit, but let's say that we have the macro file:

```
m4_divert(-1)
m4_define(Emph, <em>$1</em>)
m4_divert(1)</body></html>
m4_divert(0)<html><head><meta charset="utf-8" /></head><body>
```

and the text file

```
Welcome to my Emph(lovely) web site.
```

then after you run `m4 macros.m4 text > out.html` you'd wind up with:

```
<html><head><meta charset="utf-8" /></head><body>
Welcome to my <em>lovely</em> web site.
</body></html>
```

What just happened:

- We just automated a whole lot of cruft at once. There is just no way to produce HTML, XML, or even certain blocks of LaTeXwithout help from something that automates redundancy, and m4 will do that for you.
- The `m4_define` is the simplest macro definition: you give the macro definer a name and an expansion, where the expansion can have the usual positional parameters like $1, $2, ... That's all it takes for m4 to know what to do when it gets into your text and sees `Emph(lovely)`.
- Diversions: `m4_divert(-1)` sends output to `/dev/null`. So after that line m4 is reading in macro definitions but isn't writing anything. `m4_divert(1)` stores output into buffer 1. `m4_divert(0)` writes to standard out, which should be the normal course of affairs. At the end of the text, the buffers get

written to output in order, which is how we got the end tags at the end of the file. Or, write a `footer.m4` to put on the command line after your main text. Use `m4_undivert(1)` to empty out buffer 1 sooner.

The expansions are aggressive: if your macro doesn't have parens after it, it'll still get expanded, so if you happen to have `Emph` in plain text, that'll get turned into HTML tags. If we're going to have m4 operate on an arbitrary text or code file, we need to make certain that it doesn't surprise us. E.g., use macro names that don't make sense as standalone strings. Notice also that we're using `m4 -P`, which puts that `m4_` tag at the head of every function name. Otherwise, if you use the word `divert` in your text, it gets eaten. You may also find stray line breaks due to expansions; use `m4_dnl` to prevent those (delete to new line). Here's an m4 file with some further protections and tricks built in:

```
m4_divert(-1)
m4_changequote(‘’,‘’) # m4 eats all quote-endquote markers, so make sure
                      # they will never appear in your text by using odd ones.
                      # Notice how these aren’t the plain <> signs;
                      # vim users, try :help digraph.
                      # I also wrote a vim macro to write ( and ) for me.
                      # To avoid sad surprises, wrap all all macro inputs in t

m4_changecom(m4 comment:) #Octothorpes appear in plain text.

#A macro to define new macros.
m4_define(newXML, m4_define($1, <$2>$1</$2>)m4_dnl)

newXML(Emph,em)
newXML(Pp,p)

m4_divert(1)</body></html>
m4_divert(0)<html><head><meta charset="utf-8" /></head></body>

# Let’s throw some sample uses here, so we can test the
# m4 file with itself. When we’re happy, move the m4 divert(0)
# line to the end so these get sent to /dev/null.
Pp(Dear reader,)

Pp(HTML was Emph(originally) designed to be handwritten,
but now generating well-formed documents is just a pain.)
```

The language does a few more tricks: optional arguments, if/thens, loops, but if you keep it simple, you can fix a lot of annoyances with just a few lines of macro definitions.