

# The ABCs of Letters

by Jeffrey Copeland and Haemer

**F**or the last two years, we've focused on helping you solve programming problems. We thought we'd try shifting gears and help you solve problems that arise while running your business. If that's confusing, an example of what we mean will probably help.

Part of working is writing letters. It would be more convenient if all our mail were electronic, but it's not. Everywhere we've worked, even working for ourselves, we've ended up automating our letter-writing. In this column, we'll walk you through the process, both to offer some simple tricks that you may not already be using, and to use letter-generation as a model of how to automate a variety of other day-to-day tasks.

## Using a Letter Template

In the beginning, we handcraft all our documents. After a while, though, we figure out what looks nice and create a template for our letters, with the right headers and footers, a



paragraph style that we like, dates in the right places and so on. Once we've done that, generating a new letter starts with copying the template to an appropriately named file in the directory where we keep our letters. (We store all of ours in a single spot, so they're easy to find if we need to go back and look at them later.) Here's how:

```
#!/bin/sh
PATH=/bin:/usr/bin
```

*Jeffrey Copeland (copeland@alumni.caltech.edu) is a member of the technical staff at QMS's languages group, in Boulder, CO. His recent adventures include internationalizing a large sales and manufacturing system and providing software services to the administrators of the 1993 and 1994 Hugo awards. His research interests include internationalization, typesetting, cats and children.*

*Jeffrey S. Haemer (jsh@canary.com) is an independent consultant based in Boulder, CO. He works, writes and speaks on the interrelated topics of open systems, standards, software portability and porting, and internationalization. Dr. Haemer has been a featured speaker at Usenix, UniForum and Expo Kuwait.*

```

TEMPLATE=$HOME/roff/templates/letter.mm
LETTER=$HOME/letters/$1
mkdir -p $HOME/letters 2> /dev/null

USAGE="usage: `basename $0` filename"
abort() {
    echo $* 1>&2
    exit 1
}
test $# = 1 || abort $USAGE

case $1 in
    *.mm) ;;
    *) LETTER=${LETTER}.mm ;;
esac
test -f $LETTER && abort "Letter $LETTER
already exists."

cp $TEMPLATE $LETTER
${EDITOR:=vi} $LETTER

cmp $LETTER $TEMPLATE && rm -f $LETTER

```

This shell script isn't a DOS-like "batch file," but a real and moderately sturdy program. Let's go through it a step at a time:

1. We begin with some security precautions. The first line ensures that our script is read by the correct interpreter. The second guarantees that all invoked commands are the system versions, not local hacks or Trojan horses.
2. We define symbolic constants for the locations of the letter repository and template, as an aid to portability. You may want yours in a different spot from ours.
3. We create the directory to contain our letters if it doesn't already exist. This is also a portability issue; we move our scripts around a fair amount, and it's annoying to have our scripts fail when we execute them on a new system and discover that we've forgotten to create the right directories. The `-p` flag will create any needed intermediate directories. In our case, this means `$HOME` and its parent directories, which are certain to exist, but the extra flag will help a lot when we change the value of `$LETTER` to point to something several levels deeper.
4. We have an error exit routine that encapsulates error handling, so we can get it right the first time and forget about it. Ours, `abort`, redirects error messages to standard error and exits with a status that says the script fails. We've seen more sophisticated routines, but ours is simple and works well enough for this script.
5. We do argument parsing. In a larger script with more arguments, we might use the POSIX.2 `getopts` call to do the parsing, but this one is small enough that we do the work by hand. Notice that we make `letter foo` and `letter foo.mm` refer to the same letter.

6. We invoke the editor. If `$EDITOR` is set in the environment, we use its value as our editor; otherwise we use the default editor standardized by POSIX.2, *vi*.

7. Sometimes we change our minds and decide not to write a letter after all. In practice, we've discovered that this happens often enough that unless we're careful, we're left with a lot of unbegun letters. Accordingly, at the end of our script we remove the copy of the template if we haven't done something with it.

## The Letter Macros

As you've probably guessed from our file-naming convention, we use `nr`'s `mm` macro package to write our letters. If you use a different macro package or a different formatter, like `TEX`, you'll want to change several details in your template and the shell script, but you'll still be able to use much of the code in this column. If you're using a WYSIWYG word processor, like `WordPerfect`, you'll need to modify this in more than the details, but the principles of how to use the UNIX shell and tools still apply.

The `mm` macro package has, in recent years, acquired a suite of letter macros. They are often good, but they aren't available on all the machines we generally use. For example, `groff`'s version of `mm` doesn't yet include letter macros, nor do most versions of `mm` available on Sun machines. As a result, we use our own add-on package of macros, which is compatible with `mm`'s letter macros. Here are some of the important macros:

**.LT** Letter type: your choice for placement of header and signature blocks, and paragraph style (e.g., `.LT SB` produces a "semi-blocked" style, with paragraphs indented five spaces, and the date, writer's address and signature blocks left-justified).

**.LO** Letter options: a variety of annotations such as "Subject: Aardvarks" (`.LO SJ Aardvarks`).

**.FC** Formal closing: the line before your name (e.g., "Sincerely yours").

**.SG** Signature: your name (e.g., "Gillian G. Haemer").

**.WA/.WE** Writer's address: It goes at the top of the first page (on the left for full-blocked letters, on the right for most others).

**.IA/.IE** Internal address delimiters: the recipient's address.

**.NS/.NE** Notation delimiters: all the little things that appear at the bottom of letters after the signature, like "Attn. J. J. Copeland" and "Copy to David Brin."

These macros provide enough for you to do basic letter formatting and let you use most of the other familiar `mm` macros, such as `.P` and `.LI`, in the body of your letter.

## The Template Itself

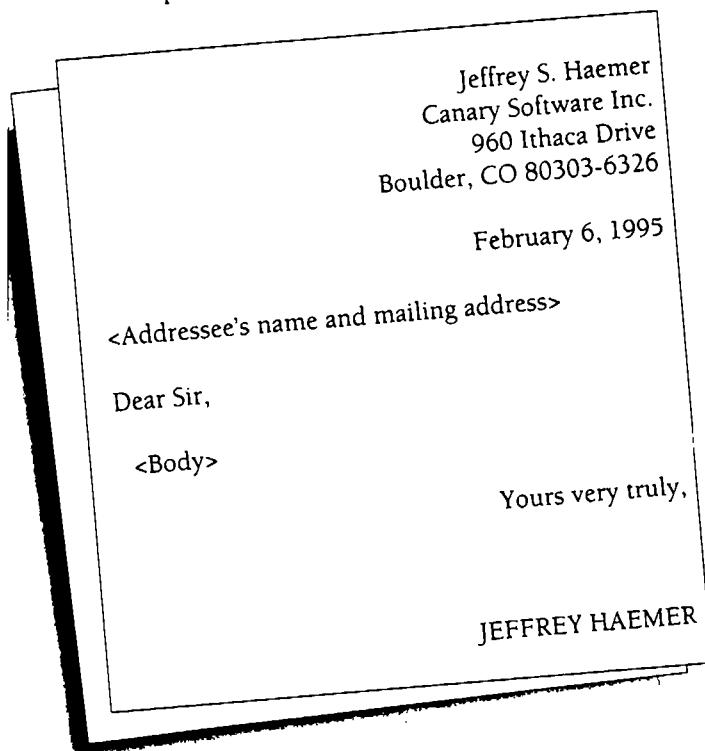
With that lead-in, here's our template:

```
.LT SB
```

## Work

```
.WA
Jeffrey S. Haemer
Canary Software, Inc.
960 Ithaca Drive
Boulder, CO 80303-6326
<Addressee's name and mailing address>
.IE
.LO SA "Dear Sir,"
.P
<Body>
.FC
.SG "J\s-2EFFREY\s0 S.\ H\s-2AEMER\s0"
```

The output of this is:



### Letterhead

This isn't a bad form letter, but it doesn't print well on our letterhead. The address in the upper-right-hand corner is superfluous, and it also overprints our letterhead. Moreover, we find we prefer a fully blocked form on letterhead. We can fix these by modifying the template:

```
.LT FB
\&
.SP
.IA
<Addressee's name and mailing address>
.IE
.LO SA "Dear Sir,"
.P
<Body>
```

```
.FC
.SG "J\s-2EFFREY\s0 S.\ H\s-2AEMER\s0"
How will our letter command know what to do? One possibility is to do more complex argument parsing. Alternatively, you could have a separate command, letterhead, that uses a different template. We combine these two approaches by modifying the original command. It begins like this:
```

```
#!/bin/sh
PATH=/bin:/usr/bin

DIR=$HOME/roff/templates
ARGV0=$(basename $0)
case $ARGV0 in
  letter) TEMPLATE=$DIR/letter.mm ;;
  letterhead) TEMPLATE=$DIR/letterhead.mm ;;
esac

# etc.
```

This lets us link the two commands: When invoked as `letter`, the script uses one template, when invoked as `letterhead`, it uses the other.

When you're developing software for your own use, you're your own maintenance organization. Whenever you can, you should centralize common code to ease your maintenance burden. If you have two commands that differ in only a few lines, it won't be long before you forget to propagate fixes and upgrades from one to the other. Here, there's only one difference: where they take their templates from. And because it's easy to imagine that we may eventually have several templates, each invoked by a different command, we have our script choose the template in a switch statement, instead of a simple "if-then-else."

### Where Are We?

Using the commonplace task of generating letters, we've introduced the topic of how to design shell scripts to automate simple office tasks, talked about robustness and easing maintenance and touched briefly on the `mm` letter macros. In the remaining columns in this series, we'll try to follow this model:

- Pick a common work-day problem
- Show how simple UNIX tools can help automate the job
- Try to introduce a tool or two that isn't widely known
- Use the design of the solution to highlight some general points about how to attack similar problems.

We like the idea of trying to solve problems that we know you're interested in. If you have particular things you'd like to see, contact us at the addresses given in this column. Next month, we'll continue talking about letters. We'll begin with how we do envelopes. ▲