# Diaries and To-Do Lists

by Jeffreys Copeland and Haemer

In this, the latest in our series on common office problems, we're going to begin discussing two interlinked dilemmas: what to do and when to do it, or calendars and to-do lists. As usual, we'll explore particular solutions–including programs–that we've used.

First, though, let's quickly revisit last month's discussion of geographic proximity. As usual, after we sent the column off to our editors, we discovered more resources for solving our problem.

Someone has given us a file from the Net, `GEONAME.TXT`, containing a list of locations in the United States with their latitude and longitude. This list provides both neighborhood and city names, so it has entries like:

```
Sugarloaf Boulder CO 400101N1052424W
```

Unfortunately, no one here can remember exactly where the list came from, and we haven't been able to find it using `archie` either, so we can't give you an FTP site. If you find it in your Net cruising, please let us know.

There are a variety of DOS databases on CD-ROM containing map data, available at commercial software retail stores. We have been using one in the QMS office that contains street-

*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.*

level data of the United States.

Part of Boulder's economy is tourism, and downtown Boulder has a pedestrian mall full of street performers. One of the performers has an act tailored to our problem. Once he has gathered a crowd of tourists with his warm-up of passable juggling and unicycle riding, he launches into his real act:

"Who here is from out of town?"
Dozens of hands go up.
"What's your zip code?"
"07601!"
"Hackensack, New Jersey, near Teaneck."
Once he warms up, he moves on to foreign postal codes.
"CR3-2DA!"
"Coulsdon, Surrey, England."
"I don't have a postal code!"
"You're from New Zealand."

Now if we get him a portable phone with an 800 number...(Thanks to Tom Mullin, our hang-gliding colleague in Boulder, for bringing the first two of these to our attention.)

## What Did I Do Last Week?

We've spent a good amount of time talking about storing and retrieving spatial data, from addresses to geographical locations. Let's spend a little time looking at temporal data, starting with diaries.

Diaries are surprisingly useful tools. We have used them for constructing the raw materials for status reports and invoices, for figuring out what weeks in 1992 we actually spent at home and just keeping notes when our children do amusing things.

Over the years, we have designed, built and used a variety of diary programs, varying from the simple and stripped-down, to the baroque, full of bells and whistles. Right now, we are in a "Cleanliness is next to Godliness" phase, so we will show you a very simple implementation that you can embellish to your heart's content. (We shall even, as is our wont, suggest a few elaborations.)

## Listing 1

```
#!/usr/bin/perl
#
# create weekly diary entries.
# diaries are kept in one file per week
# entries are time-stamped, and each day is marked;
# this relies on macros for week (.WK), day (.DY)
# and time (.TM)
# for formatting

$usage = "usage: $0 [logdir]";

$dateinfo = `date +"%W#%e %B %Y#%A %D#%H: %M"`;
($wknum, $week, $day, $time) = split(/#/, $dateinfo);

if (@ARGV == 1) {
        $LOGDIR = shift;
        die "$usage" unless $LOGDIR =~ /^[\/\w]/;
} elsif ($ENV{"LOGDIR"}) {
        $LOGDIR = $ENV{"LOGDIR"};
} else {
        $LOGDIR = $ENV{"HOME"} . "/Logs";
}
die "$usage" if @ARGV;

$LOGFILE = "$LOGDIR/$wknum";
if (-r $LOGFILE) {
        open (I, "$LOGFILE") || die "Can't read $LOGFILE: $!";
        $logp = 1;
}

$week = ".WK " . $week;
$day = ".DY " . $day;
$time = ".TM " . $time;

@logfile = <I>;
@x = grep(/^$day$/, @logfile);
close(I);

open (O, ">> $LOGFILE") || die "Can't append to $LOGFILE: $!";
select O;

print "$week\n" unless ($logp);

print "$day\n" unless @x;
print "$time\n";
close O;
system("vi -c $LOGFILE");
```

First, an overview. Our command, diary, will invoke an editor on the current diary file. Each file will span a week and will be named by the week number in the year. For example, the diary file for the fifth week of the year will be $LOGDIR/5. The program will also mark each entry with the date and time.

Listing 1 shows a short perl script that does just what we want.

(You may be wondering why we use languages like perl, awk and the shell so often in this column. Why not provide, say, full-blown C or C++ programs? Simple: We need short programs that will fit in a column. They may not be as fast as a compiled program would be, but we don't run them very often, and they are fast enough. If you need to speed them up, you can rewrite them in C or C++.)

Now we will walk through the code.

We begin by collecting the date and time information. We do this by calling the UNIX date command.

As we discussed in some detail in our earlier series on POSIX programming, the UNIX date command is built around a call to a POSIX.1 function, strftime(), that provides date information in a wide variety of formats. (Actually, the format strings for date and strftime() are more or less identical. Take a look at the man pages for each.)

Like printf(), strftime() permits arbitrary characters in the format strings. Here is an example:

```
$ date '+Today is %A.'
Today is Sunday.
```

This lets us get away with a single call to date. We just ask for all the information at once, separating the pieces by a special character, #, and then break out the individual pieces with perl's split() function, like this:

```
split(C/#/, $dateinfo)
```

(This is a good trick, and we will use it again for the to-do lists.)

Next, we figure out where we are going to put our diary. We make this flexible, in case we are keeping more than one diary, by letting the user specify a directory for the log files either on the command line or as an environment variable, $LOGDIR. If neither of these is given, we have a default, $HOME/Logs.

Using the week number we got from date, we try to read this week's diary file, $LOGDIR/$wknum. If we succeed, we look to see if we already have an entry for today.

## Laying Out Diary Files

To see how we do this, we will digress a moment to talk about the layout of our diary files. So far, we have been thinking about putting entries into a diary. Eventually, we need to do something with these entries, typically either print them or parse them. This requires formatting the entries. We could have code to do the formatting at the time we make the entries, but why not leave this job to a formatter, like troff or TeX? Here, we use troff, and begin all diary files with the macro .WK *this week*, all daily entries with .DY *today* and all individual entries with .TM *entry time*. By doing this, we can put off deciding how to format diary pages until we are ready to print them but still have tags that let us parse the files unambiguously.

For example, we can search to see if we already have an entry for today by looking for a line consisting of the string .DY *today*. Figure 1 is an example diary file.

Because many folks treat troff macros as black boxes, here is a simple working version of the macros:

```
.de WK
.SK
.S +4 +4
```

**Figure 1**

```
.WK 4 September 1995
.DY Monday 09/04/95
.TM 22:17
Spent Labor Day with kids; Jeff and his kids joined us.
.DY Tuesday 09/05/95
.TM 17:15
Finally began work on adaptive compression.
Began writing text of RS article.
.TM 22:32
Got TIFF file encoding to work on machine at home!!!
.DY Wednesday 09/06/95
.TM 17:22
Interrupted from adaptive compression by bugs from support group.
More work on RS text.
.DY Thursday 09/07/95
.TM 10:32
Phone call from Rob in Chicago: need to add list of features to new
release for his customer. Raise this issue at next project review.
(Salesmen! Why do I feel like I'm living in a Dilbert comic?)
.TM 17:33
Completed adaptive compression? Need exhaustive testing.
.TM 22:17
Successfully sent fax from DOS fax board to Jeff's fax machine!.
```

```
.B
.ce 2
Diary
.br
Week of \$1 \$2 \$3
.SP 3
.R
.S -2 -2
..
.de DY
.in 0
.SP
.I
\\$1 (\\$2)
.R
.SP
..
.de TM
.in 0
\(bu \\$1
.in +5
.br
..
```

After all this, we reopen the file for writing, append the week title (if we don't have it), today's date (if we don't have it) and the current time, marked with appropriate macros. Finally, we invoke an editor to let us put in whatever we want as our diary entry.

We have taken advantage here of `vi`'s `-c` flag, which lets us give an instruction to `vi` at start-up time. (Old-timers will recognize this as the POSIX.2 replacement for `vi`'s old + flag.) We usually want to append diary entries to the end of the file. Invoking the editor as `vi -c $ $LOGFILE` starts us up with our cursor at the bottom of the file, ready to go.

Our `diary` command should really be more flexible about what editor it invokes because some users prefer not to edit with `vi`. We leave how to do that as an exercise to the reader.

Notice also that if this week's diary file doesn't exist yet, we title it with today's date, which is not necessarily the first of the week. How would you fix that? Is it worth it?

## What to Do Next

OK, that records what we did. How about what we are going to do? Our next goal is to handle upcoming tasks.

First, a design sketch. As with `diary`, we use a directory, `$CALDIR`, to contain our lists of reminders. Into that directory we put several kinds of files.

• A list of tasks for individual dates, like this:

```
09/07/95
    November RS column due
    Take cat to vet

09/21/95
    Gillian's birthday

10/06/95
    run off to join the circus

12/25/95
    half day off
```

**Our *diary* command should really be more flexible about what editor it invokes because some users prefer not to edit with *vi*.**

• A list of recurring, daily tasks–things we need to do every day, but that it helps to have a checklist of:

```
process email

process phone mail

process U.S. mail
```

• Recurring weekly tasks:

```
Monday
    status report

Wednesday
    ex-Interactive MTS lunch

Friday
    time sheet
```

• Once-a-month tasks:

```
01
    send out monthly billing
15
    deposit checks
```

```
25
    send out billing reminders
```

This design is borrowed from the hierarchical organization of cron files in Berkeley UNIX, and it accurately reflects the way we think about our own to-do lists: one-shot events overlaid on lists of tasks that recur at daily, weekly and monthly intervals. We need to add syntax to say "end-of-month" in the monthly file because different months end on different days. But we have very few things we have to do every other month, every other day or once a fortnight.

**This design is borrowed from the hierarchical organization of *cron* files in Berkeley UNIX, and it accurately reflects the way we think about our own to-do lists.**

## Problems We Have Left

But what shall we do with all these different sorts of events? If you have been reading our columns for a while, you will have already guessed the answer. After we collect all of the events for the upcoming day, we shall make them into a diary entry. There are still problems we have not addressed in our design. Let's list a few:

• Besides scheduled events (`dentist's appt., 11/22,12:30 p.m.`), we need a way to handle what our old time-management teacher called a "grass-catcher list" (`purge paperback books in den`).

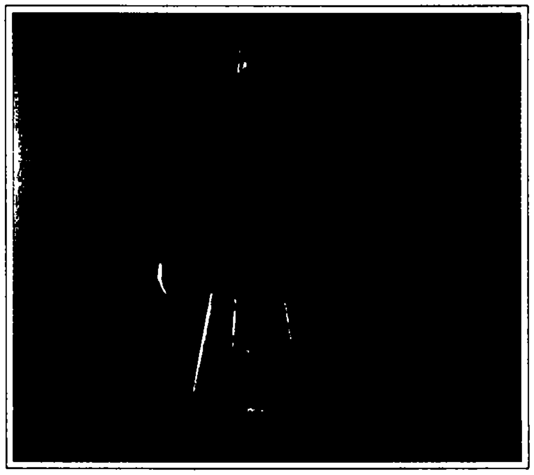
• We need a convenient way to get events–particularly one-time events–into our files. These come up so often that it would be nice not to have to memorize where to put them or how to format them.

• We need a way to mark tasks as done, and to fold undone tasks into the next day's to-do list. We are nearly out of space for this month, but next month, we'll add these to our design and go through the code to implement it. ▲

CALL FOR FREE CD ROM DEMO!

©1995 workstation solutions inc.

"I SHOULD HAVE USED QUICK RESTORE V2 INSTEAD OF UNIX UTILITIES."



*System administrator pondering his future.*

With Quick Restore V2, you can do lots of things that UNIX utilities simply can't do. ● Like lightning-fast restores and totally automatic backups. ● Quick Restore V2 controls data backup and recovery from any point in the network—without using NFS. ● Its on-line indexing, automatic volume location and direct-to-block positioning help you locate and restore your data fast. ● Our ISO-compliant recording format allows users to recover backed up data without Quick Restore V2 installed. ● And Quick Restore V2's GUI and command-line interfaces give you the power and flexibility to manage your network both on-site and remotely. ● Quick Restore V2 runs natively on all of today's popular workstation platforms and is compatible with today's leading tape formats. ● Best of all, Quick Restore V2 is simple to learn and easy to use. No needless bells. No useless whistles. ● Just solid, reliable backup and recovery software that will help you optimize the performance of your network. ● To evaluate how Quick Restore V2 can make a difference for you, give us a call at 1-800-487-0080 ext.266.

workstation solutions, inc

All company and product names are trademarks by their respective owners.
workstation solutions, inc. • one overlook drive • amherst nh 03031 • 603.880.0080 • fax 603.880.0696

**Circle No. 15 on Inquiry Card**