PLOD: Keep Track of What You're Doing

Hal Pomeranz – QMS, Inc.

ABSTRACT

PLOD (the Personal LOgging Device) is a simple text interface which allows System Administrators (and others) to keep a record of the work they do from day to day. The program was developed in Perl with device independence, flexibility, extensibility, and ease of use in mind. The user-interface is reminiscent of Berkeley mail, complete with many pre-defined tilde-escapes which perform various useful functions. Users may easily extend the program by defining their own personal escape sequences.

Introduction

Certainly it may be said of System Administration that those who forget history are doomed to repeat it. However, running from crisis to crisis, as many Administrators do, is not conducive to a good memory: in fact, it is unlikely that anybody reading this can remember with detail what they were working on even one week ago. The solution, then, is to let some external device be your memory and then provide an easy means to recall information from this external store. PLOD, the Personal LOgging Device, is one such solution.

PLOD is really just a simple Perl program which will read lines from the keyboard and store these lines, along with a date/time stamp, into a (encrypted, if desired) log file. By default, a new log file is started automatically at the beginning of every month, although the exact time period may be customized by the user. A number of additional commands are supported, either on the command line or via tilde-escapes ala Berkeley mail.

The design goals for PLOD included device independence, flexibility, extensibility, and ease of use. Everything was done so that PLOD would be a tool that would get used frequently (several times per day). If not, there would be little point in PLOD's existence. Implementing the program in Perl went a long way towards achieving device independence, since Perl runs on everything from Amigas to VMS machines. The interface was specifically chosen to be one that required no assumptions about display device (i.e., simple text only). Much of PLOD's behavior can be customized via environment variables or initialization files. Users may even add their own escape sequences.

The Theory of Logging

Many and varied are the arguments about personal logs. Some think that logs are a complete waste of time, while others find them invaluable. Some see logs purely as a defense against management scrutiny, some see them as a storehouse for accumulated wisdom and trickery. Some keep personal information in their logs along with details of their professional work, others maintain a more rigid dividing line. Some people do a daily brain dump into their logs before they go home, others prefer to update their logs several times during the day. Should the log be hard copy or on-line? Ultimately, the decision is a personal one and must fit the way you do business.

In many organizations, the job of System Administration is not well understood by those not in the System Administration group. Some System Administrators spend inordinate amounts of time justifying their existence to management. In a confrontation, it can be difficult or impossible to regenerate the countless tasks that go into keeping enterprise systems functioning smoothly. Having documentation, in the form of a log, created prior to such a confrontation can be an invaluable aid.

Hopefully, things will never get to the confrontation stage: logs can help here, too. If you are not required to publish a regular status report, publish one anyway. Extract items from your logs which are of interest to your user community and publish them in a condensed format (nobody wants to read more than one or two screenfuls of information that don't pertain directly to their own work). It often helps to place notices of system downtime in these reports, so users will pay more attention to them. Make your management and your users understand what you do. If ''they only notice you when something goes wrong'', then there's an important sales job that you as a System Administrator are not doing right.

Take your managers to lunch every month and bring your logs along. Show them items that have contributed to your products getting out on schedule. Demonstrate areas that could use more resources. ''We need more disk space'' does not impress people with signing authority, but ''We had eight hours of system downtime last month because I was playing partitioning games that could be solved by purchasing another $3,000 of disk'' usually does.

What should go into your logs? Ultimately, anything that you think is useful. Certainly, details on any problems you solve, including the syntax for any commands you're likely to forget before the problem re-occurs. If you have a great idea, or the germ of a great idea, but no time to implement it, put it in your log-- don't rely on a Post-It to be there in a month. Watching the growth of some files on your system? Put the checkpoint data into your log file. If you are angry at a user or your management, vent your frustration into your log and, after a suitable cooling off period, go back an look at what you wrote. Maybe it won't seem so important later, or maybe you can spot a way to correct the situation that you couldn't see in the heat of the moment. The best advice is to start off being as verbose and compendious as possible in what you log. Over time you will see what is useful and what is not. Also, there's the human tendency to become lazier over time: if you get in the habit early of logging large amounts of data, then you'll still be logging something when laziness sets in.

Your log should be updated often enough to keep it complete. If a nightly braindump works better for your schedule, then so be it. Other people find they need to update their logs more frequently to capture commands and output that they need to solve future problems. Sometimes it is simply impossible to update your logs as frequently as you would like (because you are spending all of your time fighting fires, for example). Update your log as soon as you can get out of crisis mode.

Perhaps the most divisive issue is whether logs should be kept electronically or in a notebook or other paper medium. On the plus side, it's easier to capture command output into an electronic logbook. Also most System Administrators spend large amounts of time in front of a keyboard, and so keeping a log on-line is often more natural than shifting gears to pen and paper. It's easy to run *grep* (or other text manipulation tools) against an electronic logbook, and rather more labor intensive to do it against paper. On the minus side, it's difficult to use your electronic logbook to help you solve the problem of getting the host which contains your logbook back on-line. Also, many System Administrators work in environments where they don't have consistent access to a single machine to keep their logs on, but in fact travel to many sites during a single day or a single week. For these people, a notebook and pen may be the only solution.

This ability to easily take your logs anywhere may be the biggest point in favor of hard-copy logging. It's also easier to doodle or draw pictures for yourself in a notebook when you're trying to work through a problem: paper logs can be much more free-form in style and content. One negative is constantly having to carry your logbook with you. If your logs are compendious, it may become difficult to physically carry much history around. Paper logs don't have an easy backup mechanism and can often be difficult to search for a small piece of information. If you do keep paper logs, be sure to use ballpoint or some other non-water soluble ink. Nothing is more depressing than having a logbook turned into an abstract watercolor design by a sudden rainstorm or unexpected trip into the sink.

Most importantly, choose a format and style of logging that is useful to you. If your log becomes spotty and incomplete, or if you only enter data into the log and never review your logs later, then you must ask yourself why you are spending time keeping a log at all.

PLOD User Interface

The user interface for PLOD looks a great deal like Berkeley mail. It is a simple text interface that many System Administrators are familiar with. For those who prefer a different interface, Paul Foley (*paul@ascent.com*) has developed an Emacs mode for PLOD which contains a number of nice features. Paul's *plod.el* file is distributed with PLOD or may be obtained from him directly.

The simplest use of PLOD is making one-line log entries from the command line:

```
host% plod A one-line log entry.
```

Typically, however, the user will want to make a multi-line entry. Simply invoke PLOD with no arguments: a date/time stamp will be printed, and the user may enter any number of lines of logging information, ending their entry with a bare period.

```
host% plod
09/13/93, 15:56 --
Here is an example of
a multi-line log entry
.
(eot)
host%
```

A number of tilde-escapes are supported in multi-line mode. For example, *~r filename* will append the contents of a file to the current log buffer:

```
host% plod
09/13/93, 16:00 --
~r .cshrc
.cshrc: 56 lines
(continue composing note)
Some more text could
go here, as desired.
.
(eot)
host%
```

A complete listing of tilde-escapes can be obtained by typing *~h* or *~?* while in multi-line mode.

By default, log entries are stored in encrypted format in the directory *.logdir* in the user's home directory. Individual log file names are four characters long in the format *YYMM*, and so a new file is started at the beginning of every month. Log entries within each file are separated by five dashes (-----) on a line by themselves. The dashes are followed by the time/date stamp for the log entry and the text of the entry itself. Note that the decision to encrypt, the location of the logging directory, the name of the log file (and therefore the cycle time before beginning a new file), and even the format of the date/time stamp are completely customizable (see the next section).

Several command line options are provided to allow users to edit or review old log files. These command line options also correspond to tilde-escapes in multi-line mode (i.e., the command line option *-E* is equivalent to the tilde-escape *~E*). The command *plod -P* will invoke the user's *PAGER* on the current log file (decrypting the file if necessary). Similarly, *plod -E* (*plod -V*) invokes the user's *EDITOR* (*VISUAL*) on the current log file. *plod -C* will simply *cat* the current log to the standard output, which can be useful for pipelines like *plod -C | lpr*.

All of the above command line options will optionally accept the name of an older log file, plus encryption key as needed. The default encryption key is *pl<yr><mn>od*, so to review the log file from August, 1993, the user would type:

```
host% plod -P 9308 pl938od
```

Note that the default file name is always four characters long, since the month portion is zero-filled, but the default encryption key does not use a zero filled month.

PLOD Customization

Customizing PLOD can be as simple as changing an environment variable or as complex as adding a new tilde-escape. Since PLOD is written in Perl, it is also easy to customize by modifying the script directly. This is not recommended, however, since modifications will have to be reapplied to each new release. PLOD has a rich set of hooks for customization, so please evaluate them carefully before modifying the distributed code. The author is always happy, however, to incorporate bug fixes or useful new features.

A number of defaults have been hard-coded into the PLOD program itself. When executed, PLOD will look for a system-wide */etc/plodrc* file for machine or site-dependent configuration information. After the */etc/plodrc* file has been evaluated, the user's environment variables are searched for personal configuration choices. Finally, PLOD checks the user's home directory for a *.plodrc* file which may contain additional or PLOD-specific configuration information.

PLOD recognizes the common *EDITOR*, *VISUAL*, *PAGER*, and *HOME* environment variables. The *LINES* environment variable controls how long the output of various tilde-escapes must be before *$PAGER* is invoked to provide output by screenful. This is somewhat like the *set crt = n* option in BSD mail.

A number of environment variables are used to control the location of various files that PLOD creates or manipulates. *LOGDIR* specifies where the log files are stored and *LOGFILE* is the name of the file in that directory. If *LOGFILE* begins with a slash (*/*), then *LOGDIR* is ignored and *LOGFILE* is taken to be the absolute path to the current log file. PLOD attempts to trap SIGKILL and SIGQUIT, and aborted logs are dropped into the file *$HOME/$DEADLOG* (unless, again, *DEADLOG* is an absolute pathname). *TMPFILE* is the absolute pathname of the scratch file PLOD uses for various operations. This file is always removed once the operation is completed.

CRYPTCMD is the absolute pathname of the command used to encrypt log files. The default is */bin/crypt*, which is not in the least secure but does provide protection against casual browsing. If encryption is not desired, simply set *CRYPTCMD* to be null. *KEYVAL* is the encryption key to be used by *CRYPTCMD*.

Finally, PLOD recognizes the *STAMP* environment variable to customize the format of the date/time stamp associated with each log entry. Note that it is generally inadvisable to put a line like

```
setenv STAMP "`date +%m/%d/%y, %H:%M`--"
```

in your *.cshrc* or other shell startup file, since this would imply the same date/time stamp for all log entries during the life of the shell. It is better to customize this variable in the */etc/plodrc* or *~/.plodrc* files, which are re-evaluated at each execution of PLOD.

Unlike most UNIX configuration files, the */etc/plodrc* and *~/.plodrc* files are evaluated as Perl code. Thus, if you wanted to customize the *STAMP* variable in your own *~/.plodrc*, you could write:

```
$STAMP = sprintf("%02d%02d%02d %02d:%02d",
         (localtime)[3,4,5,2,1]);
```

(line broken for display purposes) which would give you something like *DD/MM/YY HH:MM* in the European fashion.

While the requirement that the */etc/plodrc* and *~/.plodrc* be correct Perl syntax may be a barrier to novice users, it opens huge vistas of customization possibilities. Customization through environment variables is still an acceptable option for the non-Perl oriented.

Perhaps the nicest feature of evaluating the configuration files as Perl code is allowing users to extend PLOD by adding their own escape sequences. PLOD maintains all escape sequences as a global array, *%funcs*, of function pointers (actually, since Perl does not currently support the notion of a pointer, type globs are used). Functions are indexed in the array by the letter of their tilde-escape. For example, Figure 1 shows the definition of the *~!* escape which executes a shell command and then returns to PLOD. PLOD automatically passes any arguments following a tilde-escape to the appropriate function. The entire argument list is passed in as a single string which may have to be broken up if the function needs to process its arguments one at a time.

The scalar *$bang* is an optional descriptive message used by the on-line help function invoked with *~h* or *~?*. As Figure 2 shows, the function simply extracts a sorted list of all escape sequences defined in the *%funcs* array, and prints each one followed by the descriptive string referenced by the associated type glob. This example shows that it is trivial to bind the same function to multiple tilde-escapes. Note also the use of the *LINES*, *PAGER*, and *TMPFILE* customization variables.

In addition to the *%funcs* array, two other global data structures are available for user-defined escape sequences to manipulate. The list *@lines* contains the lines of text the user has entered for the current log entry. The first element of the list is always the date/time stamp. Figure 3 shows the code for the *~a* escape which will append the contents of the current log entry to a file.

The other data structure available is the list *@buffer*. This contains the last long Perl fragment entered using the *~M* escape. This escape sequence allows users to enter multi-line Perl fragments on the fly and have them incorporated into the current invocation of PLOD. It is possible to enter code that will manipulate *@lines*, and therefore the contents of the current log entry. It is even conceivable that such a code fragment could modify its own image in *@buffer* and produce some horrible hack of self-modifying code. This is not recommended.

Creating one's own escape sequences in not the limit of the power of this customization mechanism. Perhaps the */etc/plodrc* file could contain code to evaluate a site-wide */usr/local/etc/plodrc* before any machine-specific customizations in */etc/plodrc*:

```
sub bang {
   local($cmdline) = @_;
   system "$cmdline";
   print "(continue composing note)\n";
}
$bang = "cmdline\tExecute system command and return";
$funcs{'!'} = *bang;
```

Figure 1: Code to implement escape mechanism

```
sub helpuser {
  $long=(scalar(keys %funcs)>=$LINES) && open(TMP, ">$TMPFILE");
  for (sort keys %funcs) {
    *info = $funcs{$_};
    if ($long) {
       print TMP "~$_ $info\n";
    }
    else { print "~$_ $info\n"; }
  }
  if ($long) {
    close(TMP);
    system("/bin/cat $TMPFILE|$PAGER");
    unlink "$TMPFILE";
  }
}
$helpuser = "\t\tPrint this message";
$funcs{'h'} = *helpuser;
$funcs{'?'} = *helpuser;
```

Figure 2: On-line help function

```
if (-e "/usr/local/etc/plodrc") {
   eval { do "/usr/local/etc/plodrc"; };
   die "*** Error in " .
      "/usr/local/etc/plodrc:\n$@" if $@;
}
# do machine-specific customizations here
```

Further chicanery is left as an exercise to the interested reader. If you develop something good, please inform the author.

Conclusion

PLOD is a living breathing entity which owes many of its features to the ideas of others. The author greatfully acknowledges the contributions of David W. Crabb (crabb@phoenix.Princeton.EDU), John Ellis (ellis@rtsg.mot.com), Mike Lachowski (mlachow@erenj.com), Eric Prestemon (ecprest@pocorvares.er.usgs.GOV), Erik E. Rantapaa (rantapaa@math.umn.edu), and James Tizard (james@ringo.ssn.flinders.edu.au). The original idea for PLOD came from a conversation with Bill Mendyka (mendyka@dg-rtp.dg.com) at LISA VI.

The author would not dare to suggest that PLOD is the logging tool for everybody. A goodly number of people find it useful. You may choose to keep a journal via some other mechanism, but always keep a record of the work you do. The payoff may be infrequent, but is often enormous.

PLOD v1.6 has been recently posted to comp.sources.misc and comp.lang.perl. The comp.sources.misc newsgroup is archived at many FTP sites. Scripts posted to comp.lang.perl are archived at coombs.anu.edu.au. In addition, a shar file is available via anonymous FTP from gatekeeper.imagen.com in the directory /pub/plod. If all else fails, the author will be happy to satisfy email requests for the current version.

Author Information

Hal Pomeranz is a victim of being in the right place at the right time. A grant allowed his undergraduate institution to purchase a state of the art (at the time) workstation network for the amusement of the student hackers who were nominally supposed to keep things running. This was apparently enough of a credential to launch a career that has included System Administration stints at AT&T Bell Labs and the NASA Ames Research Center. Hal Pomeranz is the author of the fabulously overlooked *Perl Practicum* column for the USENIX *;login:* Magazine. His current email address is pomeranz@aqm.com.

```
sub appendfl {
  local($file) = @_;
  if (!open(OUTP, ">> $file")) {
    warn "*** Could not append to file $file\n";
    return;
  }
  print OUTP @lines;
  close(OUTP);
  print "Wrote ", scalar(@lines), " lines to file $file\n";
  print "(continue composing note)\n";
}
$appendfl = "file\t\tAppend contents of buffer to file";
$funcs{'a'} = *appendfl;
```

Figure 3: Implementation of ~a escape