Open Computing ‘‘Hands-On’’: ‘‘Wizard’s Grabbag’’ Column: January 94

Getting a Bang Out of Unix

Automatically generating temporary files, saving money by sending e-mail in batches, and revisiting the rounding error in `ap`

By Becca Thomas

With the start of the new year and the premiere of *UnixWorld’s Open Computing*, we are taking this opportunity to make some improvements to ‘‘Wizard’s Grabbag.’’ This month we begin a new experiment: selecting and presenting items from O’Reilly & Associates’ *Unix Power Tools*, a book that presents tips, scripts, and techniques like those published in this column. From time to time I’ll publish interesting items useful to most readers and try to locate related material that didn’t make it into the book. For our first selection we present a script that automatically generates temporary files for command operations requiring file arguments.

In the ‘‘From the Wizard’s Den’’ section I provide a tip that can help some site administrators configure their system to save money by batching outgoing electronic mail.

Finally, we have ‘‘Closing the Feedback Loop,’’ which contains selected contributions related to recent columns. Harvey Davies provides some feedback regarding his `ap` program that was originally published in my August 1992 column.

Bang Around the Command Line

Dear Editor:

Here’s a little tool that should prove useful to anyone who works from the shell command line. Some commands or command modes require you to provide an argument that names a file instead of supplying that data on the standard input or the command line. Such files are often created just for the command operation and then erased. The `!`, or ‘‘bang,’’ script [Listing 1A] can be used to name a temporary file that is populated by running a specified command.

I’ll start with an example that can be simplified with `!` [Listing 1B]. Here, we manually create two temporary files that contain sorted versions of the input data. With the `comm` program, we compare the contents of these two sorted files. The default `comm` report--no invocation options specified--consists of three columns: unique items in the first file, the second file, and common to both files. The next example [Listing 1C] shows how to collapse the four command lines [used in Listing 1B] into a single command using `!`.

We use `!` to expand tab stops in files before they are compared by `diff` [shown in Listing 1D]. The `diff` output from files containing tabs may appear misaligned because of the extra characters (>, <, +, and so forth) that `diff` prefixes to the output line. If you expand the tabs to spaces before using `diff`, proper

alignment of the output can be maintained.

Some popular paginators like `more` can't back up when they're reading from a pipe. Perusing the output from a pipe can be painful if you can't reverse direction to reread something you saw earlier. For instance, `ls -l | more` displays a screenful at a time from `ls - l` output, but if something of interest has scrolled past, you can't review it unless, of course, you're using a display system with reverse- scroll capability. However, if you invoked `more` using `more '! ls - l'`, you can review the result as much as you wish by moving around the temporary file with `more` file- perusal commands.

The main problem with the `!` script published in *Unix Power Tools* was that it didn't remove the temporary file it created. The script can't remove this file as it exits because then the temporary file might not be found when needed by the ''driver'' command [like comm in Listing 1C]. The new version in Listing 1A waits 30 seconds for the ''driver'' command to start and then erases the temporary file. Your readers could take this approach farther by adding an option that changes the delay time. For instance, `! -600` would wait 10 minutes (600 seconds) and `! -0` would never remove the file.

Jerry Peek / O'Reilly & Associates Inc. / Sebastopol, Calif

Explanation of `!` script. Line 7 defines the parent directory, and line 9 defines the full path name of the unique temporary file. Here, `$$` will be replaced by the process ID number of the shell running the instance of the `!` script.

Line 12 sets a trap to erase the temporary file when the script is terminated by one of the specified signals: hangup (1), interrupt (2), or software termination (15). The quit (3) signal is not trapped; it will abort `!` without removing the temporary file.

Lines 15-19 process the command line. If no invocation arguments were specified, line 16 reports the correct usage to standard error, line 17 outputs the path name of the temporary file, and the script terminates. Hopefully, when the driver command processes the (missing) temporary file, it will exit gracefully.

Line 22 runs the specified command line, which is contained in the `$@` variable, and redirects its output to the temporary file. Line 23 reports the temporary file name on the standard output for use by the driver command.

Line 26 is important. This line closes the standard output so the shell that's ''filling the backquotes'' with the output of the `!` program won't wait for more output. Without this line, the calling shell would wait until the subshell on line 27 exits and closes its standard output.

Line 27 runs a subshell in the background to erase the temporary file after 30 seconds.

Big Bang Contest!

We're starting a new contest this month. Send in your shell command line examples for using the `!` program discussed here. The most clever or useful contributions will be featured in a future column.

From the Wizard's Den: Batch UUCP Mail and Save