UnixWorld Online: ‘‘Wizard’s Grabbag’’: Column No. 003

Solving Email Hide-and-Go-Seek

This month Larry Ruane contributes an email search tool that--unlike `grep`-- treats email messages as a unit, allowing effective searches.

Dear Editor:

Many UNIX tools, such as `grep`, are line-oriented, but often the data comprises multi-line units. Email files (folders), for example, consist of a concatenation of multi-line email messages. Each email message begins with a line that starts with the string ‘‘From’’ followed by a space.

Using `grep` to search such files can be frustrating. For example: you remember saving an email message mentioning a great pizza restaurant near a softball field. A search for ‘‘pizza’’ or ‘‘softball,’’ or either at the same time using

```
$ egrep 'pizza|softball' mbox
```

produces an overwhelming amount of output (because you are fond of both of these topics). So you require both to appear together using

```
$ grep 'pizza.*softball' mbox
```

But this attempt nets you nothing, because both patterns must be on the *same line*. You try reversing the order of ‘‘pizza’’ and ‘‘softball,’’ but no luck. So you finally resort to bringing the mail file into an editor, and searching for one string or the other, sifting through a lot of irrelevant stuff. Sound familiar?

The `msearch` (for mail search) command solves the problem. Given a set of regular expressions, it scans a mail file, looking for email messages containing *all* the given expressions, *anywhere* within the message. It prints the message number, followed by the ‘‘From’’ and ‘‘Subject’’ lines of selected messages. It’s easy to modify the script to print other information, such as the date, or the entire message.

The user interface took a little thought because there are two variable-length lists: regular expressions, and email file names. I decided to separate them with a hyphen (-) argument, with the regular expressions coming first, so that the hyphen and the file names are optional, defaulting to the standard location for the read-mail file, `$HOME/mbox`. If you specify more than one such file, each output line is prepended with the file name (just like `grep` does for multiple-input files).

You can apply the technique to other areas. For example, we search our problems database with a similar script.

Here are some sample command-line usage examples:

Search `$HOME/mbox` for ‘‘pizza’’:

```
% msearch pizza
```

Message must contain both ‘‘pizza’’ *and* ‘‘softball’’:

```
% msearch pizza softball
```

Same, but either upper or lower case ‘‘softball’’:

```
% msearch pizza '[Ss]oftball'
```

Same, but look for either ‘‘pizza’’ *or* ‘‘softball’’, and also require ‘‘beer’’:

```
% msearch 'pizza|softball' beer
```

Search the file `/var/spool/mail/lr`:

```
% msearch pizza softball - /var/spool/mail/lr
```

Look through all files in the `mailfiles` directory:

```
% msearch pizza softball - mailfiles/*
```

Sample output: message number, from- and subject lines:

```
67 beccat@magicats.org (Becca Thomas) A new pizza place
70 lr (Lawrence M. Ruane) Re:  A new pizza place
73 beccat@magicats.org (Becca Thomas) Re:  A new pizza place
```

Explanation

Lines 11 through 18 generate a list of `awk` statements of the form:

```
/pattern1/ { found[1] = 1 }
/pattern2/ { found[2] = 1 }
 ...
```

and assigns them to the shell variable `awkstmts`. The `found` flags indicate whether the corresponding pattern was seen at least once while scanning a particular email message. The `sed` filter prepends a backslash to all slashes that occur in the user’s patterns, which is required by `awk`.

Lines 25 through 37 sets the `files` shell variable to the list of files to search, either specified by the user (line 30) or using `$HOME/mbox` (line 34) as the default case. The `printname` shell variable will be reset to one (`1`) in the case of multiple files, which will tell the `awk` program to prefix each line of output with the file name.

Next, we process the input files sequentially and independently (line 40), running the `awk` program (lines 43-66) on each. When this program recognizes the beginning of an email message (line 44), it determines whether the previous email message matched all the patterns, which is the case if all the `found` flags are set; if so, an output line identifying the previous file is printed.

Lines 59 through 64 save the first ''From:'' and ''Subject:'' lines of the current email message for later use. The actual ''From:'' and ''Subject:'' strings are removed using `substr()` to reduce output clutter. (The ''From:'' line, with the colon, always indicates the human sender of the message; the initial ''From'' line can be something else like ''Mailer-Daemon''.) Only the first ''From:'' and ''Subject:'' lines are saved, in case an email message includes another message.

Next come the dynamically generated statements (line 65), which set `found` flags if patterns are matched. The ''From'', ''From:'' and ''Subject:'' lines are included in the pattern search because `awk` pattern matching ''falls through'' (one line can match multiple patterns).

It would have been more straightforward to pass the expressions as variables to `awk`, but this approach doesn't work because matching must be done with fixed patterns.

The `awk` program is enclosed in double quotes so the values of the `npat` and `awkstmts` shell variables are available inside the `awk` program. However, this approach requires that one escape all dollar signs and double quotes with backslashes.

The extra ''From '' that is appended to the email file (line 42) acts as a sentinel so we don't have to duplicate the code in lines 45-54 in an END section for the last email message. (We could have put that processing into a function that is called from two places, but only ''new'' `awk` recognizes user-defined functions).

Larry Ruane / Programmer / Minimus Software, Inc. / Parker, Colorado / lr@minimus.com

---

Edited by Becca Thomas / Online Editor / UnixWorld Online / beccat@wcmh.com

Go To Content Search

Last Modified: Tuesday, 22-Aug-95 15:51:43 PDT