Wizard Boot Camp, Part Ten

Jerry Peek

This month, we'll end the Wizard Boot Camp series with a third article about utility programs that you should know about — and some not-so-obvious ways to use them.

csplit, split

Some large files — including archives, multipart email messages, business information for a large set of customers, and data files with repeating patterns — may need to be split into smaller "chunks" for reorganization, easier storage or transport. That's what *csplit* (1) is for. Give it a pattern, an offset, a repetition count, and/or a line number, and it will parse an input file into a series of smaller output files.

By default, *csplit*' s output files are named *xx00*, *xx01*, and so on. To rearrange a big file, you can simply *cat* those output files together in a different order. Here's a simple example using the shells' curly-brace operators:

```
$ csplit oldfile '/pattern/' '{5}'
  ...creates files xx00 - xx05...
$ cat xx0{5,0,2,1,3,4} > newfile
$ rm xx0[0-5]
```

Using a shell wildcard operator, like `xx0[502134]`, wouldn't work with *cat* because it sorts filenames in alphanumeric order. We'll see more about *csplit* in a future column — part of a new series about handling text files.

In these days of multi-gigabyte files and terabyte (or larger) filesystems, a file can still be "too big." For instance, if you're trying to send a huge file over a network connection that often times out, even the automatic-retry ability of some file transfer utilities won't get all of the file through.

Splitting the file into smaller chunks, then reassembling the pieces on the other end, can save headaches and time. The *split* utility is great for that. It splits input — either files or *stdin* — into equal-sized files named (by default) *xaa*, *xab*, *xac*, and so on. (The last file may be smaller.) For instance, on the sending side, split the file into 1-Megabyte chunks:

```
$ ls -l kcpr.mp3
-rw-r—r— ... 70535208 ... kcpr.mp3
$ split -b1m kcpr.mp3
$ ls -l x??
-rw-r—r— ...  1048576 ... xaa
-rw-r—r— ...  1048576 ... xab
...
-rw-r—r— ...   280616 ... xcp
```

After you've transmitted all of the files, do a quick check on the receiving side to be sure they all have the same size. Then *cat* them together.

```
% ls -l x??
-rw-r—r— ...  1048576 ... xaa
...
-rw-r—r— ...   280616 ... xcp
% cat x?? > kcpr.mp3
% rm x??
```

Using a checksum program like *md5sum* on both the original file and the reconstructed version can give you more confidence that the files are the same.

file

In general, Linux doesn't require filename extensions such as *.exe* or *.txt* to know what to do with a file. Executable files — files whose execute bit was set by, for instance, *chmod* — start with a two-byte *magic number*. The best-known magic number is probably `#!`, which lets you specify the file's interpreter (*/bin/sh*, */usr/bin/perl*, etc.)

The *file* utility will guess what type of data is in many types of files. That includes unidentified or mis-identified files you receive attached to an email message. For example:

```
$ file mystery-file.dat
mystery-file.dat: PDF document, version 1.3
```

locate, updatedb

If your system runs the *updatedb* utility (from *cron* or otherwise), you'll have a database on your system that lets you find files or directories by name much faster than a command like `find /-name...`. The *locate* utility searches that database. For instance, to find all files and directories whose name includes *examples*:

```
$ locate examples
/usr/bin/dh_installexamples
/doc/adduser/examples/INSTALL
/doc/zsh/compctl-examples.gz
...
```

The *locate* manpage has details — including how to use wild-

card characters to restrict matching. But some uses aren't quite so obvious. If you think about the syntax of the pathnames you want, you can often use string-matching to find them. To locate any file in a directory named *examples*, for instance, use a search pattern that matches a directory name in a pathname:

```
locate /examples/
```

To do more sophisticated pattern-matching, you can filter *locate*'s output through a tool like *grep*. You can even dump the entire database:

```
locate / | grep ...
```

To search the contents of files or directories with a certain name, pipe the list of pathnames to **`xargs grep`**. For instance, to search all files with a name including *foo* for text containing *bar*, try this:

```
$ locate -0 foo | xargs -0 grep -Hs bar |
cat -v
/usr/local/bin/ascript:echo "back at the
bar..."
/usr/share/doc/m4/examples/foo:bar
Binary file
/usr/share/emacs/21.4/lisp/mail/footnote.elc
matches
```

We're using the `-0` (zero) option for both *locate* and *xargs*; it separates pathnames with NUL characters, which avoids problems caused by "special" characters in filenames. The *grep* option `-H` makes tells *grep* to always output a filename (even if *xargs* happens to pass only one filename to *grep*). And the *grep* option `-s` keeps *grep* silent about arguments that are directory files or unreadable files. The `cat-v` avoids sending "unprintable" characters to your terminal.

You may want to run *updatedb* multiple times to make more than one *locate* database: one for all users, one for system files, one for each user's home directory (readable only by that user), and so on. In that case, you and other users may want to set the *LOCATE_PATH* environment variable to tell *locate* which databases to search:

```
$ grep LOCATE_PATH /etc/profile
LOCATE_PATH="/var/cache/locate/locatedb:/us
r/local/locatedb/$USER"
export LOCATE_PATH MANPATH PATH
```

look

Everyone knows about the *grep* utilities. Less well-known is *look*. It searches the first characters on each line of data — like a *grep* search starting with the anchor character ^ (caret). *look* defaults to a linear (sequential) search or uses binary search with its `-b` option. A binary search rapidly searches a sorted data file — even a very large one.

By default, *look* searches the system word list. That's handy for checking spelling. If there are words that start with the argument you type, you'll see them:

```
$ time look gas
Ga's
Gascony
Gascony's
gas
gas's
...

real     0m0.088s
user     0m0.065s
sys      0m0.009s
```

The *time* utility shows system resource usage. Compare the linear search to a binary search:

```
$ time look -b gas
gas
gas's
...

real     0m0.004s
user     0m0.001s
sys      0m0.003s
```

Note that a binary search matches the search string exactly as a prefix. To use the `-d` (dictionary order) or `-f` (case-insensitive) search options with a binary search, the file must be sorted in that order!

OTHER USES FOR THE SYSTEM WORD LIST

The system word list, often in the directory /usr/share/dict, can be searched with more than look. Need an eight-character word starting with n plus a vowel, and ending in the suffix "ly"?

```
$ cd /usr/share/dict
$ grep -i '^n[aeiou]....ly$' words
narrowly
normally
```

You can also create files to search later with *look*.

```
$ nice sort -t: -k1,1 invoices_raw >
invoices
  ...later...
$ look 172-102i: invoices
172-102i:2008-05-17:Smith, Alan:...
```

nice reduces the scheduling priority of *sort*, reducing the impact that process has on other users and processes.

mail, sendmail,...

Want to send a quick email message? Years ago, everyone used *mail* to send and read plain-text messages. Now it's half-forgotten but still useful for sending email. Also, if *sendmail* or another interface to the system mail transfer agent is available (and configured correctly), you can send email with almost-complete control over the message header and body.

Different versions of *mail* and *sendmail* support different options. Study and test to see what works on each system.

mail sends text from the keyboard or standard input. From the keyboard, end text with CTRL-D or a dot (period) on a line by itself. The `-s` option specifies a subject:

```
$ mail joe@foo.xyz ann@bar.xyz
Subject: test message
test test
that's all
.
$ myprog | mail -s 'myprog output'
joe@foo.xyz
$ mail -s 'afile contents' ann@bar.xyz <
afile
```

With most versions of *mail*, you don't supply a message header. But *sendmail* sends a complete message with a full header and body. (Header and body are separated by exactly one newline character: an empty line.) The `-t` option tells *sendmail* to read addresses from the message header. It's a good idea to add the `-f` option with a return address (the SMTP FROM, or envelope sender).

Before sending a message with *sendmail*, we'll check it with `cat-e`, which puts `$` at the end of each line. It shows that the line between the header and body is empty. *sendmail* typically isn't in users' shell search paths, so you may need to use its full pathname:

```
$ cat -e msg
From: Joe Smith <joe@foo.xyz>$
To: Ann Doe <ann@bar.xyz>,$
```

```
  zoe@plum.xyz, ron@kumquat.xyz$
Subject: Test from sendmail$
$
test test$
$ /usr/sbin/sendmail -fjoe@foo.xyz -t < msg
```

A good way to send batches of messages with MIME-formatted bodies is with *sendmail* and a shell loop. *Listing One* demonstrates the setup: a message and a list of addresses, one per line. The loop prepends a `To:` address field to the stock message, then sends the message to that address with *sendmail*. A `sleep 3` command pauses three seconds after each message to avoid flooding the system mailer; this may not be needed:

```
$ cat msg
From: Joe Smith <joe@foo.xyz>
Subject: Please vote for me
MIME-Version: 1.0
Content-Type: multipart/mixed; ...
...
$ cat addrs
"Ann Doe" <ann@bar.xyz>
"Zoe Jones" <zoe@plum.xyz>
...
$ cat addrs |
> while read addr
> do
>   echo "To: $addr" |
>   cat - msg |
>   /usr/sbin/sendmail -t -fjoe@foo.xyz
>   sleep 3
> done
```

There are tools available to MIME-encode messages, but one of the easiest ways is to send a message to yourself with a MIME-capable program like Thunderbird, save the message in a file, and clean up its header before re-sending it with *sendmail*.

od and friends

Being able to really see what's in a file, byte by byte, can help you debug a lot of problems. The Swiss Army Knife of byte-by-byte viewing is *od*. Here's an example: the first few bytes of a JPEG image file:

```
$ od -c -w6 clinica.jpg
0000000 377 330 377 340   \0 020
0000006   J   F   I   F   \0 001
0000014 001 001   \0   H   \0   H
0000022   \0   \0 377 341   \0 026
```

With its `-c` option, *od* shows bytes as characters if they have ASCII representations; it shows escape sequences for others (like `\0` for NUL bytes) and octal values for the rest.

Linux has useful utilities other than *od*:

- `cat-vte` encodes only "non-printable" characters and TABs, and (as we've seen earlier) it marks the ends of lines with `$`.
- The flexible pager *less* will ask "*filename* may be a binary file. See it anyway?", then show hexadecimal values of "non-printable" data in reverse video.
- The *strings* utility searches files for text strings — for instance, finding filenames embedded in an executable binary.

There's more about *od* and non-text files in "Bits and Pieces: Comparing Binary Data (and More)", online at http://www.linux-mag.com/id/4089/.

ssh/scp

There's a lot to say about these powerful networking utilities, and much of it has been said in other *Linux Magazine* columns over the years. The Power Tools column "Transfer Tips, Part I," at http://www.linux-mag.com/id/1312, shows many command-line examples. The most important point to take away from that column is that *ssh* starts a shell on the remote system and that:

- It's helpful to understand how wildcards and other shell operators are handled on both the local and remote systems,
- Your command line can redirect the standard input and standard output of either or both the local and remote shells,
- The remote shell has a context (shell and environment variables, a current directory that you can change with *cd*, and more).

In Closing

The ten articles in this "Wizard Boot Camp" series have covered a lot of ground: certainly not everything you need to know about Linux systems, but (we hope) a good starting point. Use your wizardly Linux powers well…and keep digging for more!

Jerry Peek is a freelance writer and instructor who has used Unix and Linux for more than 25 years. He's happy to hear from readers; see http://www.jpeek.com/contact.html.