Fix Images Fast with *Netpbm*

By Jerry Peek

Imagine that:

- You're back from your summer vacation with 500 TIFF photo files that you'd like to convert to JPEG. You also need to rotate a lot of them. However, you don't want to open them one-by-one in your camera's photo editing program.

- The process-control system on your factory floor takes one picture of each finished product. You need to build HTML web pages with clickable thumbnails of those photos, ten per page, and correct the perspective distortion caused by the inconvenient camera angle.

- Your fax-to-email gateway sends new faxes as a TIFF file in a MIME email message. When you travel, you want those faxes converted to a PNG file on your web server, so you can read your faxes with any web browser.

- You've got graphics files in some obscure format that you need to convert to a more common format.

- You'd like to add a copyright label to your images, or add a shadow to the images so they look like they're floating, or normalize image contrast, or concatenate images, or overlay images, or separate images into individual planes (R, G, B).

A capable graphics program like the GIMP can handle many of those tasks with its scripting language, *script-fu*. But there's another solution: a big set of small utility programs called *Netpbm*. You can run these from the command line or call them from another program to convert almost any image format into another and do all sorts of other operations.

Let's dig into this package of powerful programs — and some shell "glue," such as loops and tests that help to tie them together.

Introducing Netpbm

Netpbm is a package of more than 200 image manipulation programs. It started life in the 1980s as *PBMplus*, the Portable Bitmap Package. Since then, it's been expanded and updated. The software's current maintainer is Bryan Henderson and the project's home page is http://netpbm.sourceforge.net.

Netpbm has been ported a lot: there are RPMs and binary distributions for *Linux*, *Unix*, *Windows*, *Mac OS X*, and many more. Some vendors have also added their own tweaks and "bug fixes" (not all of which the maintainer agrees with, as you can read on the home page).

Most of Netpbm's hundreds of programs convert to or from one of three base graphics formats: *PBM*, *PGM*, and *PPM*. All of these formats describe images as a matrix of pixels. In PBM (portable bitmap) images, the pixels are black or white; in PGM (portable greymap), pixels are a shade of grey; and in PPM (portable pixmap), pixels are full-color.

A program name tells you which formats it handles. For instance, the *picttoppm* program converts *Macintosh* PICT files to PPM, and *ppmtompeg* converts a series of PPM images into an MPEG movie.

A lot of the program names include *pnm*, which stands for "portable anymap format." These programs can handle PBM, PGM, and PPM formats.

For instance, *tifftopnm* will convert a TIFF-format fax file (which has black and white pixels) into PBM, and will also convert a color TIFF photo file into PPM. Because programs that handle PNM automatically choose the right base format, you often don't need to think about it, as you'll see.

PAM is a new, fourth Netpbm format that can handle any type of image data. Some new and useful utilities operate on PAM files.

For example, *pamperspective* corrects perspective distortion in an image, which is handy when you need to fix a whole series of images with the same distortion. But older Netpbm versions don't handle PAM yet, and its utilities work in the same way as older ones, so we won't spend much time on it.

Shell Glue, Part I

Like other well-designed Unix-type utilities, most of the Netpbm programs read from standard input (unless you give them a filename) and write image data to standard output. So, you can call Netpbm programs from another program or script, feed them input, and read their output. If you're using a shell (from a script or the command line), the shell's redirection operators — like `|` and `>` — let you chain a series of Netpbm programs to do image conversions on-the-fly, often in a single step.

For example, you might have a BMP-format screen shot from a Windows system. You'd like to make it into a PNG file to view in your web browser. How? Search the list of Netpbm programs to find one whose name starts with *bmp*. Or, if your shell supports command-name completion, you can type the first few letters of the program name (here, *bmp*) at a shell prompt and then press TAB to see which program name matches. (If more than one name matches, press TAB again to see them all — or, in *tcsh*, use CONTROL-D.)

To convert from BMP to PNG, you'll read the BMP file with *bmptopnm* (or the older program *bmptoppm*) and write the PNG file with *pnmtopng*. A pipe connects the two programs: *pnmtopng* reads the standard output of *bmptoppm*. You can also redirect the standard output of *pnmtopng* to a file.

Here goes:

```
$ bmptoppm ss.bmp | pnmtopng > ss.png
bmptoppm: Windows BMP, 463x503x24
bmptoppm: No palette for 24 bit bitmap
$ ls -l ss.*
-rw-rw-r— ... 700230 Aug 4 12:39 ss.bmp
-rw-rw-r— ... 19296  Aug 5 10:22 ss.png
```

bmptoppm tells you what it's doing. If you don't want verbose output, you can redirect the standard error of *bmptoppm* to */dev/null*, the Linux "bit bucket" device that throws away whatever you send to it:

```
$ bmptoppm ss.bmp 2> /dev/null | \
  pnmtopng > ss.png
```

The operator **2>** is explained in the June 2004 "Power Tools" column, "Execution and Redirection," which you can read online (beginning in September 2004) at http://www. linux-mag.com/2004-06/power_01.html.

What if the BMP file had been compressed with *gzip*? No problem! Use *zcat* to uncompress on-the-fly, then pipe the output of *zcat* to *bmptoppm*:

```
$ zcat ss.bmp.gz | bmptoppm | \
   pnmtopng > ss.png
bmptoppm: Windows BMP, 463x503x24
bmptoppm: No palette for 24 bit bitmap
```

When you first see a long command line like that, it may look tedious and make you think of opening the GIMP so you can point-and-click your way through the conversion. That might be the fastest choice — at first, anyway — until you get used to typing these little names and pressing RETURN to do the job in a flash. But what if you have a directory or a whole CD-ROM full of these images? We'll look at that in "Shell Glue, Part II" below.

Cropping, Scaling, Getting Info, and More

Many Netpbm programs convert image formats, while other programs in the suite measure the images (showing the dimensions or making a histogram, for instance) or edit the image non-interactively. The long list of programs is worth browsing through. Some are esoteric (want to make blue-green images for 3D glasses?), but there are some you may use often.

The *pnmfile* program (obsoleted by *pamfile*) reads Netpbm files and writes a description of each to standard output. This is handy for programming, though it can also give you a quick summary of a Netpbm file that's sitting in your directory.

You can chain as many Netpbm programs as you need to make the output you want

pnmscale (superseded by *pamscale*, which not everyone has yet) changes the size of an image in any of a number of different ways. You can do obvious things, like scaling a 640x480 image to 320x240, but *pnmscale* can also do smart automated conversions for a group of arbitrary-sized photos that you want to scale all at once. For instance, the `-xysize` flag fits any image into the specified size without changing its aspect ratio.

Another handy program is *pnmsmooth*. It "smooths" the jagged edges you get by enlarging an image too much (where individual pixels become square blocks of pixels). This may not work well for line drawings, but it does a decent job on photos.

So, for instance, to read a BMP file, triple its size, smooth it, and write it as a PNG file:

```
$ bmptopnm file.bmp | pnmscale 3 | \
  pnmsmooth | pnmtopng > file.png
```

You can chain as many Netpbm programs as you need to make the output you want. But doing multiple operations on a single image like this — especially when you can't tweak the settings as you're looking at the result — will probably make you wish for an interactive program like the GIMP. That may not be true if you're converting batches of images from a shell prompt, though, or writing a script that uses Netpbm programs. Let's see more examples of the latter.

Shell Glue, Part II

One of the great things about shells — especially shells with interactive command-line editing — is that they let you write programs on-the-fly from the command line. There's no need to edit a script file: just do the job from a shell prompt. As the job gets more complex, though — or as the idea evolves into something you'll want to use again later — you might decide to paste the code into a file and make it into a script.

Here's an example of that process: you've got a CD-ROM full of images of various types and sizes, and you need a web page that displays those images. The web page should have thumbnails, which the user can click on to get a full-sized view from a PNG file.

The code will have a loop that steps through the files on the CD-ROM and operates on each one. If the CD-ROM is mounted on */mnt/cdrom*, a loop starting with `for file in /mnt/cdrom/*` could do the job. But if the CD has thousands of files, the `/mnt/cdrom/*` could expand into a list that's too long for the shell to handle. (If the files are spread across subdirectories, you'd need to handle that, too.) So we'll use a redirected-input `while` loop that can handle an unlimited number of files. (Redirecting input to a `while` loop is described in the May 2004 article "Great Command-line Combinations," online at http://www.linux-mag.com/2004-05/power_01.html).

Overall, the code will look something like this, with a pipe that feeds a list of files to a `read` command that stores the image pathnames, one by one, in the shell variable named `ipath`:

```
generate-list-of-files... |
while read ipath
do
  ...process $ipath...
done
```

The *find* utility can generate a list of files. (You could add a *find* operator like `-name '*.jpg'` to select only certain files.) The filenames may not be in order, but that's easily solved by routing the output of *find* through *sort*.

If you were doing simple conversion (without building a web page or sets of thumbnails and images) you'd need just a line or two more of code, so you could type this little bit of code at a shell prompt. This case is a little more complicated, so the code should go in a script file.

We'll put thumbnails in the subdirectory *thumbs* and full-size images in the subdirectory *images*. We'll collect the HTML code into a file named *index.html* by redirecting the standard output of the *while* loop — as explained in the June 2004 column "Execution and Redirection." (Because old-style HTML has a simple syntax, we'll use it to make our example easier to read. Of course, your code could generate modern HTML using CSS, and so on.) The code is in *Listing One*.

We've omitted code to put beginning and ending HTML in *index.html*, as well as to arrange thumbnails into table rows. We've also skipped niceties like adding the height and width of each thumbnail to each HTML `<img>` tag because it's a bit ugly. You could add that with code like *Listing Two*.

LISTING ONE: Script to convert images and build web page

```
#!/bin/sh

# Make temp PNM file, arrange cleanup:
temp=/tmp/ONVERTER$$
> $temp
chmod 600 $temp
trap 'rm -f $temp; exit' 0 1 2 15

find /mnt/drom -type f -print |
sort |
while read ipath
do
  # Make $ipath (like /mnt/drom/foo.jpg)
  # into $name (foo.jpg) and $base (foo):
  file=${ipath#/mnt/drom/}
  base=${file%.*}

  # convert image to PNM format and
  # make or copy full-size PNG image:
  case "$file" in
  *.jpg | *.jpeg)
    jpegtopnm "$ipath" > $temp
    pnmtopng $temp > "images/${base}.png"
    ;;
  *.png)
    pngtopnm "$ipath" > $temp
    p "$ipath" images
    ;;
  *) eho "$0: can't convert $file" 1>&2
     exit
     ;;
  esac

  # Make thumbnail within 100x100 boundary:
  pnmsale -xysize 100 100 $temp |
    pnmtopng > thumbs/${base}.png

  # Make HTML code (redirected to index.html):
  echo "<a href=\"images/${base}.png\">
    <img src=\"thumbs/${base}.png\"></a>"

done > index.html
```

LISTING TWO: Storing *pnmfile* output into shell variables

```
eval $(pnmfile $temp |
  sed 's/.*\([0-9]*\) by \([0-9]*\).
    */width=\1 height=2/')
```

The code in *Listing Two* runs *pnmfile*, parsing its output with *sed* and writing that into a shell command that sets shell variables. Let's take that step-by-step.

The output of *pnmfile* looks like this:

```
file: PPM raw, 450 by 600  maxval 255
```

sed uses a regular expression that finds the width and height values around the word `by`, then outputs a line like this:

```
width=450 height=600
```

Shell command substitution — the operators `$( )` — give those two variable settings to the shell's *eval* command, which interprets them. In this example, the variable *width* gets `450` and *height* becomes `600`. Once we've set the variables, they can be used in the HTML `img` tag:

```
# Make HTML code (redirected to index.html):
echo "<a href=\"images/${base}.png\">
<img src=\"thumbs/${base}.png\"
width=$width height=$height></a>"
```

Of course, you could also add a caption to the thumbnail, make an HTML page for the full-sized image, and more.

And Now...

We've covered the basics of Netpbm and how to use the programs from both the command line and in a shell script.

All that's left is to browse through descriptions of the more than 200 utilities and find the ones that you need.

POWER TIP: Watching Remote Logs

If you need to monitor log files on a remote host — actually, *any* text files that are growing over time — you might use a log-watching application, either over a network or by logging onto that host first. But a simpler choice is to run the GNU version of *tail* over the network, using *ssh* to make the connection and keep it secure.

For instance, to watch the files *tmp/startx.log* and *procmail/from* underneath your home directory on the remote host *loghost*, you might run this command from your local host:

```
% ssh loghost tail —retry —follow=name \
   tmp/startx.log procmail/from
```

GNU *tail* can watch many files at once. It will recover if one of the log files is replaced (for instance, if a cron job like *rotate-and-compress* renames one of the log files and starts a new empty one).

Jerry Peek is a freelance writer and instructor who has used Unix and Linux for more than 20 years. He's happy to hear from readers; see http://www.jpeek.com/contact.html.