

More graphics from the command line

This article discusses commonly used image manipulations using command line tools, mainly from the ImageMagick suite. It expands on the examples from the "Command line imaging tools" article.

Last year I wrote an article for IBM DeveloperWorks about image manipulations on the command line using ImageMagick. The article was quite well received, and since then I have fielded many email questions on ImageMagick. This article expands on the techniques discussed in that previous article, as well as answering as many of those questions as I can. If this is the first ImageMagick article from IBM DeveloperWorks that you've found, you would do well to have a look at this first article as well.

This article takes the form of discussing specific problems as examples, but the concepts should be applicable to other problem spaces as well. This is the same approach as taken in the previous article. The techniques discussed here also work in combination with those we've discussed previously.

It should be noted that there are many ways to do the things discussed in this article. I will only discuss the methods I use, and know work for me. That doesn't mean the other tools out there are broken, it just means that I'm happy with what I am using now.

1. Curving corners

If you have a look at MacOS X, and many websites, the pictures have quite nice curved corners. How do you achieve this effect with ImageMagick? Well, we're going to have to show some ingenuity in producing this effect by using the **composite** command.

Before we get there though, let's talk about the strategy we're going to employ. If you think about it, an image with curved corners can be made by taking some standard pre-made corners, and superimposing them over the original image. There's no real need for the corners to be curved even -- we could have angled corners, or something much more fancy.

So, the first step is to make some corners. Below is a corner I whipped up in the GIMP. I won't show you all four, as the others are just this one rotated by differing multiples of 90 degrees.

The GIMP

The GIMP, the GNU Image Manipulation Package, is a very useful raster graphics editor, much like Adobe Photoshop. It's great for tweaking images, or for creating your own new pictures. Checkout www.gimp.org for more details.



You should note that the curve on this image is actually in white, and the rest of the image is transparent. This transparency will allow the image we are adding the corners to to show through. This can be a little confusing, as some image viewers such as **xview** will show the transparency in black or some other color.



The actual corner image will become more clear when we superimpose it upon an image, so let's get on with that. I have a thumbnail which I made earlier of the view from the shore of one of Canberra's lakes. Without the rounded corners, the thumbnail looks like this:



To superimpose an image onto another, you use the **composite** command. Let's just do one corner, to see what happens...

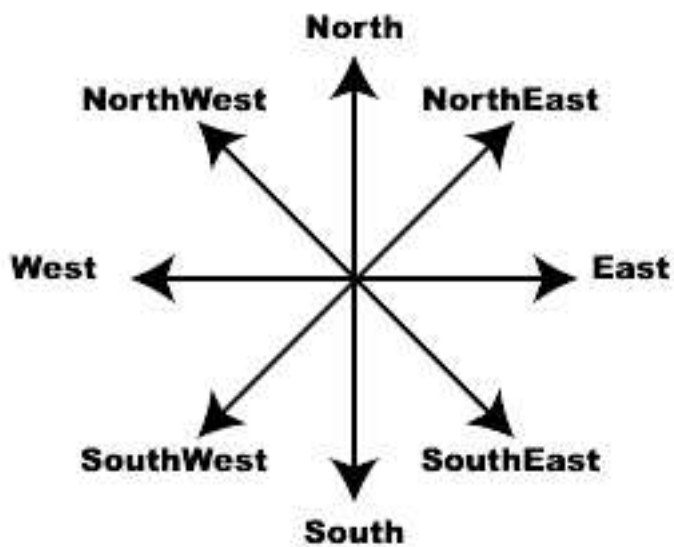
```
composite -gravity NorthEast rounded-ne.png lake.png lake-1.png
```

Here, the gravity argument defines where on the image to put the superimposed image -- in our case the rounded corner. This particular command gives us the following image:



Gravity

The gravity argument specifies where on the background image the superimposed image is placed. The possible gravities, and their effects are:



For example, SouthWest will force the superimposed image to the bottom left hand corner of the background image.

So let's do the rest of the corners...

```
composite -gravity NorthEast rounded-ne.png lake.png lake-1.png
composite -gravity NorthWest rounded-nw.png lake-1.png lake-2.png
composite -gravity SouthEast rounded-se.png lake-2.png lake-3.png
composite -gravity SouthWest rounded-sw.png lake-3.png lake-4.png
```

Which gives us the finished image:



Which looks kinda cool in my humble opinion. You should also take note that there is no reason for these corner images to be rounded. If you're interested in angled corners or such, then they're equally possible -- just change the corner images in a bitmap editor.

If you want to use my rounded corners, a URL is listed in the resources section at the end of this article.

2. Putting frames around images

Another thing which several readers asked about was how to add frames to images. Again, this is relatively easy to do with ImageMagick.

2.1. A raised or lowered border

The first type of frame I will show you is a raised or lowered border. This effect works by tweaking the colors at the edge of an image to give the impression that it is either raised above the surrounding surface, or pushed below it. For the effect, you need to specify a size, with the horizontal size first, and then the vertical size. These sizes must obey the rule that twice the size specified must be less than or equal to the dimension of the image in that direction. For example, you can't specify a frame size vertically that is more than half the vertical size of the image.

To create a raised border, use the *-raise* command line argument. For example, to create a 5 pixel by 5 pixel border, we execute:

```
convert -raise 5x5 tree.png tree-raised.png
```

Which gives us the finished image:



To create a lowered border, just use the *+raise* command line argument instead. For example:

```
convert +raise 5x5 tree.png tree-lowered.png
```

Which gives a slightly different finished image:



2.2. A simple colored border

If you're after something a little more simple, you might be interested in a border of just a solid color. ImageMagick can do this for you as well.

```
convert -bordercolor red -border 5x5 flower.png flower-border.png
```

Which creates:



What border colors can we specify on the command line? Well, the list is simply too long to put into this article. To get a copy of the list, execute this command:

```
convert -list color
```

Here's a list of some of the more commonly used colors of the 683 available: aquamarine, azure, beige, bisque, black, blue, brown, burlywood, chartreuse, chocolate, coral, cornsilk, cyan, firebrick, gainsboro, gold, goldenrod, green, honeydew, ivory, khaki, lavender, linen, magenta, maroon, moccasin, navy, orange, orchid, peru, pink, plum, purple, red, salmon, seashell, sienna, snow, tan, thistle, tomato, turquoise, violet, wheat and yellow.

You can also of course specify your own colors by using any of the following formats, where R represents the red value, G the green, B the blue, and A the alpha (transparency) value:

- #RGB - (R,G,B are hex numbers, 4 bits each)
- #RRGGBB - (8 bits each)
- #RRRGGBBB - (12 bits each)
- #RRRRGGGBBBB - (16 bits each)
- #RGBA - (4 bits each)
- #RRGGBBAA - (8 bits each)
- #RRRGGBBBAAA - (12 bits each)
- #RRRRGGGBBBBAAA - (16 bits each)
- rgb(r,g,b) - (r,g,b are decimal numbers)
- rgba(r,g,b,a) - (r,g,b,a are decimal numbers)

2.3. Building a more complicated frame

Next let's build a slightly more complicated frame, using the *-frame* command line argument. First we'll add a simple frame which is identical (except for the color) to the border we built in the previous example.

```
convert -mattecolor black -frame 5x5 beach.png beach-frame.png
```

The arguments are *-mattecolor* and *-frame* instead of *-bordercolor* and *-border*, but the rest is the same as with the border command.



Now we can add some extra complexity by adding some gray shading similar to what the *-raise* command gave us.

```
convert -mattecolor black -frame 5x5+2 beach.png beach-frame2.png
```

Which is getting there:



Finally, we can add some more decoration, to get the final effect I want...

```
convert -mattecolor black -frame 5x5+2+2 beach.png beach-frame3.png
```

Which finally gives us:



If you're looking at ways to make nice frames for your images, then I recommend that you spend a few moments playing with the arguments to the *-frame* command. For example, here's some interesting frames for a picture of a rock at King's Canyon, in Australia.

```
convert -mattecolor gray -frame 25x25+0+25 rock.png rock-frame1.png
```




```
convert -mattecolor gray -frame 25x25+25+0 rock.png rock-frame2.png
```



3. Processing many images at once

In my previous article, I showed you sample code to apply conversions to many images at once. As has been pointed out by several people, the code I showed was not the best way of doing this.

Here's the code I showed you:

```
for img in `ls *.jpg`  
do  
    convert -sample 25%x25% $img thumb-$img  
done
```

Now it turns out that this is poor bash style, as it doesn't handle spaces in filenames very gracefully (each word will be treated as a separate filename). Instead, a better way of doing this in bash is to do:

```
for img in *.jpg  
do  
    convert -sample 25%x25% $img thumb-$img  
done
```

Which which will handle spaces in filenames much more gracefully.

It turns out however that both of these solutions aren't needed with ImageMagick -- we can just use the **mogrify** command. **mogrify** is used to convert a sequence of images (although it will work for single images as well).

That code snippet above becomes:

```
mogrify -sample 25%x25% *.jpg
```

*Note that this will overwrite the original images with new ones. This is one of the limitations of **mogrify**, in that it is harder to specify output filenames.* The only way to specify an output filename is to change the format of the output image compared with the input image. This will result in a different extension for the new image. For example:

```
mogrify -format png -sample 25%x25% *.jpg
```

This will create a series of output files with the jpg at the end of the filename has been replaced with a png, with the associated image format change.

All of the conversions previously defined will also work with the **mogrify** command, so if you don't mind the original images being overwritten then it's a good choice.