The Definitive Guide to ImageMagick

Michael Still

Apress®

The Definitive Guide to ImageMagick

ISBN (Hardback): 1-59059-590-4

Library of Congress Cataloging-in-Publication data is available upon request.

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Lead Editor: Matt Wade
Technical Reviewer: Doug Jackson
Editorial Board: Steve Anglin, Dan Appleman, Ewan Buckingham, Gary Cornell, Tony Davis, Jason Gilmore, Jonathan Hassell, Chris Mills, Dominic Shakeshaft, Jim Sumser
Project Manager: Kylie Johnston
Copy Edit Manager: Nicole LeClerc
Copy Editor: Kim Wimpsett
Assistant Production Director: Kari Brooks-Copony
Production Editor: Linda Marousek
Compositor and Artist: Kinetic Publishing Services, LLC
Proofreader: Kim Burton
Indexer: Carol Burbo
Cover Designer: Kurt Krames
Manufacturing Director: Tom Debolski

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, e-mail `orders-ny@springer-sbm.com`, or visit `http://www.springeronline.com`.

For information on translations, please contact Apress directly at 2560 Ninth Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax 510-549-5939, e-mail `info@apress.com`, or visit `http://www.apress.com`.

The source code for this book is available to readers at `http://www.apress.com` in the Source Code section and also at `http://www.stillhq.com`.

CHAPTER 4

Using Other ImageMagick Tools

Most of this book discusses the `convert` command, but several other interesting tools ship with ImageMagick. I'll introduce these tools in this chapter and show how to use them as relevant throughout this book; for example, Chapter 3 showed how to use the `mogrify` command to update the metadata in image files. ImageMagick has two classes of tools: the command-line tools and the tools with graphical user interfaces. I'll discuss these two types of tools in separate sections of this chapter.

Using the Command-Line Tools

I'll discuss the command-line tools that ship with ImageMagick in alphabetical order in the following sections so you can use this chapter as a reference guide.

compare

The `compare` command creates a graphical representation of the differences between two images, and it gives you a mathematical measurement of the difference between the two images. In this section, I'll show the effects of the command on a picture I took during a recent hot-air-balloon ride. Figure 4-1 shows the input image.

Figure 4-1. *The original image*

Chapter 5 discusses the `spread` command-line argument, but for now I'll show the output of the `spread` command and use it as example input for the `compare` command. Figure 4-2 shows a `spread` operation with an argument of 1.

Figure 4-2. *The original image, spread by a factor of 1*

The `compare` command will show you the differences between these two images graphically. If you run the following command:

```
compare original.jpg changed.jpg compared.jpg
```

then the final image in the command line will contain the differences between the first two images on the command line. For this example, the output looks like Figure 4-3.

Figure 4-3. *The differences between the original image and the image that was spread by a factor of 1*

The effect of the `spread` command-line argument becomes more pronounced as the factor increases. (Again, you can find a full explanation of the `spread` command-line option in Chapter 5.) Briefly, `spread` takes a random pixel within the circle surrounding the current pixel and swaps the two. The argument to the `spread` command is the radius of that circle. Figure 4-4 shows a spread factor of 10, and Figure 4-5 shows the differences.

Figure 4-4. *The original image, spread by a factor of 10*

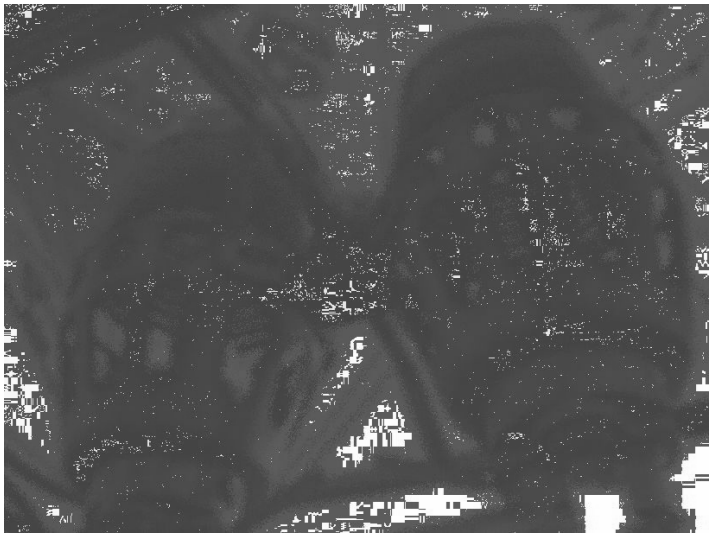

Figure 4-5. *The differences between the original image and the image that was spread by a factor of 10*

The `compare` command has some options as well. To compare only some channels, use the `channel` command-line argument, which lets you select on which channels the comparison occurs. (You can also use the `channel` argument with the `convert` command if you want to apply a conversion to only selected channels.) Additionally, you can compare just a portion of an image using the `extract` command-line argument to `compare`. This command-line argument takes a `geometry` argument, as discussed in Chapter 2.

■ **Note** Chapter 3 explains channels in more detail, but in summary a channel is one of the samples for a given pixel; for example, an RGB image contains a red channel, a green channel, and a blue channel.

You can also select the metric used to measure the difference between pixels with the `metric` command-line argument. Selecting a metric is a highly technical area that is out of scope for this book, so I recommend you refer to the ImageMagick documentation at `http://www.imagemagick.org/script/compare.php` for more information. If the images are large, then the comparison can take quite some time, in which case you can use the `monitor` command-line argument to show the progress of the command. For example, here's a comparison with a `monitor` command-line argument, followed by its output:

```
compare -monitor input1.png input2.png compare.png
```

```
Mogrify image: 100%
Save image: 100%
```

This progress information updates as the command runs.

composite

The `composite` command allows you to overlay one image on another. In the past, the `composite` command was the `combine` command, so if you can't find the `composite` command in your ImageMagick installation, look for `combine` instead. Many of the `convert` command-line options are also implemented by the `composite` command. In this section, I'll focus on the command-line options that are unique to the `composite` command. Specifically, as an example of how to use the `composite` command, I'll show how to annotate the photo in Figure 4-6 with some text.

Let's say you want to put the image shown in Figure 4-7 on top of the one shown in Figure 4-6.

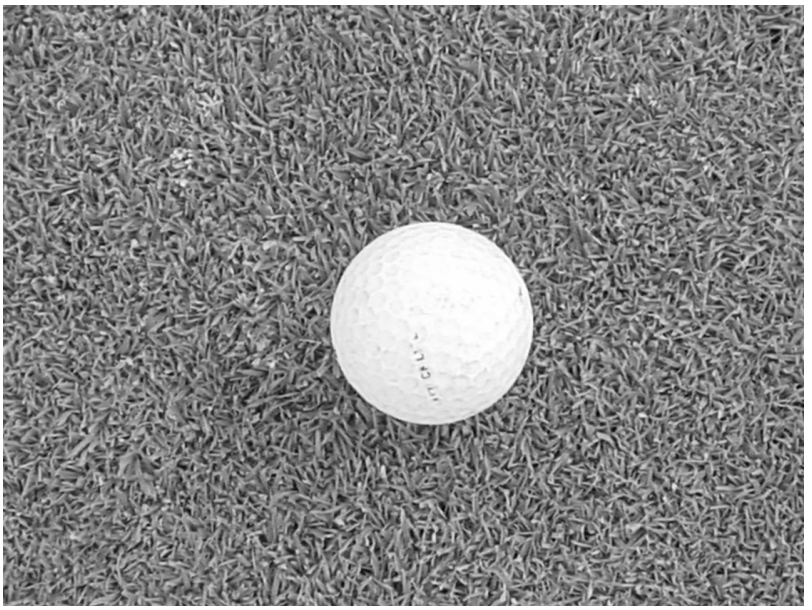

Figure 4-6. *A picture of a golf ball on grass*

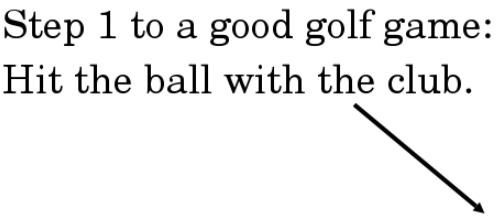

Figure 4-7. *Another picture to overlay on top of the photo in Figure 4-6*

To composite these images, be sure to name the image that will be on top first, because the second image is the one that dictates the size of the output image. This is the command line to use:

```
composite ontop.png underneath.png output.png
```

This produces the output image in Figure 4-8.

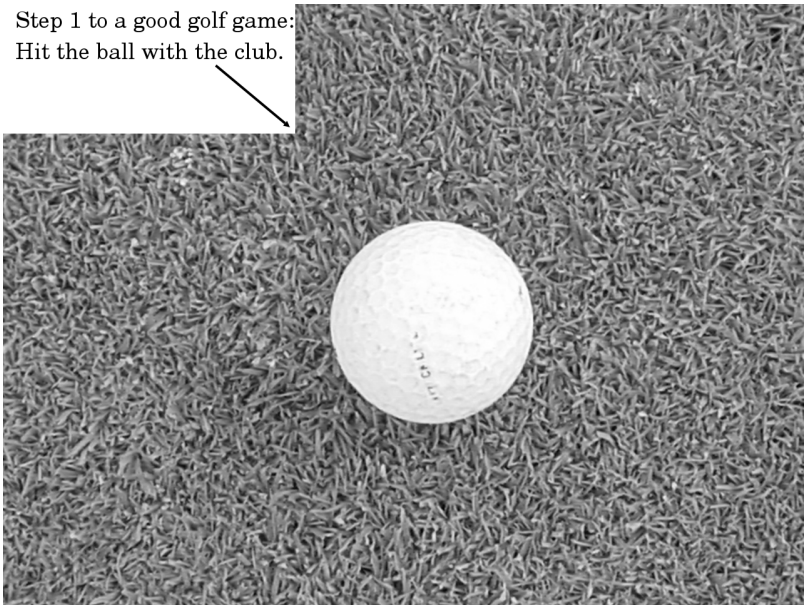

Figure 4-8. *The output image after a composite operation*

This isn't exactly what I intended the output image to look like, however, because I haven't specified an alpha channel in the image to be placed on top. If you add an alpha channel, then the new output image looks like Figure 4-9.

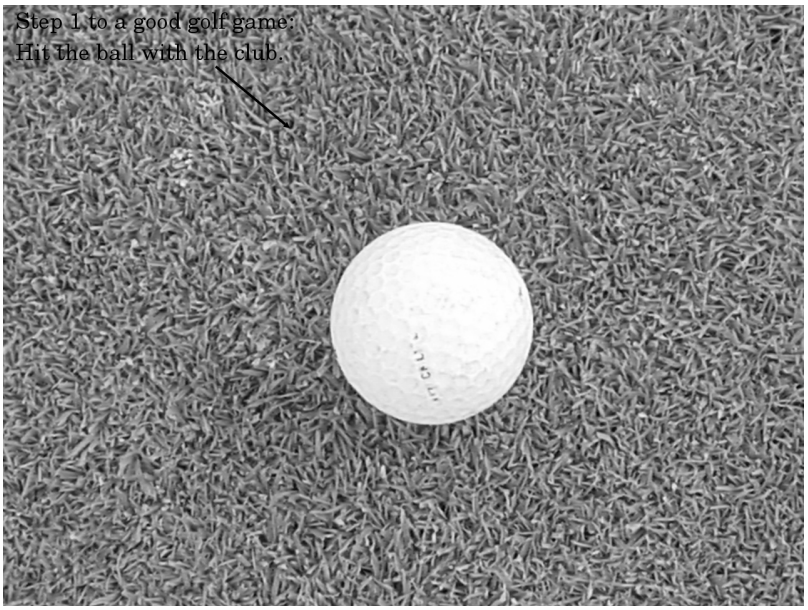

Figure 4-9. *The output image after a composite operation with a top image that has an alpha channel*

You can see from this example that the `composite` command will respect the alpha channel if one is defined for the image on top. (Chapter 7 discusses alpha channels in more detail.)

You can harness the `composite` command and transparency to add some really nice effects. For example, I get quite a few requests for details on how to create images with rounded corners, which is a style that has been made popular by Apple's Mac OS X. To do this, you need to make a rounded corner, with the transparency set up correctly so that the inside of the corner is transparent. If you want premade corners, you can download mine from `http://www.stillhq.com/extracted/article-imagingtoolsmore/corners/`.

I'll now show how to give the image in Figure 4-10 rounded corners.

Figure 4-10. *A bird photo with square corners*

To do this, you need to use four invocations of the `composite` command:

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

This gives you the finished image in Figure 4-11.

Figure 4-11. *A bird photo with rounded corners*

For more discussion on the `gravity` command-line option used in this example, refer to Chapter 7. You can also ask the `composite` command to dissolve the images into each other, which can produce animations with some nice effects. For instance, let's say you have two photos of flowers, as shown in Figure 4-12 and Figure 4-13.

Figure 4-12. *The first flower*

Figure 4-13. *The second flower*

Say you want to combine these two photos with different dissolve levels. Generally, the command line you use looks like this:

```
composite -dissolve 42% input1.jpg input2.jpg output.jpg
```

This will dissolve `input1.jpg` into `input2.jpg` by 42 percent. Figure 4-14 shows some examples of various dissolve percentages.

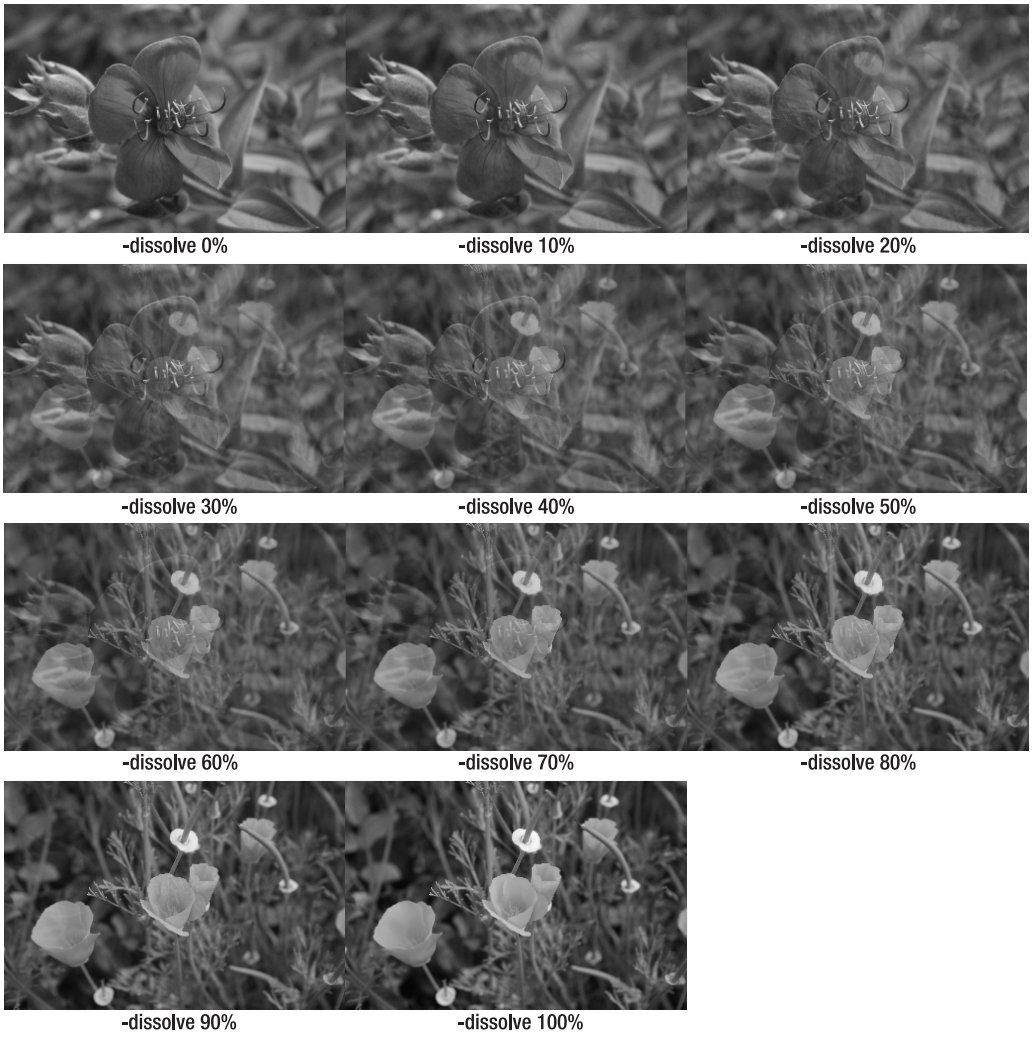

Figure 4-14. *The two images combined with different dissolve levels*

You can also combine images using the `blend` command-line option, which takes a percentage argument much like `dissolve` but produces slightly different results visually. The difference between the two operations is that the `dissolve` command-line option adjusts the transparency of the images before combining them, whilst the `blend` command uses a weighted average of the images to produce its output.

The `composite` command can also produce watermarks on images. For example, I'll now show how to place the watermark shown in Figure 4-15 on the picture of the flower from Figure 4-12.

Figure 4-15. *A watermark to apply to the image*

To do this, use this command line:

```
composite -watermark 30% watermark.png input.jpg output.jpg
```

The main difference between watermarking and dissolving or blending is that the watermark image doesn't need to be the same size as the input image. Figure 4-16 shows the output image that this command line gives you.

Figure 4-16. *The image with a watermark applied*

The problem with this image is that the watermark image doesn't have an alpha channel set; it has a white background instead, which is why it has that lighter rectangle in the image. You can fix this by making the white in the image 100 percent transparent. After that, you get the image shown in Figure 4-17.

Figure 4-17. *The image with a transparent watermark applied*

The `watermark` command-line option takes an argument that specifies the transparency of the watermark. Figure 4-18 shows some of the transparency values and what the results look like.

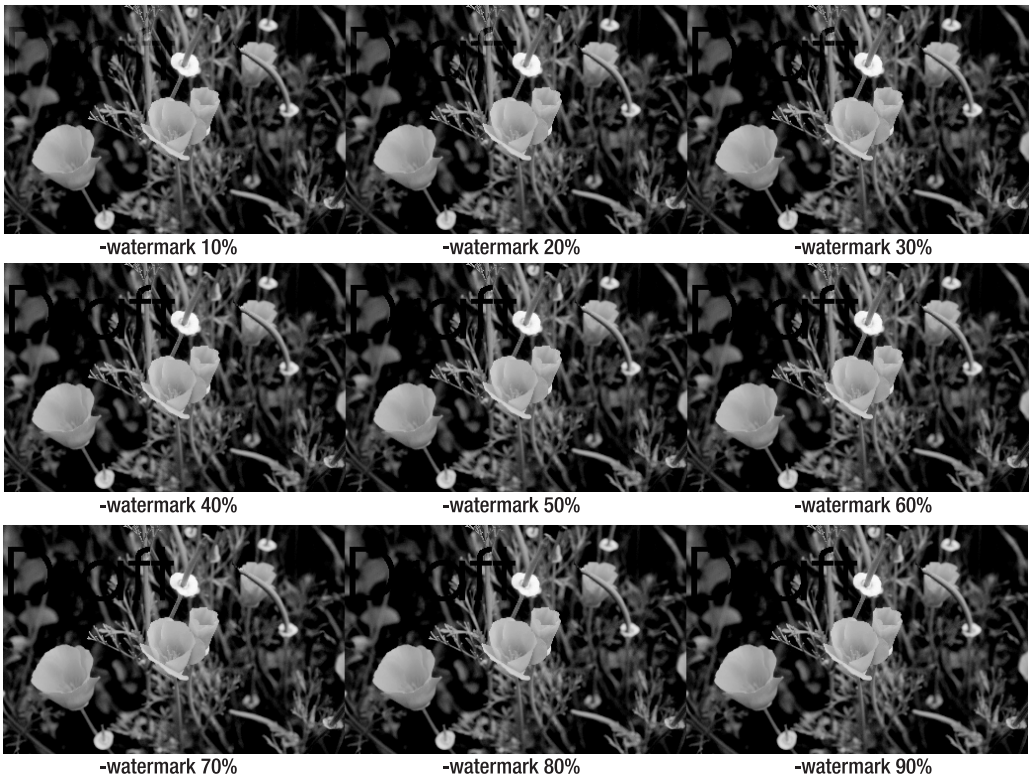

Figure 4-18. *Various transparency options for the* `watermark` *command-line option*

You can choose the placement of the watermark by using the `gravity` command-line option like this:

```
composite -watermark 30% -gravity center watermark.png input.jpg output.jpg
```

This gives you the image shown in Figure 4-19.

Figure 4-19. *A centered watermark*

Finally, for the `composite` command, I'll show you how to use the `stegano` command-line option to hide images inside other images. You can use this to determine whether an image you find on the Internet has been taken from your site (although you can also use it for passing secret messages). To hide the image, first you need an image to hide other images in, and then you need the hidden image. For this example, I'll show how to hide the image shown in Figure 4-20.

Figure 4-20. *An image to hide*

I'll show how to hide this inside the image shown in Figure 4-21.

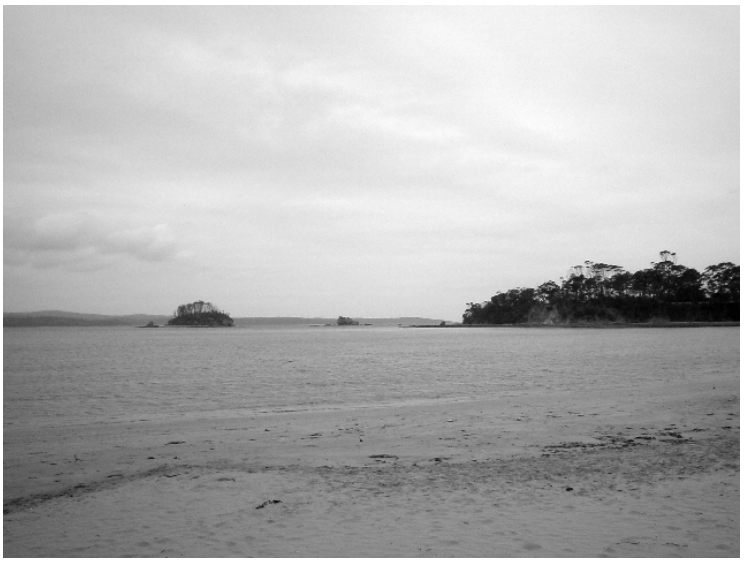

Figure 4-21. *The image in which you'll hide the logo*

To do this, use this command line:

```
composite -stegano 42 logo.png input.jpg output.png
```

The argument to the `stegano` command-line option is the inset inside the input image at which to start hiding the logo. This number needs to be kept secret, or other people will be able to extract the logo. This gives you the image shown in Figure 4-22.

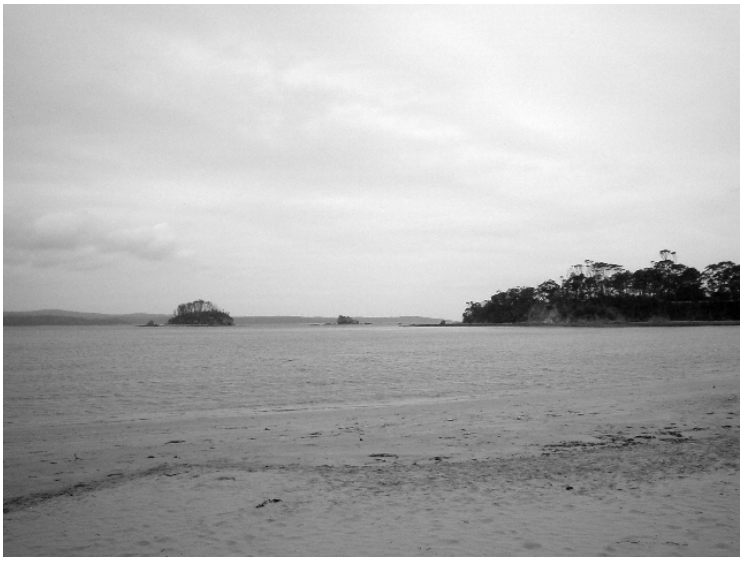

Figure 4-22. *The image with the logo hidden*

To extract the image again, you use the `display` command like this:

```
display -size 204x61+42 stegano:output.png
```

You should note three points about this command line. First, you need to know the size of the logo, which can be determined by using the `identify` command (which is discussed later in this chapter in the "identify" section). Second, you need to know the offset of the image within the other image. It is also important to note that this command will take quite a while to execute. On my 1.7-gigahertz Centrino laptop, this command took 51 seconds to execute. When ready, the `display` command will display the logo image. Finally, the image with the logo hidden inside it is affected by having the logo placed in it, so you need to be willing to accept some image quality loss.

How do you find stolen images using this technique? Well, it's not a good active search technique, because you'd need to perform this slow technique to every image on the Internet. If, however, you stumble across images that you suspect are from your site, then you can use this to prove the point. Unfortunately, the embedded logo might not survive transformations on the stolen image such as resizing, cropping, and so forth.

conjure

ImageMagick implements a scripting language for automating processing called the Magick Scripting Language (MSL). The `conjure` command takes these scripts in Extensible Markup Language (XML) form and executes them. The scripting language used is out the scope of this book, but Chapters 8 through 11 cover lots of other programming options. You can read more about the `conjure` command at `http://www.imagemagick.org/script/conjure.php`.

convert

The `convert` command is the subject of the majority of this book, because it contains most of the ImageMagick functionality. Therefore, I won't document the command any further here, instead referring you to the rest of the book for information. Specifically, see the following chapters: 3, 5, 6, and 7.

identify

Chapter 3 discussed the `identify` command. This command outputs interesting information about the image file (or files) that it's passed. If you're interested in only simple information about the image and want it to be returned efficiently, then you're best off using the `ping` command-line option:

```
identify -ping input.jpg
```

```
input.jpg JPEG 816x612 DirectClass 103kb 0.040u 0:01
```

You can get more information about the image if you use other command-line options. If you specify no command-line options at all, then you get similar output to the `ping` command-line option, as shown here:

```
identify input.jpg
```

```
input.jpg JPEG 816x612 DirectClass 103kb 0.040u 0:01
```

Here you can see that the image is 816×612 pixels and is JPEG compressed. If you want more information about the file, try adding the `verbose` command-line option:

```
identify -verbose input.jpg
```

```
input.jpg JPEG 816x612 DirectClass 103kb 0.030u 0:01
Image: input.jpg
  Format: JPEG (Joint Photographic Experts Group JFIF format)
  Geometry: 816x612
  Class: DirectClass
  Type: TrueColor
  Endianess: Undefined
  Colorspace: RGB
  Channel depth:
    Red: 8-bits
    Green: 8-bits
    Blue: 8-bits
  Channel statistics:
    Red:
      Min: 95 (0.372549)
      Max: 255 (1)
      Mean: 226.582 (0.888556)
      Standard deviation: 12.6188 (0.0494854)
    Green:
      Min: 0 (0)
      Max: 255 (1)
      Mean: 19.4916 (0.0764375)
      Standard deviation: 45.3697 (0.177921)
    Blue:
      Min: 0 (0)
      Max: 255 (1)
      Mean: 42.3754 (0.166178)
      Standard deviation: 40.5556 (0.159042)
  Colors: 31265
  Rendering-intent: Undefined
  Resolution: 72x72
  Units: PixelsPerInch
  Filesize: 103kb
  Interlace: None
  Background Color: white
  Border Color: #DFDFDF
  Matte Color: grey74
  Dispose: Undefined
  Iterations: 0
  Compression: JPEG
```

```
  Quality: 80
  Orientation: Undefined
  JPEG-Colorspace: 2
  JPEG-Sampling-factors: 2x1,1x1,1x1
  Signature: 4a957426ccdc4f819c591f0f020920bef1b1ec4739fd17084bab38c617ccf83a
  Profile-exif: 6140 bytes
...dump of those 6,140 bytes omitted for clarity...
    Image Description:                                .
    Make: SONY.
    Model: CYBERSHOT U.
    Orientation: 1
    X Resolution: 72/1
    Y Resolution: 72/1
    Resolution Unit: 2
    Date Time: 2005:05:13 07:27:09.
    Y Cb Cr Positioning: 2
    Exif Offset: 220
    Exposure Time: 10/1600
    F Number: 40/10
    Exposure Program: 2
    ISO Speed Ratings: 100
    Exif Version: 0220
    Date Time Original: 2005:05:13 07:27:09.
    Date Time Digitized: 2005:05:13 07:27:09.
    Components Configuration: ....
    Compressed Bits Per Pixel: 2/1
    Exposure Bias Value: 0/10
    Max Aperture Value: 48/16
    Metering Mode: 2
    Light Source: 0
    Flash: 0
    Focal Length: 50/10
    Maker Note:
    Flash Pix Version: 0100
    Color Space: 1
    Exif Image Width: 1632
    Exif Image Length: 1224
    Interoperability Offset: 638
    Interoperability Index: R98.
    Interoperability Version: 0100
    File Source: .
    Scene Type: .
    Custom Rendered: 0
    Exposure Mode: 0
    White Balance: 0
    Scene Capture Type: 0
  Tainted: False
  Version: ImageMagick 6.2.3 06/09/05 Q16 http://www.imagemagick.org
```

You can see that this produces a lot of information about the image, including all the information embedded in the JPEG file's EXIF metadata tags (a metadata format specific to the JPEG file format). In return for seeing all this information, the command can take quite some time to complete. The `identify` command can also take a `format` argument, which gives you a chance to output only the information you want. The format of the `format` argument is identical to the format used for the `comment` command-line argument, as discussed in Chapter 3.

You can use this information in some interesting ways; for an example, check out the programming example in Chapter 8.

import

ImageMagick also ships with a screen-capture program called `import`. When you run the command, the cursor in X Windows becomes a set of crosshairs, and when you click a window, the contents of that window are saved into the specified file. For example, the following command line:

```
import capture.png
```

saves the image shown in Figure 4-23 when you click a window.

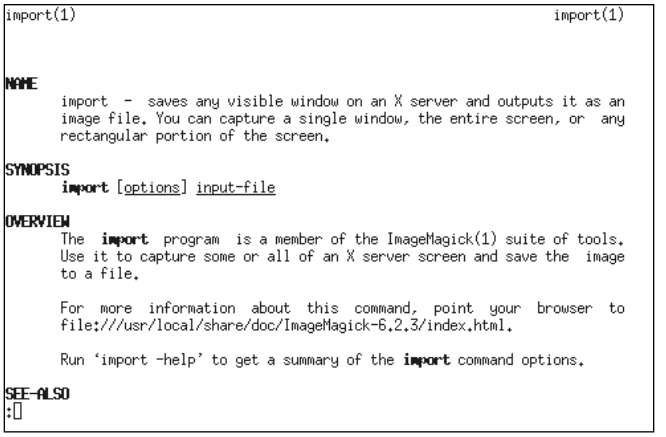

Figure 4-23. *A captured window*

To import the frame from the window as well, use the `frame` command-line option:

```
import -frame capture.png
```

You can import more than one image at a time by specifying more than output filename on the command line:

```
import output1.png output2.png
```

If you need some delay between the image captures, you can use the `pause` command-line option, which takes the number of seconds to delay before each capture (including the first):

```
import -pause 10 output1.png output2.png
```

This produces the two captures shown in Figure 4-24 and Figure 4-25, with a total capture time of 20 seconds.

Figure 4-24. *The first captured window from the multiple import*

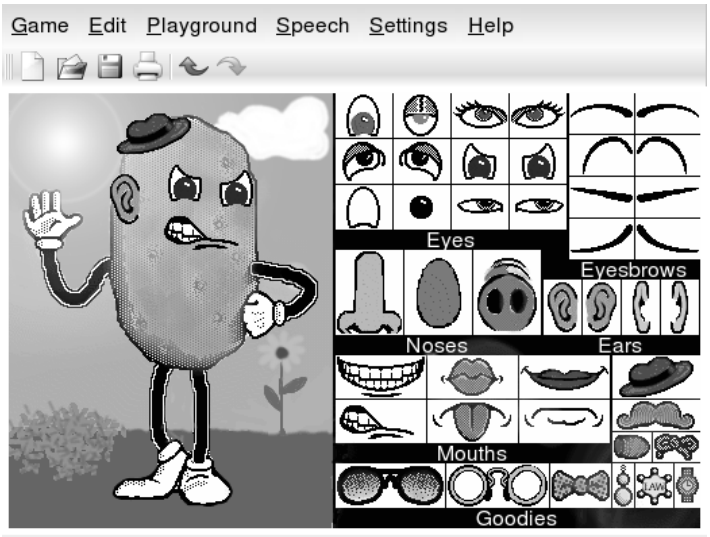

Figure 4-25. *The second captured window from the multiple import*

The `import` command needs an X Windows server, so unless you have one enabled on your Microsoft Windows machine, then this command won't work.

mogrify

Chapter 2 briefly introduced the `mogrify` command, which takes the same arguments as the `convert` command; however, the difference is that the `mogrify` command takes only one image file as an argument. Instead of saving the changed image into the second named file like the `convert` command does, the `mogrify` command will change the original image. This makes the `mogrify` command much more convenient for processing many images at once, because you can specify all of them on the command line, like so:

```
mogrify -implode 4 *.jpg
```

This example applies the `implode` effect (discussed in Chapter 5) to all the images in the current directory whose filenames end with `.jpg`. To achieve the same effect with the `convert` command, you need to implement a simple shell script:

```
for item in *.jpg
do
  convert -implode 4 $item /tmp/$item
  mv /tmp/$item $item
done
```

The `convert` command requires a lot more work and will be less reliable. What if you wanted to change the image format of a bunch of images? The `format` command-line option allows you to do this. For example, to turn a bunch of JPEG files into PNGs, you can use this command line:

```
mogrify -format png *.jpg
```

This will convert all the files in the current directory ending with `.jpg` into PNG files by changing the extension in the filename. This can also be handy if you want to keep the original images but want a shorthand way to apply a transformation to all those images. If you use this option, bear in mind that there can be issues associated with converting images to another format, as discussed in Chapter 3.

montage

The `montage` command creates an image from a sequence of images. For example, to create an image that contains thumbnails of the images it was passed, use the following command line:

```
montage *.jpg output.png
```

Figure 4-26 shows an example of the output.

Figure 4-26. *Output from the* `montage` *command*

The `montage` command can take some arguments of its own as well. For example, to label the images in the montage, use the `label` command-line argument. To label the images with the filename, use the following command line:

```
montage -label %f *.jpg output.png
```

This displays the output shown in Figure 4-27.

Figure 4-27. *Output from the* `montage` *command, with the* `label` *option*

You can find more documentation on the substitution strings to use in the `label` string in Chapter 3.

To put a frame around the images, you can add the `frame` command-line option. Here's a frame with a size of 5:

```
montage -label %f -frame 5 *.jpg output.png
```

This gives you the output shown in Figure 4-28.

Figure 4-28. *Output from the* `montage` *command, with the* `label` *option and a* `frame` *option of 5*

In contrast to that, Figure 4-29 shows a frame with a size of 1.

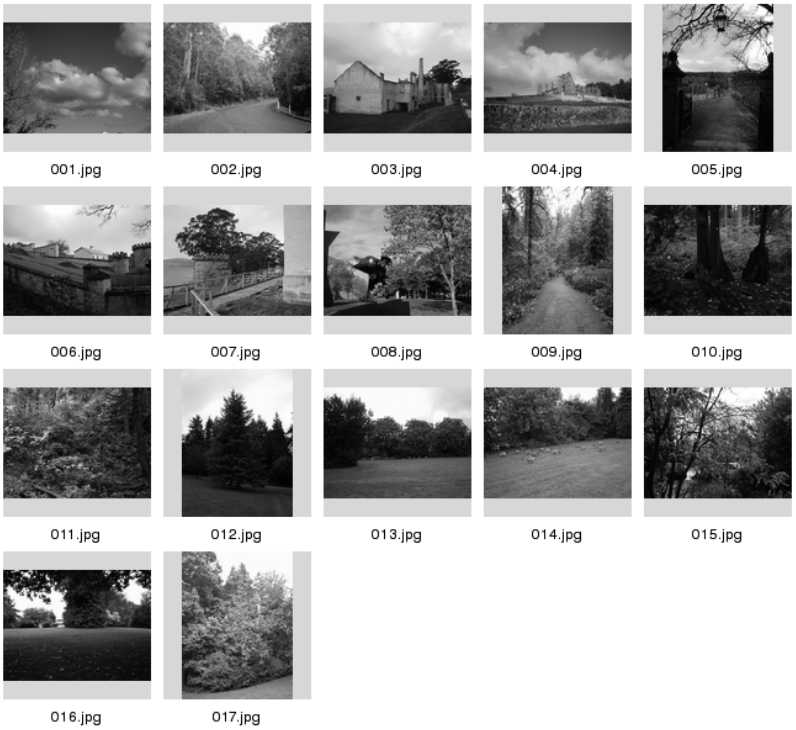

Figure 4-29. *Output from the* `montage` *command, with the* `label` *option and a* `frame` *option of 1*

Apart from frames, you can also create shadows around the images by using the `shadow` command-line option. Here's an example:

```
montage -label %f -shadow *.jpg output.png
```

ImageMagick creates the image shown in Figure 4-30.

Figure 4-30. *Output of the* `montage` *command, with a shadow*

You can combine the shadow effect with a frame, like this:

```
montage -label %f -shadow -frame 5*.jpg output.png
```

You get the effect shown in Figure 4-31.

Figure 4-31. *Output of the* `montage` *command, with a shadow and a frame*

Many of the color and stroke options discussed in Chapter 7 affect the way frames are drawn. For example, you can use the border color, background color, fill color, and stroke color to change features such as the background color of the image and the way the frame is painted. Experiment with these options if you'd like to further customize the way frames in montages look. The `montage` command can even create HTML image maps; however, since image maps are a bit dated as a concept now, I won't cover them in this chapter.

You can also manipulate the space that each thumbnail consumes. For instance, to enforce a 100×200 space for each thumbnail image, you can use this command line:

```
montage -label %f -frame 5 -geometry 100x200 *.jpg output.png
```

This produces the output shown in Figure 4-32.

Figure 4-32. *Output from the* `montage` *command, with the* `label` *option, a* `frame` *option of 1, and a given frame size enforced*

You can find more information about the `geometry` command-line option in Chapter 2. I'll briefly show you how to add space around images, though. You do this with a geometry that consists of a plus sign and the horizontal spacing and a plus sign and the vertical spacing. For example, to put 10 pixels of horizontal spacing and 5 pixels of vertical spacing around an image, use a command line like this:

```
montage -label %f -frame 5 -geometry +10+5 *.jpg output.png
```

This gives you the image shown in Figure 4-33.

Figure 4-33. *Output from the* `montage` *command, with the* `label` *option, a* `frame` *option of 1, and spacing enforced*

You can create gaps in the montage by using the special `null` filename:

```
montage 001.jpg null: 002.jpg output.png
```

This gives you a montage with a blank space in the middle, as shown in Figure 4-34.

Figure 4-34. *A montage with a gap*

Furthermore, you can label some images and not others by using the `label` command-line option to specify a label and then use the `+label` command-line option to turn labels off:

```
montage -label "An image" 001.jpg +label null: -label "Another image" ➥
002.jpg output.png
```

This gives you an image with a gap in the middle and no label for the gap, as shown in Figure 4-35.

Figure 4-35. *An example of not labeling all images*

Finally, you can tell the `montage` command how many images to have per row in the output. You do this with the `tile` command-line option, which lets you specify the maximum number of thumbnails to appear in a row. For example, to enforce having three thumbnails in each row, you can use the following command line (which also uses some of the examples described previously):

```
montage -label %f -frame 5 -tile 3 *.jpg output.png
```

This produces the output shown in Figure 4-36.

Figure 4-36. *Output from the* `montage` *command, with the* `label` *option, a* `frame` *option of 1, and a maximum of three thumbnails per row*

If you want to specify the number of rows, then you can specify a grid like this:

```
montage -label %f -frame 5 -tile 5x5 *.jpg output.png
```

This produces the output shown in Figure 4-37.

Figure 4-37. *Output from the montage, having specified a number of output rows*

This example has a blank line at the bottom, because you told ImageMagick you wanted five output rows, even though only four were needed. To tell ImageMagick to work out how many images to put in each row and specify only the number of output rows, then just specify that number with an x at the front:

```
montage -label %f -frame 5 -tile x5 *.jpg output.png
```

ImageMagick works out how many images should go in each of the five rows and creates the montage shown in Figure 4-38.

Figure 4-38. *Output from montage, having specified a number of output rows but not the number of images in each row*

Using the Graphical Tools

Along with the command-line tools you've seen so far, ImageMagick ships with a variety of graphical tools. These graphical tools require an X Windows server to work, which can make them harder to use on Microsoft Windows machines. If you're running Windows, then I recommend looking into installing an X Windows server if these tools sound interesting.

animate

The `animate` command displays either an animated file, such as an animated GIF file, or a sequence of image files as an animation in a window. If you're using Microsoft Windows,

then you need to have an X Windows server installed for this command to work. Figure 4-39 shows an example of `animate` in action; it displays a set of images from Chapter 5 of this book.

Figure 4-39. *The* `animate` *command running*

The `animate` command also has a graphical menu that will appear if you click the image. Figure 4-40 shows what the menu looks like.

Figure 4-40. *The* `animate` *command's menu*

Click any of these buttons to be prompted for what you'd like `animate` to do. Options include controlling what animation you're viewing, setting the direction of the animation (forward or backward), setting the speed of the animation, and getting information on the images being viewed. You can also access help from within the application. Right-click the image to make the menu disappear. The information about the images is the repackaged output of the `identify` command, which looks like Figure 4-41 when displayed by `animate`.

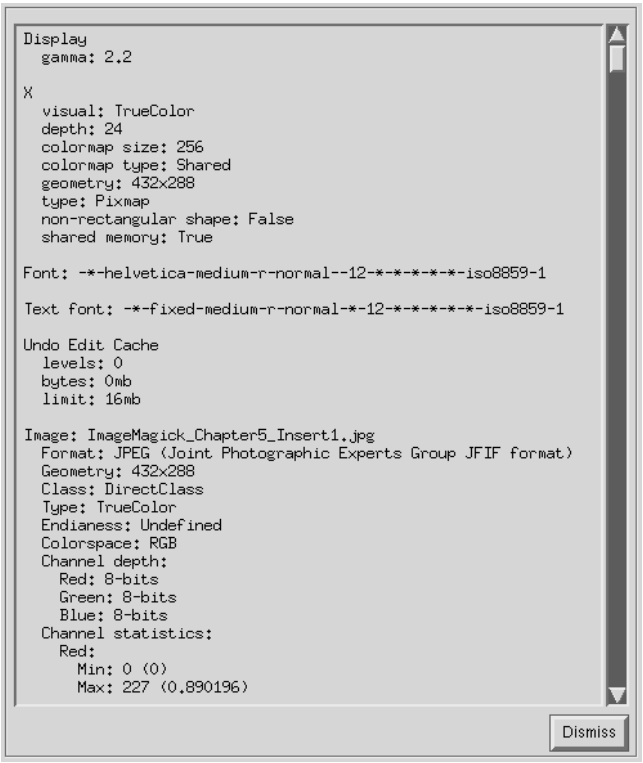

Figure 4-41. *The* `animate` *command displaying information about some images*

display

The `display` command is the other graphical command implemented by ImageMagick, which also requires an X Windows server. If you're using Microsoft Windows, then you'll need to install such an X Windows server before this command will work. When invoked with an image filename, `display` simply displays the image on the screen. You can also include the `backdrop` command-line option to tell `display` to display the image centered on the screen, with the rest of the screen being covered in a neutral backdrop. For example:

```
display -backdrop image.png
```

To specify the color of the backdrop, you can set the background color like this:

```
display -background green -backdrop image.png
```

You can find more information about how to specify colors in Chapter 7.

Furthermore, `display` can automatically watch files when they're being displayed and redisplay them if they change on disk. This can come in handy if you have an automated process that produces an image and you want to keep and eye on it, such as when displaying on your desktop what your Web camera is currently sending to the Internet. You implement this with the `update` command-line option:

```
display -update 5 input.jpg
```

This will tell `display` to check whether the `input.jpg` file has changed on disk every five seconds and redisplay it if it has. You can also change the title of the window that the `display` command uses with the `title` command-line option:

```
display -title "This is a picture of a foo" input.jpg
```

This command-line option also works for the `animate` and `montage` commands. You can include information about the image in this string as well as use the format strings discussed in Chapter 3.

Finally, for the `display` command line, you can tell the command to display the image as the background for a given X window. For this to happen, just specify the window ID to use:

```
display -window root image.png
```

This sets your desktop pattern to the image in `image.png`. You can also specify the individual ID of an X window. To get the ID of a window, use the `xwininfo` command. This command will turn your mouse cursor into a set of crosshairs, and when you click a window, it will dump useful information such as this:

```
xwininfo: Please select the window about which you
          would like information by clicking the
          mouse in that window.

xwininfo: Window id: 0x2600010 ➥
"mikal@challenger: /home/mikal/imagemagickbook/content"

  Absolute upper-left X:  21
  Absolute upper-left Y:  478
  Relative upper-left X:  4
  Relative upper-left Y:  21
  Width: 772
  Height: 511
  Depth: 24
  Visual Class: TrueColor
  Border width: 0
  Class: InputOutput
  Colormap: 0x20 (installed)
  Bit Gravity State: NorthWestGravity
  Window Gravity State: NorthWestGravity
  Backing Store State: NotUseful
  Save Under State: no
  Map State: IsViewable
  Override Redirect State: no
  Corners:  +21+478  -607+478  -607-61  +21-61
  -geometry 128x39+17-57
```

This is similar to how the `import` command works.

When you have an image being displayed that isn't the background of an X window, if you click the image, a menu appears with a number of options. Figure 4-42 shows this menu.

Figure 4-42. *The display menu*

The File menu lets you access the functions you'd expect from such a menu, such as opening new files, moving the next file in a list provided on the command line, moving to a previous file listed on the command line, saving your changes, printing, and so forth.

The Edit menu supplies undo and redo facilities, as well as cut and paste. The View menu lets you change the size at which the image is displayed, apply changes, and refresh the image.

The Transform, Enhance, F/X, Image Edit, and Miscellany menus let you perform many of the transformations covered in this book. These menus provide you with an interactive method of determining which transformations to perform in which order to get the result you desire. You can also apply these preview transformations with the `preview` command-line option. See the ImageMagick documentation at `http://www.imagemagick.org/script/display.php`.

The Help menu provides online help. Right-click the image for a list of shortcuts.

Conclusion

In this chapter, you looked at the various command-line and graphical tools that ImageMagick provides. The `convert` command is the focus of most of this book, so refer to other chapters for more information about it. Specifically, see Chapter 2, Chapter 3, Chapter 5, and Chapter 6.

ImageMagick can perform a lot of useful functions with the commands it provides, including creating montages of images with a large variety of formatting options, displaying your images for you, capturing windows from your X Windows session, and displaying images as the background in those windows on your X session. ImageMagick can perform many functions that aren't covered by the `convert` command, and this chapter has shown you a few of them. I could show you other details about these commands, but it would take hundreds of pages to do so. I recommend you refer to the ImageMagick documentation at `http://www.imagemagick.org` if you need more specific details of a command that you've seen in this chapter.

The next chapter shows you some of the more artistic transformations that ImageMagick can perform, and Chapter 6 shows you the remainder of the image transformations ImageMagick offers. Chapter 7 covers how to draw on existing images and create new images with ImageMagick; then I'll move introduce four examples of programmer interfaces to ImageMagick, so read on for more details.