ImageMagick 命令行使用方法

孤高天使 <yeshengzou@gmail.com>

2010 年 1 月 29 日

http://www.zys-free.com

标签: imagemagick

本文是对官方网站上 Command Line Processing 这篇文档的翻译, 它在:

http://www.imagemagick.org/script/command-line-processing.php

ImageMagick 的命令行形式有时非常的简单, 像下面这样:

```
1  $ convert image.jpg image.png
```

可有时, 它也非常的复杂:

```
1  $ convert label.gif +matte \
2  \( +clone -shade 110x90 -normalize -negate +clone -compose Plus -
       composite \)
3  \( -clone 0 -shade 110x50 -normalize -channel BG -fx 0 +channel -
       matte \)
4  -delete 0 +swap -compose Multiply -composite button.gif
```

上面第一个命令的作用是把一张 JPEG 格式的图片转为 PNG 格式, 第二个命令可能就比较让人头大了. 它的作用是把一张简单的二维质感的图片, 修辞出一种浮雕效果 (如图1).

先简单说明一下 ImageMagick 的命令行使用格式. 从上面的第二个有些夸张的命令可以看出, 为了让我们写出的东西更容易阅读, 可以使用 \ 来分行. \ 是 Unix 下的命令行分行用字符, 在 Windows 下, 你可以使用 ^ 来分行. 之后的文档中, 我们和上面一样使用 Unix 风格. 当然, Unix 和 Windows 的命令行环境还有其它一些区别.

图 1: 第二个命令的效果

接下来, 看看 ImageMagick 的一个完整命令是如何构成的. 我们希望你在读完这些文字并理解后, 可以使用 ImageMagick 的命令行完成一些复杂点的图像处理工作, 不要总是去求助于它的程序接口 (http://www.imagemagick.org/script/api.pl

当你在使用 ImageMagick 命令行时, 可以看看 ImageMagick 使用示例 (http://www.imagemagick.org/Usage/) , 以获得更多的帮助.

1 命令的结构

ImageMagick 的命令行由下面这些元素构成:

1. 一个, 或多个文件名.
2. 零个, 一个, 或多个图像设置项.
3. 零个, 一个, 或多个图像操作项.
4. 零个, 一个, 或多个图像序列操作项.
5. 零个, 一个, 或多个图像组.
6. 零个, 或一个图像输出名 (`convert`, `composite`, `montage.php`, `compare`, `import`, `conjure`, 这些需要).

下面会详细介绍这些构成元素的细节.

2 输入文件名

ImageMagick 扩展了 “输入文件名” 它原本的含义, 现在它包括了:

- 文件名通配符.
- 一种明确的图片格式.

- 内置图像或图案.
- 标准的输入输出, 文件描述符.
- 一张图片中的某些帧.
- 一张图片中的部分区域.
- 缩放了的内嵌图像.
- 裁切了的内嵌图像.
- 文件名引用.

现在对上面的列表作一些说明.

2.1 文件名通配符

在 Unix shell 环境下, 有一些特殊的字符是作为通配符使用的, 如 * 和 ? . ImageMagick 在各个平台上都支持文件名通配符. 假如你想把某目录下的 `1.jpg`, `2.jpg`, `3,jpg`, `4.jpg` 和 `5.jpg` 这些文件转成一个 GIF 动画, 那么你可以使用这条命令方便地引用所有的 JPEG 文件:

```
1  $ convert *.jpg images.gif
```

2.2 明确的图片格式

图像的数据, 都是以一种确定的格式存储的, 比如常见的 JPEG, PNG, TIFF 等. ImageMagick 在读取, 解析图片之前, 必须要知道图片的格式.

多数图像格式, 在文件中都设有一些标识来表明它属于哪种格式. 如果没有, ImageMagick 会根据文件的扩展名来判断. 如 `image.jpg` 会告诉 ImageMagick 这是一张 JPEG 格式的图片. 某些情况下, ImageMagick 不知道图片的格式, 那么这时就需要手动指定了. 如, 我们有一张名为 `image` , 存储了 RGB 三原色位深原始信息的图片 (未经过任何压缩的位图), ImageMagick 当然无法自己得知它是什么格式的图片, 所以, 这时就需要我们明确指定图片格式.

```
1  $ convert -size 640x480 -depth 8 rgb:image image.png
```

2.3 内置图像或图案

ImageMagick 有许多内置的图像和图案, 要使用 checkerboard 中的图案:

```
1  $ convert -size 640x480 pattern:checkerboard checkerboard.png
```

(PS: 我没搞懂这是什么东西 >_<)

2.4 标准的输入输出, 文件描述符

Unix 和 Windows 都支持通过管道来重定向输入输出. ImageMagick 支持从标准的输入输出流中读写图像数据, 由依次使用一个虚文件名 - 来实现. 下面的例子把 convert 的输出通过管道重定向到了 display .

```
1  $ convert logo: gif:- | display gif:-
```

(logo: 理解成是一个内置的图像, logo 这种特殊类型.)

第二个用于确定图像格式的 gif: 是可选的, 因为 GIF 这种格式有自己的标识, ImageMagick 认识它. convert 同样能以这种方式接受标准输入:

```
1  $ convert rose: gif:- | convert - -resize "200%" bigrose.jpg
```

其它的一些管道, 你可以通过它们的文件描述符来访问. 文件描述符 0, 1, 2 已经被预定义为标准输入, 标准输出, 错误输出. 如果一个管道被指定为文件描述符 N, 那么你可以通过 fd:N 来使用它. 下一个例子展示了如何通过管道重定向, 把文件描述符 3, 4 的数据添加到文件描述符 5 中.

```
1  $ convert fd:3 fd:4 -append fd:5
```

ImageMagick 6.4.9-3 中才添加了对文件描述符的支持.

对于 Python, 你可以通过调用 File 对象的 fileno() 方法来获取文件描述符.

在需要的时候, 你也可以为文件描述符指定具体的图像格式:

```
1  $ convert gif:fd:3 jpg:fd:4 -append tif:fd:5
```

2.5 选取图片的某些帧

某些图片格式可以包括有多个图像帧. 你可以只获取第一帧, 最后一帧, 或中间的某些帧. 为此, 你可以在文件名之后, 以方括号括起来的形式指定帧. 下面的例子中, 对于一个有多帧的 GIF 图片, 我们只取其第一帧.

```
1  $ convert 'images.gif[0]' image.png
```

> 在 Unix shell 的环境下, 一般中括号是会被转义的, 所以, 我们需要使用单引号把文件名引起来. Windows 的命令行环境下不用单引号也可以, 但多写一对单引号并不会有什么问题. 另外, 对于单引号和双引号的作用, 在 Unix 和 Windows 这两个平台上, 常常是相反的, 所以, 如果你使用 Windows , 那么请注意将我们例子中的单引号改为双引号.

你也可以一次获取多帧, 在方括号中标出一个范围即可, 如下面的例子, 我们获取了前四帧的图像:

```
1  $ convert 'images.gif[0-3]' images.mng
```

最后, 你可以一次获取非连接的多帧. 下面的命令以 3,2,4 的顺序获取图像:

```
1  $ convert 'images.gif[3,2,4]' images.mng
```

注意上面的最后两个命令, 输出被写入了一个类型为 MNG 的文件当中. 因为 MNG 支持保存多帧图像, 而如 JPG 之类的格式只是保存单帧的图像. 在下面 “图像输出名” 一节我们还会介绍这方面的内容.

2.6 选取一张图片中的部分区域

最原始的位图图像, 就是一个表示出各像素点颜色的序列, 它的文件中没有任何的其它像宽, 高, 格式标识等附加信息. 对于这类原始的图像数据, 在处理时我们必须明确指定图像的宽和高, 或者给出一个范围. 在下面的例子中, 我们要处理的图片是一个 8 位的 RGB 位图, 宽是 6000, 高为 4000, 而我们只需要获取一块中心附近 600x400 的图像信息.

```
1  $ convert -size 6000x4000 -depth 8 'rgb:image[600x400+1900+2900]'
       image.jpg
```

使用 `-extract` 选项也可以实现相同的功能:

```
1  $ convert -size 6000x4000 -depth 8 -extract 600x400+1900+2900 rgb:
       image image.jpg
```

2.7 缩放内嵌图像

读入一些图片的同时, 重新定义它们的尺寸是很方便的. 假设你有很多的大的 JPEG 图片需要转换成一组 PNG 格式的缩略图:

```
1  $ convert '*.jpg' -resize 120x120 thumbnail%03d.png
```

这里, 所以的图片都会被读入并且分别被重定义大小. 在读入的同时定义尺寸在效率上更好, 同时资源占用也更少:

```
1  $ convert '*.jpg[120x120]' thumbnail%03d.png
```

2.8 裁切内嵌图像

同上.

```
1  $ convert '*.jpg' -crop 120x120+10+5 thumbnail%03d.png
```

```
1  $ convert '*.jpg[120x120+10+5]' thumbnail%03d.png
```

2.9 文件名引用

靠一个另外的文件名, 来表示实际要读取的图像文件名, 实现这个功能有两种方法, 第一种方法是使用 @ 这个符号, 来指定一个文件, 这个文件中包含有实际需要读取的图像文件名, 如一个名为 myimages.txt 的文本文件中有如下内容:

```
1  frame001.jpg
2  frame002.jpg
3  frame003.jpg
```

下面的命令, ImageMagick 就会读入 frame001.jpg, frame002.jpg, frame003.jpg .

```
1  $ convert @myimages.txt mymovie.gif
```

另一种方法, 是在文件名中使用格式化字符串, 然后在后面的方括号中指定一个范围. 看下面的例子:

```
1  $ convert image-%d.jpg[1-5]
```

ImageMagick 实际上会去读取如下的图像:

```
1  image-1.jpg
2  image-2.jpg
3  image-3.jpg
4  image-4.jpg
5  image-5.jpg
```

3 图像设置

命令行中的一个设置项, 可以作用于读入的图像, 图像操作, 或者将会被输出的图像. 一个设置项被设置后会一直发挥作用除非它被重置或命令执行完毕. 图像设置命令包括有:

```
-adjoin -affine -alpha -antialias -authenticate
-background -bias -black -point -compensation
-blue -primary -bordercolor -caption -channel
-comment -compress -debug -define -delay -density
-depth -display -dispose -dither -encoding -endian
-extract -family -fill -filter -font -format
-fuzz -geometry -gravity -green -primary -interlace
-intent -interpolate -label -limit -linewidth -log
-loop -mask -mattecolor -monitor -orient -page
-pointsize -preview -quality -quiet -red -primary
-region -render -repage -sampling -factor -scene
-seed -size -stretch -stroke -strokewidth -style
-texture -tile -transparent-color -treedepth -type
-undercolor -units -verbose -virtual-pixel -weight
```

下面例子中的 `-channel` 会被作用于每一个图像, 就像我们说的那样, 设置项会持续起作用.

```
$ convert -channel RGB wand.png wizard.png images.png
```

4 图像操作

图像操作项与图像设置项不同, 它只作用于紧接着存在的一个图像, 仅仅是这一个图像, 之后, 图像操作项就会失效. 在这里, 我们说明一下. 命令行参数有三种, 前面讲过的图像设置项, 这里的图像操作项, 以及后面会讲到的序列操作项. 图像操作项包括下面这些.

```
-annotate -black -threshold -blur -border -charcoal
-chop -clip -clip-path -clip -mask -colors -colorize
-colorspace -compose -contrast -convolve -crop -cycle
-despeckle -draw -edge -emboss -enhance -equalize
```

```
-evaluate -extent -flip -flop -floodfill -frame
-gamma -gaussian -blur -implode -lat -level -map -mask
-median -modulate -monochrome -negate -noise -normalize
-opaque -ordered-dither -paint -posterize -raise
-profile -radial -blur -raise -random-threshold
-resample -resize -roll -rotate -sample -scale
-sepia -tone -segment -shade -shadow -sharpen -shave
-shear -sigmoidal -contrast -solarize -splice -spread
-strip -swirl -threshold -transparent -thumbnail -tint
-transform -trim -unsharp -version -wave -white-point
-white -threshold
```

下面例子, `-negate` 这项只对 wand 有效, 对 wizard 是无效的 (但是, 如果把它放在 wizard 后的话, 则会作用于这两张图).

```
$ convert wand.png -negate wizard.png images.png
```

5 图像序列操作项

一个图像序列操作项, 只作用于紧接着存在的一个图像序列, 它们是.

```
-append -average -clut -coalesce -combine -composite
-crop -deconstruct -delete -flatten -fx -identify
-insert -map -morph -mosaic -process -reverse
-separate -swap -write
```

6 图像定位

许多命令行选项都有一个 `geometry` 参数, 用于指定图像的宽, 高等信息. 因为图像的坐标系, 尺寸, 位置等信息是我们经常会用到的, 所以为了方便, `geometry` 这个参数可以用不同的格式给出. 关于这点, 接下来我们会详细地介绍.

一些命令行选项可接受如下多种格式的 `geometry` 参数. 请记住, 它们处理具体参数时的效果是不同的, 详细的内容可查阅它们各个的说明文档.

```
-adaptive-resize -border -borderwidth -chop -crop
-density -extent -extract -frame -geometry
-iconGeometry -liquid-rescale -page -region -repage
-resize -sample -scale -shave -splice -thumbnail -window
```

geometry 参数可以使用下表列出的多种格式指定, 在表后我们会讨论更多的细节. 最常用的一种格式是 size[offset] , 意为 size 是必须给出的, 而 offset 则是可选的. 不过, 有时 [size]offset 也行. 同时要注意, geometry 这个参数中, 绝不允许出现空格符.

size 的形式	大概的说明 (实际效果对于不同的选项可能差别较大)
scale%	宽和高同时根据指定的百分比缩放
scale-x%scale-y%	宽和高根据指定的百分比分别缩放
width	指定宽度, 高度根据原尺寸比例自动确定
width x height	指定最大的宽度和高度, 图像宽高原比例保存不变 (注意没空格, 下同)
width x height^	指定最小的宽度和高度, 宽高原比例不变
width x height!	指定宽度和高度, 忽略原始比例
width x height>	和 width x height 一样, 但只对比这个尺寸大的图像有作用
width x height>	和 width x height 一样, 但只对比这个尺寸小的图像有作用
area@	等比例缩放图像, 使其总像素值不大于 area. (对于一张 1:1 的图, 若设置为 121@, 则结果正好是 11x11.)

注意, 上面结尾的修辞字符是可以组合使用的.

{size}{offset}	指定 offset 偏移 (默认是 +0+0). {size} 的格式见前一个表格.
{+-}x{+-}y	指定水平和垂直的偏移, 以像素为单位, 两者必须同时指定 偏移量对 -gravity 设置项有效, 对其它的带 % 或另外的 size 操作无效.

7 对宽和高的基本设置及操作符 (%^!)

下面展示了一些简单的例子, 用以说明如何给出 -resize 的 geometry 参数. 我们将使用内置的 logo: 这张图作为我们的 “输入图像”. logo: 这张原始图片宽为 640 像素, 高 480 像素, 记为 640x480. 就像你看到的, 宽总在高的前面. 这条规则同样适用于我们可能会讲到的 “坐标” 或 “偏移”.

```
$ convert logo: -resize '200%' bigWiz.png
```

```
$ convert logo: -resize '200x50%' longShortWiz.png
```

```
1  $ convert logo: -resize '100x200' notThinWiz.png
```

```
1  $ convert logo: -resize '100x200^' biggerNotThinWiz.png
```

```
1  $ convert logo: -resize '100x200!' dochThinWiz.png
```

(叙述略, 请自行观察结果)

8 忽略宽或高的表示方法 (<>@)

看一些例子:

```
1  $ convert logo: -resize '100' wiz1.png
```

```
1  $ convert logo: -resize 'x200' wiz2.png
```

```
1  $ convert logo: -resize '100x200>' wiz3.png
```

```
1  $ convert logo: -resize '100x200<' wiz4.png
```

(叙述略, 请自行观察结果)

(@ 的作用前面已经提到过了)

请注意引号的使用. 上面的例子, 包括后面的例子中, 我们都使用引号把 geometry 引起来了的. 很多时候, 这样做不是必须的, 但当你使用了 < 和 > 的时候, 就一定要使用引号, 否则这两个符号会被当成命令行的重定向操作处理. 另外, 在 Windows 平台上, ^ 也必须使用引号. 所以, 为了安全起见, 我们最好养成对于 geometry 总是使用引号的习惯.

9 图像定位中的偏移

我们通过一些例子来说明 geometry 参数中的 offsets. 使用它的一个典型情形是在 -region 这个选项中. -region 跟在一些其它的命令后, 用于指定一块矩形区域. 所以, 你除了需要指定这个矩形区域的宽和高, 还需要指定它的一个起始点 (左上角的点). 下面的第一个例子中, 我们指定了一个 100x200 的区域, 位置在 x=10, y=20, 或者我们应该写成 (x,y) = (10,20).

```
1  $ convert logo: -region '100x200+10+20' -negate wizNeg1.png
```

```
1  $ convert logo: -region '100x200-10+20' -negate wizNeg2.png
```

```
1  $ convert logo: -gravity center -region '100x200-10+20' -negate
       wizNeg3.png
```

注意, offsets 必须带上 +/-. 它表示的是一个相对偏移, 而不是一个绝对坐标. offsets 的参照点不是固定的, 但默认情况下, 它是 (0,0), 即左上角, 上面的第一个例子就是这种情况.

offsets 有可能 "出界", 就像第二个例子中的那样, -10+20, 对于这个有一部分出界的矩形执行 -negate, 实际的效果也就相当于 90x200+0+20.

第三个例子中, 一来就使用了 -gravity 选项, 它把当前坐标原点 (或叫参照原点) 设置为图像的正中, 即 (320,240) 的位置, 因为这张图的尺寸是 640x480. 这意味着后面的 offsets 的实际效果与前面两例就有所不同, 变成了 (320-10, 240+20) = (310,260). 同时 100x200 也不再是根据左上角来计算, 而是根据中心点计算. 即以 (310,260) 为中心的一个 100x200 的矩形. 显然, 它的左上角在 (310-50,260-100) = (260,160).

10 图像组 (Image Stack)

在学校时, 老师肯定有教过你, 在把结果直接写在答题卷上前, 或许你应该先在草稿上演算一番. 这里我们谈的 "图像组" 大概就是这个草稿纸的意思.

它让你在一个隔离的组中处理一张图像或一个图像序列, 处理完后, 把结果返回到正常流程中. 图像组用一对括号标示, 里面的所有操作只对当前组有效. 如下例, 我们限制 -rotate 操作只对 wizard.gif 有效:

```
1  $ convert wand.gif \( wizard.gif -rotate 30 \) +append images.gif
```

> 特别注意, 在 Unix 平台下, 用于图像组的括号是需要使用 \ 转义的, 因为小括号在 shell 中有其它的特殊作用. 但是, Windows 平台下的小括号不需要转义. 另外, 在小括号内的两侧, 都有一个空格, 请留意.

前面我们已经谈到了一些命令行中的操作项, 不过, 下面这几个操作项对于处理一个图像组是比较常用的:

```
1  -clone -delete -insert -swap
```

上面几个操作项的参数, 都是一个索引值, 用以表示在图像组中的某个图像. 图像组中的第一个图像的索引值是 0. 同时, 这个索引值你也可以使用负数, -1 就是图像组中的最后一个图像.

11 图像输出名

ImageMagick 扩展了原来的 “输出文件名” 的概念, 在 ImageMagick 中, 一个图像输出可以是:

1. 一种明确的图像格式.
2. 输出到系统的标准输出.
3. 文件名引用.

下面分别介绍一下他们.

11.1 明确的图像格式

图像信息都是以某种即定的格式存储起来的, 这些格式包括我们熟悉的 JPEG, PNG, TIFF 等. ImageMagick 在写出某个图像时, 必须知道应该使用哪一种格式写出. 通常情况下, ImageMagick 可以根据扩展名来判断格式, 如, 对于 `image.jpg` ImageMagick 会以 JPEG 格式来写出. 某些情况下没有给出图像的格式信息, 而你又没有明确指定, 这时, ImageMagick 会以图像原来的格式写出. 比如, 我们希望把图像存为名为 `image` 的原始 RGB 位图:

```
1  $ convert image.jpg rgb:image
```

11.2 标准输出

Unix 支持通过管道在各个程序间重定向输入输出. ImageMagick 可以使用一个特殊的文件名 - 来实现管道间的重定向操作. 下面的例子中, 我们把 `convert` 的输出重定向到 `display` 中:

```
1  $ convert logo: gif:- | display gif:-
```

这里, 你不一定要明确指定图像格式, GIF 这种格式有它自己的标示, ImageMagick 可以自动正确地识别出.

11.3 文件名引用

另外还有一种输出方式, 是使用格式化字符串来输出多个图像. 假如我们把输出图像名写成 `image-%d.jpg` , 同时我们的图像序列中有 3 个图像, 那么会得到如下的多个输出结果:

```
1  image-0.jpg
2  image-1.jpg
3  image-2.jpg
```

还有一种情况, 假如你希望用图像的某些属性来作为其文件名的一部分, 那么可以这样:

```
1  $ convert rose: -set filename:area '%wx%h' 'rose-%[filename:area].
       png'
```

其结果是:

```
1  rose-70x46.png
```