Playing Card Generator User Guide

Guide for version 1.0 of ‘cardgen’.

Introduction

A while back I was looking for software that would create playing card images. I found a bash script that would generate cards with pictures on them using the ‘convert’ command from ImageMagick. It was not what I wanted but it did inspire me to write software to generate a bash script[1] which could compose each card the way I wanted. This software gave me more control and flexibility for creating cards.

To compose the playing cards, ‘cardgen’ uses separate .png image files for the faces, indices and pips of the card. These are kept in separate directories, with a sub-directory for each style. This allows for different decks of cards to be generated from different combinations of these. We will see examples of this later. The intention is that ‘cardgen’ can be used to quickly prototype decks without having to create every individual component of the card from scratch. ‘cardgen’ exclusively uses.png image files because they support transparency.

To get started we first need a Linux style development environment, that has a C++ tool chain, make, tar and imageMagick installed. To use ‘cardgen’ it must be installed, then the environment providing the image files it needs must be set up.

Installing the ‘cardgen’ software

System requirements

The system requirements for building ‘cardgen’ are:

- Linux environment
- C++ tool chain
- make utility
- tar utility
- imageMagick

Most Linux environments have all but ‘convert’ command by imageMagick already installed. Running the following commands will either indicate that the applications are installed or supply the install command:

1. g++
2. make
3. convert

Ensure all commands are installed.

[1] Note the reason I generate a bash script rather than generating the card images directly is that it allowed me to easily compare bash scripts during development rather than comparing the generated card images themselves. Additionally, the script can be edited so that each card can be individually customised, if you’re up to the challenge of editing ‘convert’ commands.

Alternatively, for 'convert', go to the imageMagick download page and follow the instructions (https://imagemagick.org/script/download.php).

Obtaining the 'cardgen' source code

The 'cardgen' source code can be obtained in 2 ways, either from GitHub or from a tar file. For GitHub you need git installed and an internet connection, then execute the following commands:

```
1. git clone https://github.com/PhilLockett/cardgen.git
2. cd cardgen
```

This will also download the latest versions of this document and 'CardWork.tar.gz' which contains the required environment which contains the basic image components required as we shall see later.

The source can also be obtained from the tar file 'cardgen-1.0.tar.gz'. With the tar file execute the following commands:

```
1. tar zxf cardgen-1.0.tar.gz
2. cd cardgen-1.0
```

Installation steps

Once the 'cardgen' source code has been obtained we need to do the following to install it:

- Run the configure script: `./configure`
- Build it: `make`
- Install it: `sudo make install`
- Test it: `cardgen –help`

This installation process is only required the first time.

Cygwin build and install Example

Here is an example from my cygwin environment of installing 'cardgen' from a tar file. What you type is shown in bold. You should see similar output.

```
Phil@Gateway-Desktop ~
$ tar zxf cardgen-1.0.tar.gz

Phil@Gateway-Desktop ~
$ cd cardgen-1.0

Phil@Gateway-Desktop ~/cardgen-1.0
$ ./configure
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
checking for a thread-safe mkdir -p... /usr/bin/mkdir -p
checking for gawk... gawk
checking whether make sets $(MAKE)... yes
checking whether make supports nested variables... yes
checking for g++... g++
checking whether the C++ compiler works... yes
checking for C++ compiler default output file name... a.exe
checking for suffix of executables... .exe
checking whether we are cross compiling... no
checking for suffix of object files... o
```

```
checking whether we are using the GNU C++ compiler... yes
checking whether g++ accepts -g... yes
checking whether make supports the include directive... yes (GNU style)
checking dependency style of g++... gcc3
checking that generated files are newer than configure... done
configure: creating ./config.status
config.status: creating Makefile
config.status: creating src/Makefile
config.status: creating src/config.h
config.status: executing depfiles commands

Phil@Gateway-Desktop ~/cardgen-1.0
$ make
Making all in src
make[1]: Entering directory '/home/Phil/cardgen-1.0/src'
make  all-am
make[2]: Entering directory '/home/Phil/cardgen-1.0/src'
g++ -DHAVE_CONFIG_H -I.     -g -O2 -MT cardgen.o -MD -MP -MF
.deps/cardgen.Tpo -c -o cardgen.o cardgen.cpp
mv -f .deps/cardgen.Tpo .deps/cardgen.Po
g++ -DHAVE_CONFIG_H -I.     -g -O2 -MT desc.o -MD -MP -MF .deps/desc.Tpo -c -
o desc.o desc.cpp
mv -f .deps/desc.Tpo .deps/desc.Po
g++ -DHAVE_CONFIG_H -I.     -g -O2 -MT dump.o -MD -MP -MF .deps/dump.Tpo -c -
o dump.o dump.cpp
mv -f .deps/dump.Tpo .deps/dump.Po
g++ -DHAVE_CONFIG_H -I.     -g -O2 -MT init.o -MD -MP -MF .deps/init.Tpo -c -
o init.o init.cpp
mv -f .deps/init.Tpo .deps/init.Po
g++  -g -O2   -o cardgen.exe cardgen.o desc.o dump.o init.o
make[2]: Leaving directory '/home/Phil/cardgen-1.0/src'
make[1]: Leaving directory '/home/Phil/cardgen-1.0/src'
make[1]: Entering directory '/home/Phil/cardgen-1.0'
make[1]: Nothing to be done for 'all-am'.
make[1]: Leaving directory '/home/Phil/cardgen-1.0'

Phil@Gateway-Desktop ~/cardgen-1.0
$ sudo make install
Making install in src
make[1]: Entering directory '/home/Phil/cardgen-1.0/src'
make[2]: Entering directory '/home/Phil/cardgen-1.0/src'
 /usr/bin/mkdir -p '/usr/local/bin'
  /usr/bin/install -c cardgen.exe '/usr/local/bin'
make[2]: Nothing to be done for 'install-data-am'.
make[2]: Leaving directory '/home/Phil/cardgen-1.0/src'
make[1]: Leaving directory '/home/Phil/cardgen-1.0/src'
make[1]: Entering directory '/home/Phil/cardgen-1.0'
make[2]: Entering directory '/home/Phil/cardgen-1.0'
make[2]: Nothing to be done for 'install-exec-am'.
 /usr/bin/mkdir -p '/usr/local/share/doc/cardgen'
 /usr/bin/install -c -m 644 README '/usr/local/share/doc/cardgen'
make[2]: Leaving directory '/home/Phil/cardgen-1.0'
make[1]: Leaving directory '/home/Phil/cardgen-1.0'

Phil@Gateway-Desktop ~/cardgen-1.0
$ cd ..
```

```
Phil@Gateway-Desktop ~
$ cardgen --help
Usage: cardgen [Options]
  Generates the bash script "draw.sh" which draws a pack of playing cards.

  Options:
        --help                          This help page and nothing else.
        -v --version                    Display version.

        -i --index directory            Subdirectory of indices to use
(default: "1").
        -p --pip directory              Subdirectory of pips to use (default:
"1").
        -f --face directory             Subdirectory of faces to use
(default: "1").

        -s --script filename            Script filename (default: "draw.sh").
        -o --output directory           Output filename (default: same
directory name as face).
        -w --width integer              Card width in pixels (default: 380).
        -h --height integer             Card height in pixels (default: 532).
        -c --colour name                Background colour name (defined at:
http://www.imagemagick.org/script/color.php, default: "white").
        -a --KeepAspectRatio            Keep image Aspect Ratio (default:
false).

        --IndexHeight value             Height of index as a % of card height
(default: 10.5).
        --IndexCentreX value            X value of centre of index as a % of
card width (default: 8.07).
        --IndexCentreY value            Y value of centre of index as a % of
card height (default: 9.84).
        --CornerPipHeight value         Height of corner pip as a % of card
height (default: 7.5).
        --CornerPipCentreX value        X value of centre of corner pip as a
% of card width (default: 8.07).
        --CornerPipCentreY value        Y value of centre of corner pip as a
% of card height (default: 20.41).
        --StandardPipHeight value       Height of standard pip as a % of card
height (default: 18.5).
        --StandardPipCentreX value      X value of centre of standard pip as
a % of card width (default: 25.7).
        --StandardPipCentreY value      Y value of centre of standard pip as
a % of card height (default: 18.65).
        --ImageBoarderX value           Image Boarder in X direction as a %
of card width (default: 14.54).
        --ImageBoarderY value           Image Boarder in Y direction as a %
of card height (default: 10.14).
        --ImagePipOff                   Don't display image pip on the court
cards.
        --ImagePipHeight value          Height of image pip as a % of card
height (default: 14.29).
        --ImagePipCentreX value         X value of centre of image pip as a %
of card width relative to ImageBoarderX (default: 12.63).
        --ImagePipCentreY value         Y value of centre of image pip as a %
of card height relative to ImageBoarderY (default: 9.77).
```

```
        --CentreX value                 Shortcut for: --IndexCentreX value --
CornerPipCentreX value.
        --Inputs value                  Shortcut for: -f value -p value -i
value.
```

Once you see the help page you know the software is working, but to use it you need an environment set up.

Cygwin uninstall and remove 'cardgen'

If you don't like 'cardgen' and want to completely remove it from your system, follow these steps, otherwise skip to the next section. First, uninstall 'cardgen':

```
Phil@Gateway-Desktop ~
$ cd cardgen-1.0

Phil@Gateway-Desktop ~/cardgen-1.0
$ make uninstall
Making uninstall in src
make[1]: Entering directory '/home/Phil/cardgen-1.0/src'
 ( cd '/usr/local/bin' && rm -f cardgen.exe )
make[1]: Leaving directory '/home/Phil/cardgen-1.0/src'
make[1]: Entering directory '/home/Phil/cardgen-1.0'
 ( cd '/usr/local/share/doc/cardgen' && rm -f README )
make[1]: Leaving directory '/home/Phil/cardgen-1.0'
```

Now verify 'cardgen' has been removed by executing a command that should now fail:

```
Phil@Gateway-Desktop ~/cardgen-1.0
$ cardgen --help
-bash: cardgen: command not found

Phil@Gateway-Desktop ~/cardgen-1.0
$ cd ..
```

The source files and directories can be removed using a file explorer or by executing these commands:

```
Phil@Gateway-Desktop ~
$ rm -rf cardgen-1.0          <-  Be careful to only remove the cardgen-1.0 directory!

Phil@Gateway-Desktop ~
$ rm cardgen-1.0.tar.gz
```

Now all traces have been removed and you no longer need to read the rest of this document.

Set up environment

With 'cardgen' installed you need an environment in which to use it. This environment provides some basic images and scripts to get you started. This setting up is only required the first time and involves unzipping a tar file and running a setup script to complete the process. It is recommended that this is done outside of the 'cardgen' source code directory.

```
Phil@Gateway-Desktop ~
$ tar zxf CardWork.tar.gz

Phil@Gateway-Desktop ~
```

```
$ cd CardWork/

Phil@Gateway-Desktop ~/CardWork
$ ./setup.sh
```

The card composition takes place in the 'CardWork' directory. This 'CardWork' directory can be renamed, but the subdirectories it contains should not be renamed as 'cardgen' has their names hardcoded and looks for them explicitly.

Environment breakdown

The 'CardWork' directory provides the following subdirectories:

- boneyard
- faces
- indices
- pips
- scripts

The 'boneyard' contains additional image files used for generating default Jokers. The 'CJoker.png' and 'DJoker.png' images files should be copied into the selected 'indices' directory as required.

The 'faces' directory contains subdirectories of images used for the face cards. Typically, these are the court cards plus the ace of spades image. In fact, the only difference between style 1 and 2 is the colour of the ace of spades, which is why the setup script simply copies the face images from 1 to 2. The Rouen subdirectory is a reimagining of an old-style deck of cards but with added ace of spades and jokers.

The 'indices' directory contains subdirectories of images used for the card's indices. The only difference between style 1 and 2 is the colour of the club indices, black has been changed to white. Again, this is performed by the setup script, but we will see how to do that using scripts later in this guide. Note that subdirectory 3 contains four sets of indices. Typically, a deck only uses red and black indices, but this example uses different colours for each suit. If images for the spade indices are not found, clubs will be used for the spade cards. Similarly, if indices for hearts are not found, indices for diamonds will be used.

The 'pips' directory contains subdirectories of images used for the card's pips. The only difference between style 1 and 2 is the colour of the club and spade pips, black has been changed to white. Again, this is performed by the setup script, but we will see how to do that using scripts later in this guide. Note that subdirectory 3 contains only four pip images. These are used for both the standard pips and the corner pips. Optionally you can add a set of small pips which will be used as the corner pips for cases where standard pips and corner pips need to be different.

The 'scripts' directory contains numerous bash scripts that we will use throughout the rest of this guide.

All of these images are used to compose cards by resizing and positioning. It should be noted that if the selected 'faces' subdirectory contains an image for a card, the image is used, otherwise the standard pips arrangement is used.

Anatomy of a Playing Card

'cardgen' composes playing cards from five items and allows the position and size to be modified, but in a manor consistent with all individual instances of the item. For example, changing the position and size of the index changes both the top-left and bottom-right index. The five items are:

- Indices
- Corner Pip
- Standard Pip
- Face Image
- Face Pip

All image files must be .png files and it is highly recommended to use transparent backgrounds.

Indices

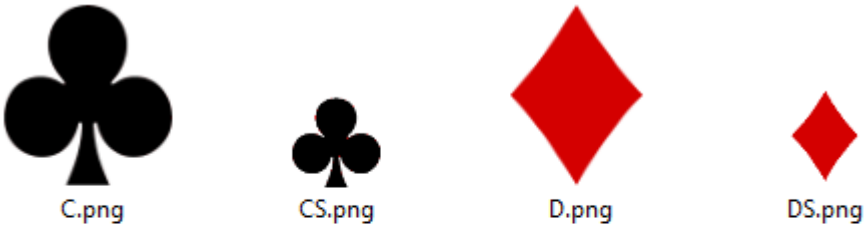
The index is the letter or number that appears in the two corners of the card. Images representing the indices are found in the 'indices' directory. The images are all .png files and use transparency as the background. The naming convention uses the initial letter of the suit followed by the index it represents. Obviously, 'C' represents Clubs, 'D' for Diamonds, 'H' for Hearts and 'S' for Spades. However, if the images for the Spades are not found, Clubs will be used because they are usually the same colour. Similarly, if indices for Hearts are not found, indices for diamonds will be used. This saves unnecessary repetition, but also allows for Hearts to be a different colour to Diamonds and Clubs to be a different colour to Spades, if desired. Example:

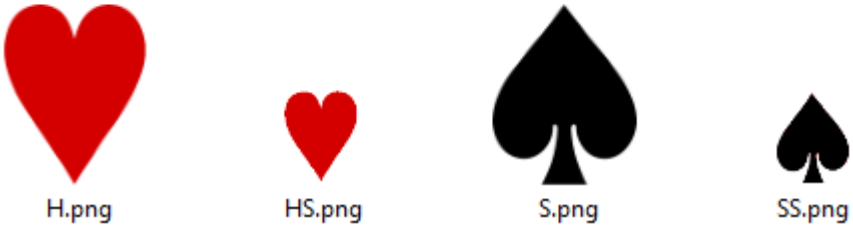
This directory can also contain images for the Joker text, which initially are in the 'boneyard' directory. This will be added to the corners of the generated joker cards. There are no controls provided for the Jokers and the only options are to provide a joker image which can be augmented with the joker text. The images are all .png files and use transparency as the background. The naming convention uses the initial letter of the suit followed by 'Joker'. In this case the joker text must be explicitly provided for each suit requiring it. In the example below only the Jokers for Clubs and Diamonds will have the text.

CJoker.png DJoker.png

Corner Pip

The Corner pip is the suit symbol displayed underneath the index. Images representing the Corner pips are found in the 'pips' directory. Like the indices, the pip images are all .png files and use transparency as the background. The naming convention uses the initial letter of the suit. These images are used for the standard pip, the corner pip and the face pip on the court cards. Optionally a second set of images can be provided named using the suit initial followed by an 'S' (for small) used specifically for the corner pip. This allows for the option to have simpler suit signs for the smaller corner pip and more complex designs for the others. If the smaller image is available it is used, if not the larger image is used.


C.png CS.png D.png DS.png

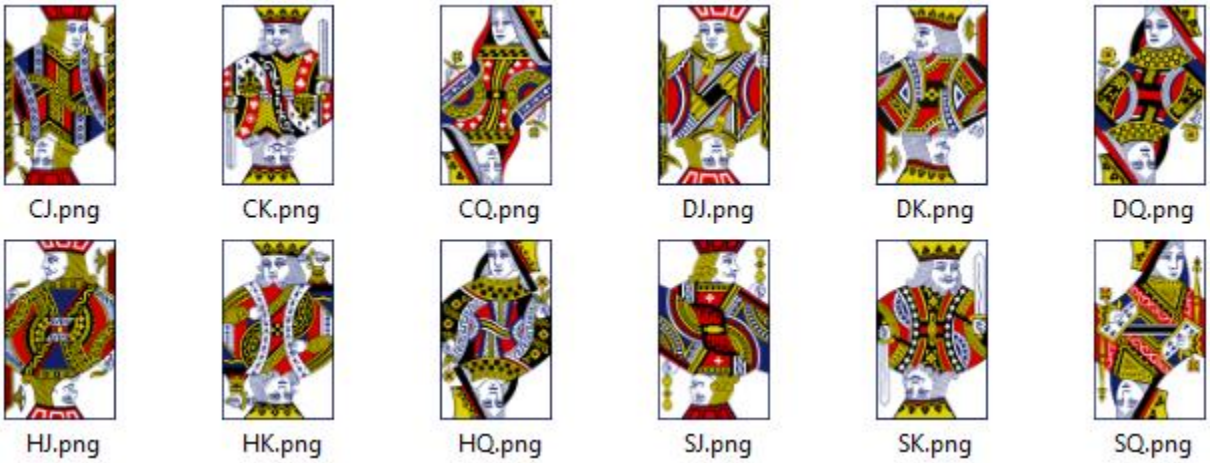

H.png HS.png S.png SS.png

Standard Pip

The Standard pip is the suit symbol displayed in the centre of the numeral cards (Aces through 10s). Like the Corner pips, the images representing the Standard pips are found in the 'pips' directory and are all .png files and use transparency as the background. The naming convention uses the initial letter of the suit.

Face Image

The Face image is the image that appears in the centre of the card. Images representing the faces are found in the 'faces' directory. The images are all .png files and use transparency as the background. The naming convention uses the initial letter of the suit followed by the index of the card.



Typically, these images represent the court cards, but may also include an image for any other card, such as the ace of spades (SA.png). Notice these court card face images do not include the suit symbol which is controlled by the Face pip item and is overlaid on the face image.

Face images are not restricted to the court cards and ace of spades. Any card can be represented by a face image using the naming convention of the initial letter of the suit followed by the index. Joker images can also be provided also, using the naming convention of the initial letter of the suit followed by 'Joker'.

Face Pip

The Face pip is the suit symbol displayed in the corner of the face image of the court cards (Jacks through Kings) and use the same image files as the Standard pips. The naming convention uses the initial letter of the suit.

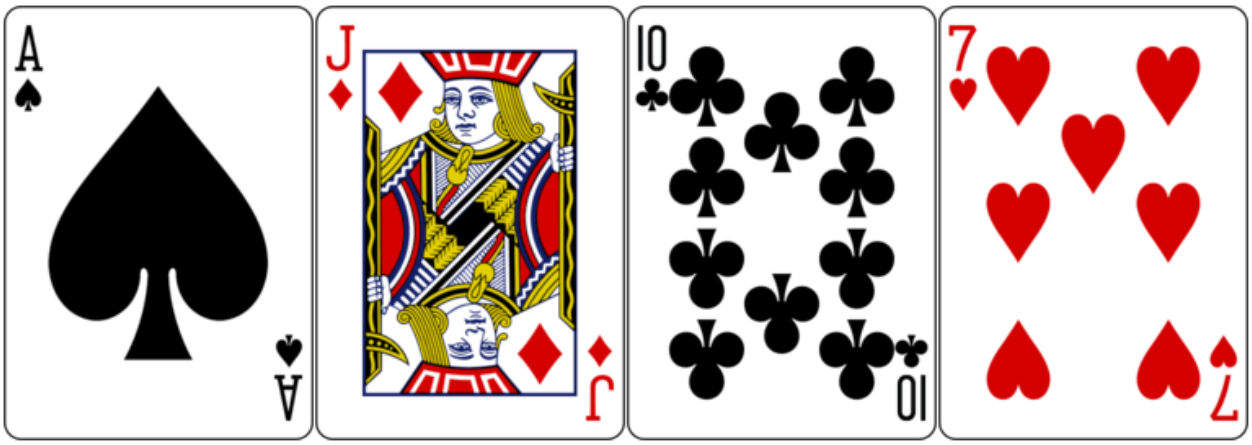
The Face pip item is overlaid on the face image of the court cards only. But can be suppressed as we shall see in the Rouen deck example.

A simple Example using 'cardgen'

The following example uses mostly default parameters, with the exception that we are maintaining the aspect ratio of the image files from the faces directory by using the -a parameter.

```
Phil@Gateway-Desktop ~/CardWork
$ cardgen -a

Phil@Gateway-Desktop ~/CardWork
$ ./draw.sh
Output created in cards/1/
```

This example instructs 'cardgen' to use the default images from faces/1, indices/1 and pips/1 and generates the draw.sh script file. When we run the script, card images are generated in cards/1.



Creating sheets

With your cards generated as individual files, you might want to see them in a single image. The following examples show how to use various scripts to generate sheets from the cards. Test sheet 1 is an 8 by 7 arrangement of cards in the order that is typically used by card printers. Test sheet 2 is a 7 by 8 arrangement and test sheet 3 is a 14 by 4 arrangement, both of these being in card order.

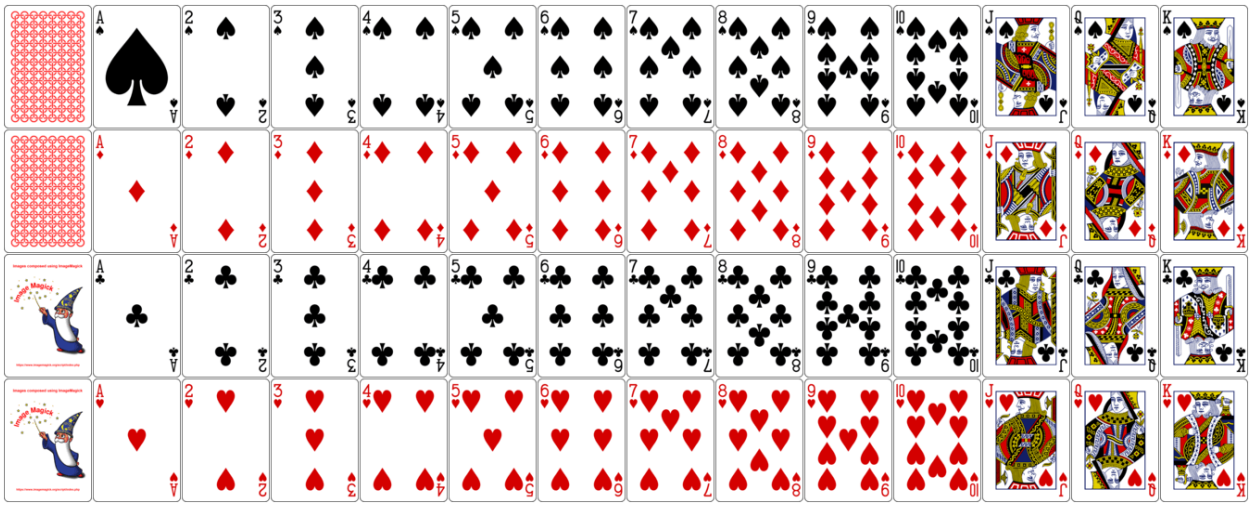
```
Phil@Gateway-Desktop ~/CardWork
$ cd cards/1/

Phil@Gateway-Desktop ~/CardWork/cards/1/
$ ../../scripts/gensheet1.sh

Phil@Gateway-Desktop ~/CardWork/cards/1/
$ ../../scripts/gensheet2.sh

Phil@Gateway-Desktop ~/CardWork/cards/1/
$ ../../scripts/gensheet3.sh
```

The above examples use the default values, but all 3 of these scripts take the following parameters:

-w	required card width in pixels (default 380)
-h	required card height in pixels (default 532)
-b	gap between cards in pixels (default 4)
-f	file name without the .png extension (default testsheet1, testsheet2 or testsheet3)

So, if these large files take up too much space, try generating smaller versions instead. Here we will generate a quarter size sheet as the file small.png:

```
Phil@Gateway-Desktop ~/CardWork/cards/1
$ ../../scripts/gensheet3.sh -w 95 -h 133 -b 1 -f small
```



Of course, there is no requirement to generate these sheet files at all.

The directory also contains a refresh script called x_refresh.sh. If disc space is at a premium you could delete all the .png files in the directory to save space, then simply run this script to generate the cards again when needed. Don't delete the script and it must still have access to the source images in the faces, indices and pips directories.

```
Phil@Gateway-Desktop ~/CardWork/cards/1/
$ ./x_refresh.sh
```

Note that the files here are named so that when listed alphabetically, all the cards appear first, followed by the sheets, then the refresh script. We're finished here, so back to base.

```
Phil@Gateway-Desktop ~/CardWork/cards/1/
$ cd ../..
```

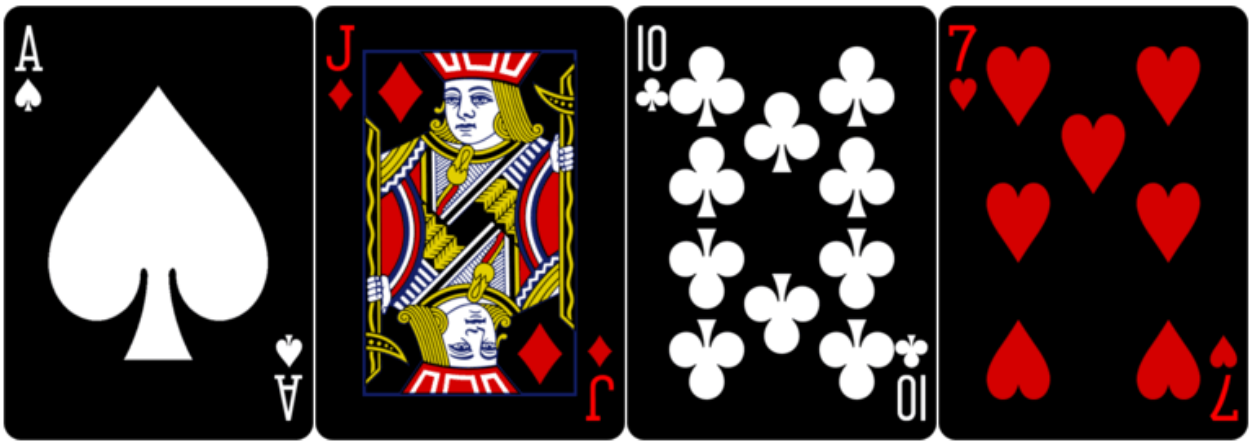
Generate some more decks

The following examples show how to use some more of the parameters.

Generate the Black deck

This example shows how to specify non default input files. Here we can use a short cut because the images we want are in faces/2, indices/2 and pips/2, so we can specify Inputs 2 instead of listing the input directories individually. It also shows how to set the background colour. This shows the advantage of using transparency in the .png files, the back ground colour shows through the component images.

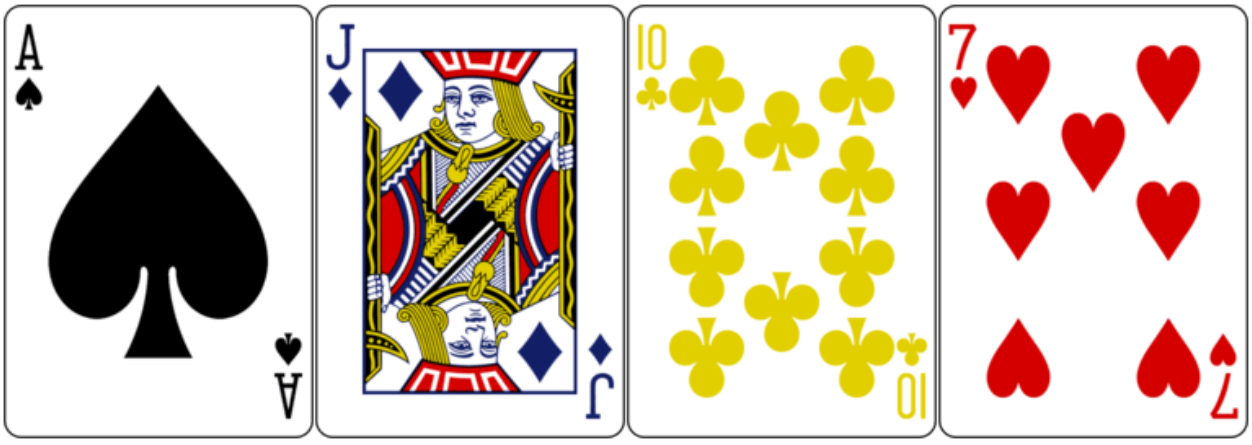
```
Phil@Gateway-Desktop ~/CardWork
$ cardgen -a --Inputs 2 -c black

Phil@Gateway-Desktop ~/CardWork
$ ./draw.sh
Output created in cards/2/
```



Generate the 4-colour deck

This example uses indices/3 and pips/3, but the default faces from faces/1. However, this means that the output, by default, would be generated in the cards/1 directory, overwriting the contents. To avoid that we specify output to cards/3 using the -o parameter. This example illustrates the advantage of removing the pips from the standard court faces and adding them during generation, they match the pips used for the rest of the suit.

```
Phil@Gateway-Desktop ~/CardWork
$ cardgen -a -i 3 -p 3 -f 1 -o 3

Phil@Gateway-Desktop ~/CardWork
$ ./draw.sh
Output created in cards/3/
```

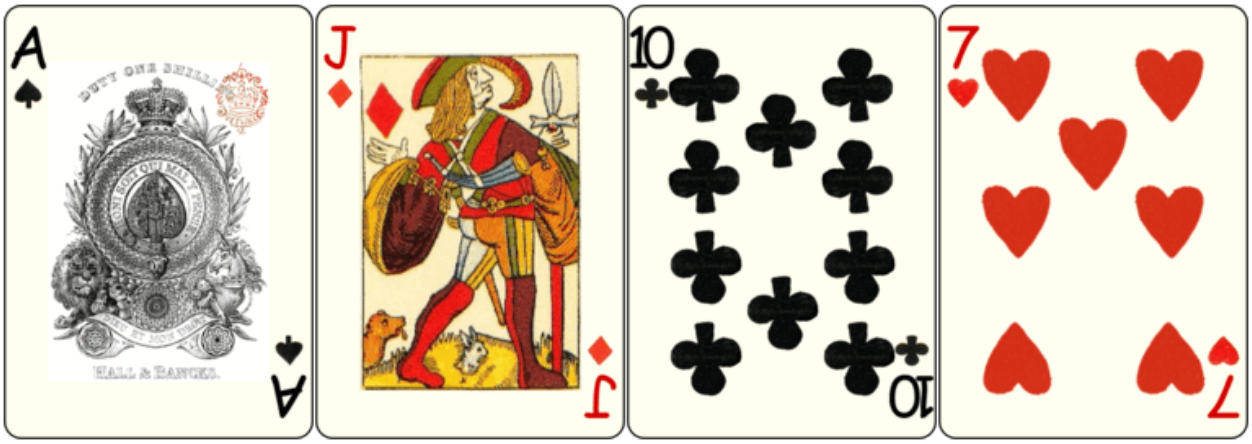


Generate the Rouen deck

This example uses custom faces, indices and pips. In this case the court card images are full length with individually placed pips, so we disable the generation of the pips on the images using the ImagePipOff parameter. This example also sets the background colour to give them an aged look.

```
Phil@Gateway-Desktop ~/CardWork
$ cardgen -a --Inputs Rouen -c ivory1 --ImagePipOff

Phil@Gateway-Desktop ~/CardWork
$ ./draw.sh
```

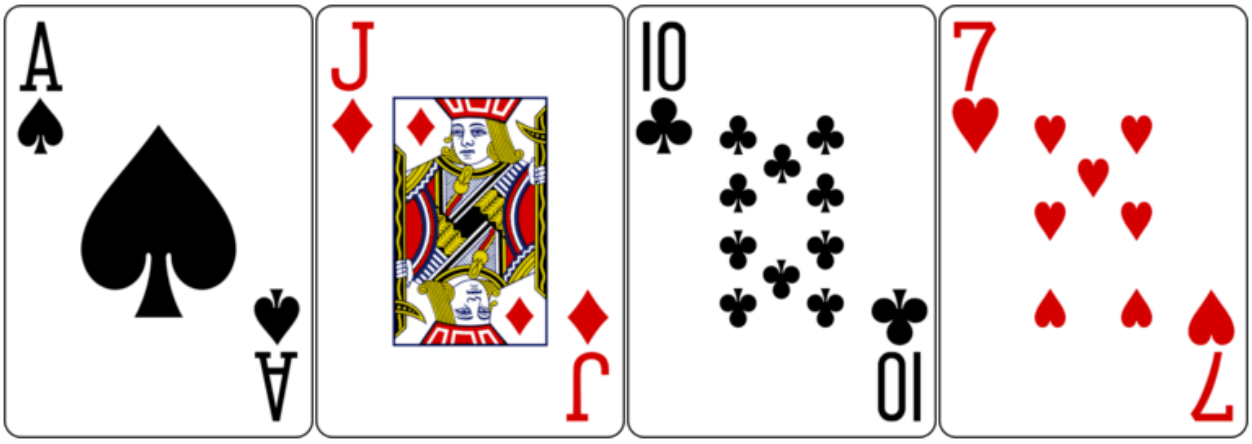
```
Output created in cards/Rouen/
```



Generate Jumbo index

In this example we use standard faces, indices and pips, but use several parameters to adjust the size and position of the images to create an approximation of Jumbo index cards.

```
Phil@Gateway-Desktop ~/CardWork
$ cardgen -a -o Jumbo --StandardPipHeight 9 --StandardPipCentreX 36 --
StandardPipCentreY 30 --CentreX 12 --IndexCentreY 12 --IndexHeight 16 --
CornerPipCentreY 28 --CornerPipHeight 13 --ImageBoarderX 25

Phil@Gateway-Desktop ~/CardWork
$ ./draw.sh
Output created in cards/Jumbo/
```



Here is a breakdown of the parameters used.

Maintain aspect ratios of images from the faces directory:

-a

Output images to cards/Jumbo directory:

-o Jumbo

Reduce the size of the standard pips and push them toward the centre of the cards:

--StandardPipHeight 9 --StandardPipCentreX 36 --StandardPipCentreY 30

The values here are percentage values of either the width or height of the card. The help page (cardgen --help) shows the details of this. The default StandardPipHeight is 18%, so here we have halved the size of it. Note that all images, except those from the faces directory, always maintain their aspect ratios, so we can only specify the height.

The default StandardPipCentreX is approximately 25.3% from the edge of the card. Here we have increased it, moving it toward the centre of the card. This affects both the left and right columns of pips, but obviously the centre column is unchanged.

The default StandardPipCentreY is approximately 18.6% from the top and bottom of the card. Here we have increased also, pushing the top and bottom rows toward the centre. The rows in between are then proportionally spaced.

Move the vertical centre position of the index and corner pip toward the centre:

--CentreX 12

We could have used IndexCentreX 12 and CornerPipCentreX 12 to move them both separately, but instead we used the short cut. Again, as the help page details, these are percentage values.

Increase size of the index and move it down:

--IndexCentreY 12 --IndexHeight 16

Increase size of the corner pip and move it down:

--CornerPipCentreY 28 --CornerPipHeight 13

Reduce the size of the images:

--ImageBoarderX 25

We only need to increase the size of the left boarder because we are maintaining the aspect ratio by using the -a parameter, so the whole image is reduced proportionally. By default, faces images are stretched to fit the space defined by ImageBoarderX and ImageBoarderY. Sometimes caution is required with the -a parameter as it is not obvious if the image size is determined by the X boarder or the Y boarder. It can be beneficial to remove the -a, adjust the boarders, then put the -a back, especially when reducing the boarder size.

Note that the image pips, seen here on the Jack of Diamonds, are automatically scaled based on the new size of the court card image.

Generating Customised cards

So far, we have generated cards using the images provided in the CardWork.tar.gz package. Now we are going to use scripts to create recoloured image files from the original files.

Note that these recolour scripts work best on the standard images in faces/1, indices/1 and pips/1. This is because the scripts are hardcoded to look for the specific red, blue, yellow, etc. that those files contain. These scripts simply replace one colour with another. If you have other images you want to use

with slightly different colours, you could try adjusting the fuzz parameter (-f) to a larger percentage value. Or you could modify the script.

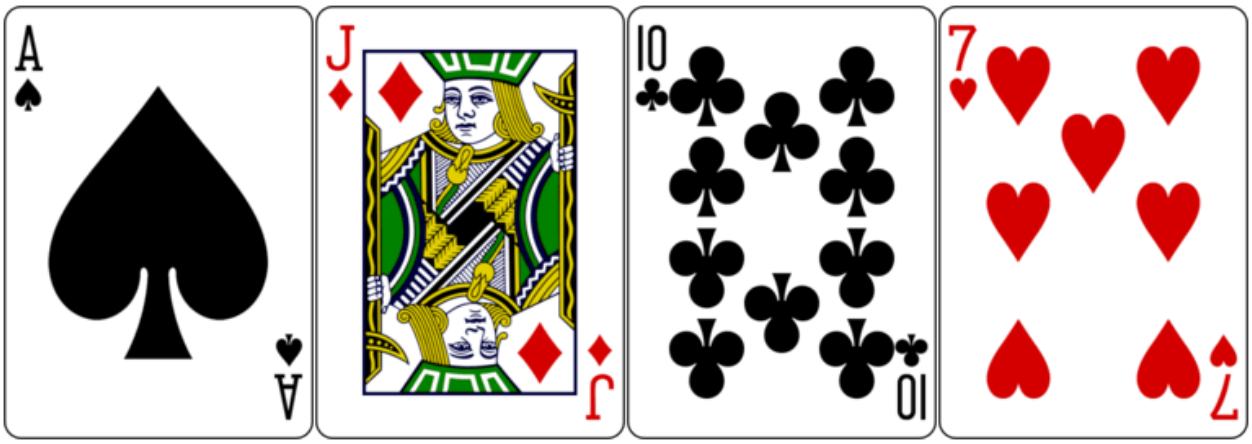
Generate the green deck

First, we will enter the faces/1 directory then use the recolourfaces script to change the red and blue colours. The script can also be used to change the yellow (-y), black (-k) and white (-w) colours. The red is changed to green and the blue is made darker. Notice that colour names can be used or rgb specifications. The colour specification needs to comply with those defined by ImageMagick. See the Example Usage here: http://www.imagemagick.org/script/color.php. In this case we use the -o parameter to generate the output in the faces/green directory.

```
Phil@Gateway-Desktop ~/CardWork
$ cd faces/1/

Phil@Gateway-Desktop ~/CardWork/faces/1
$ ../../scripts/recolourfaces.sh -r green -b 'rgb( 0, 0, 60)' -o green -f 1
red=green, yellow=rgb(226, 207, 0), blue=rgb( 0, 0, 60), black=rgb(0, 0, 0),
white=rgb(255, 255, 255), Leftovers:
```

Now generate the green cards.

```
Phil@Gateway-Desktop ~/CardWork/faces/1
$ cd ../..

Phil@Gateway-Desktop ~/CardWork
$ cardgen -a -f green

Phil@Gateway-Desktop ~/CardWork
$ ./draw.sh
Output created in cards/green/
```



Generate a custom deck

This time we will change the colour of the faces, indices and pips and add images for all the aces. In this example we darken the red, yellow and blue of the original images and create the new files in the directory faces/Mine.

```
Phil@Gateway-Desktop ~/CardWork
$ cd faces/1/
```

```
Phil@Gateway-Desktop ~/CardWork/faces/1
$ ../../scripts/recolourfaces.sh -r 'rgb(139,0,0)' -y 'rgb(218,165,32)' -b
'rgb(0,0,73)' -o Mine -f 1
red=rgb(139,0,0), yellow=rgb(218,165,32), blue=rgb(0,0,73), black=rgb(0, 0,
0), white=rgb(255, 255, 255), Leftovers:
```

Next, we darken the red indices and pips. Note that these scripts only support changing the red and the black colours.

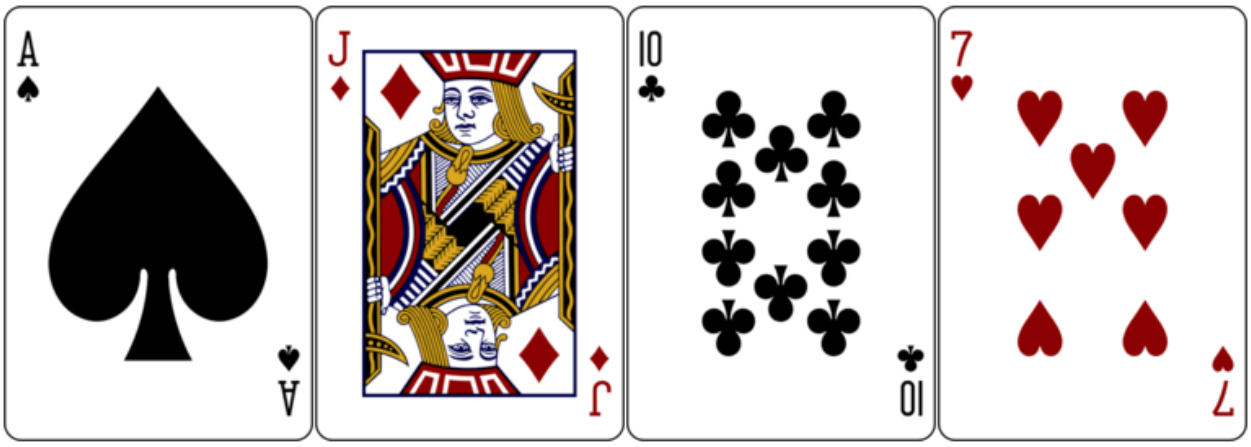
```
Phil@Gateway-Desktop ~/CardWork/faces/1
$ cd ../../indices/1/

Phil@Gateway-Desktop ~/CardWork/indices/1
$ ../../scripts/recolourindices.sh -r 'rgb(139,0,0)' -o Mine -f 1
red=rgb(139,0,0), black=rgb(0, 0, 0), Leftovers:

Phil@Gateway-Desktop ~/CardWork/indices/1
$ cd ../../pips/1/

Phil@Gateway-Desktop ~/CardWork/pips/1
$ ../../scripts/recolourpips.sh -r 'rgb(139,0,0)' -o Mine -f 1
red=rgb(139,0,0), black=rgb(0, 0, 0), Leftovers:
```

Now let's copy the standard pips and use them for the aces.

```
Phil@Gateway-Desktop ~/CardWork/pips/1
$ cd ../../faces/Mine/

Phil@Gateway-Desktop ~/CardWork/faces/Mine
$ cp ../../pips/Mine/C.png CA.png

Phil@Gateway-Desktop ~/CardWork/faces/Mine
$ cp ../../pips/Mine/D.png DA.png

Phil@Gateway-Desktop ~/CardWork/faces/Mine
$ cp ../../pips/Mine/H.png HA.png

Phil@Gateway-Desktop ~/CardWork/faces/Mine
$ cp ../../pips/Mine/S.png SA.png
```

Finally, we generate the cards, as before, with some minor size adjustments.

```
Phil@Gateway-Desktop ~/CardWork/faces/Mine
$ cd ../..

Phil@Gateway-Desktop ~/CardWork
$ cardgen -a --Inputs Mine --StandardPipHeight 13 --StandardPipCentreX 33 --
StandardPipCentreY 26 --CentreX 8 --IndexHeight 8 --CornerPipHeight 6 --
CornerPipCentreY 19 --ImagePipHeight 13

Phil@Gateway-Desktop ~/CardWork
$ ./draw.sh
Output created in cards/Mine/
```



Of course, all image files could be modified as required using your favourite image editor, but ensure transparency is maintained.

Generating new indices

The Rouen example above used custom indices, but they were actually generated using the following scripts. The testfonts.sh script creates a fonts directory and fills it with as many images as it can successfully generate. The images are named after the font name and contain an example of each of the indices. If you see a font you like, run the genindices.sh script with the font name and it will generate all the index images in the indices directory. These can then be used with the cardgen -i parameter.

```
Phil@Gateway-Desktop ~/CardWork
$ scripts/testfonts.sh
convert: unable to read font `Bitstream-Charter' @
warning/annotate.c/RenderType/964.
convert: unable to read font `Bitstream-Charter-Bold' @
warning/annotate.c/RenderType/964.
convert: unable to read font `Bitstream-Charter-Bold-Italic' @
warning/annotate.c/RenderType/964.
.
.
.
convert: unable to read font `Utopia-Bold-Italic' @
warning/annotate.c/RenderType/964.
convert: unable to read font `Utopia-Italic' @
warning/annotate.c/RenderType/964.
convert: unable to read font `Z003-Medium-Italic' @
warning/annotate.c/RenderType/964.

Phil@Gateway-Desktop ~/CardWork
$ scripts/genindices.sh -f DejaVu-Sans-Book
```

Additional packages

Now we are going to generate decks that use an image on every card.

Generate the Terry Pratchett deck

The first example uses book covers from Terry Pratchett books, which are portrait aspect images. Portrait aspect images are drawn once on the cards, as we have already seen for the court cards. These

additional packages are not part of the CardWork package but are optional additions. First download then unzip the additional package.

```
Phil@Gateway-Desktop ~/CardWork
$ tar zxf Pratchett.tar.gz
```

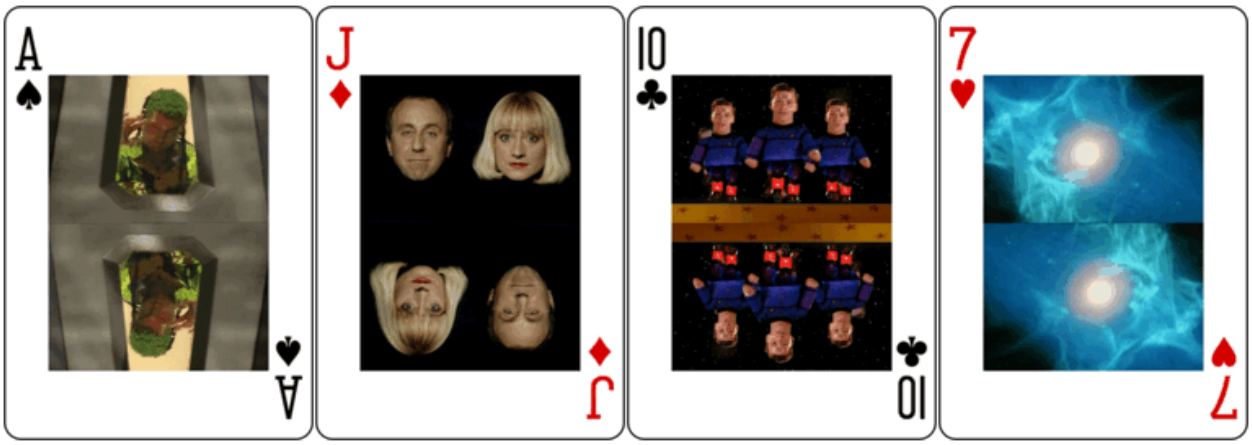
Now generate the cards as usual.

```
Phil@Gateway-Desktop ~/CardWork
$ cardgen -a -f Pratchett --ImagePipOff

Phil@Gateway-Desktop ~/CardWork
$ ./draw.sh
Output created in cards/Pratchett/
```

Generate the Red Dwarf deck

Now we will generate image cards using landscape images, which will be drawn twice on each card. For this we will use screen captures from the first 52 episodes of Red Dwarf.

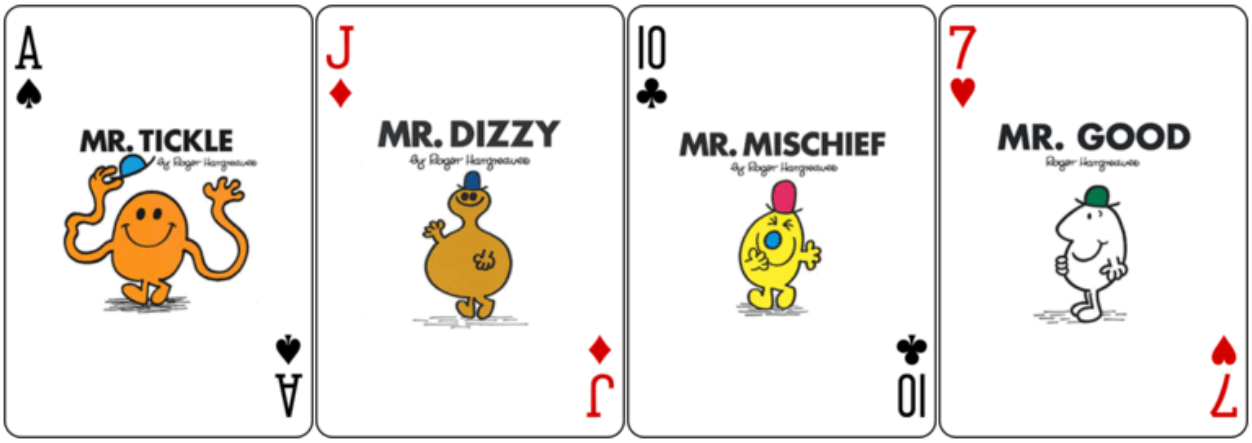
```
Phil@Gateway-Desktop ~/CardWork
$ tar zxf RedDwarf.tar.gz

Phil@Gateway-Desktop ~/CardWork
$ cardgen -a -f RedDwarf --ImagePipOff

Phil@Gateway-Desktop ~/CardWork
$ ./draw.sh
Output created in cards/RedDwarf/
```



Generate the Mr. Men deck

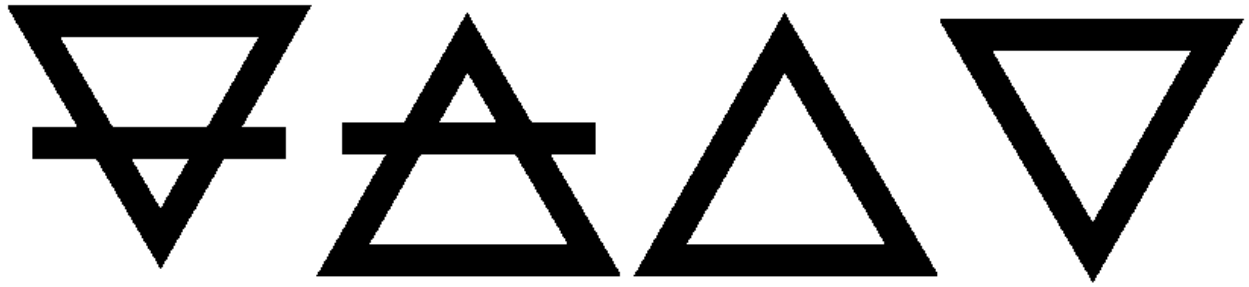
As an example of using a square image, we will use the Mr. Men book covers and note that 'cardgen' favours a single image here.

```
Phil@Gateway-Desktop ~/CardWork
$ tar zxf MrMen.tar.gz

Phil@Gateway-Desktop ~/CardWork
$ cardgen -a -f MrMen --ImagePipOff

Phil@Gateway-Desktop ~/CardWork
$ ./draw.sh
Output created in cards/MrMen/
```

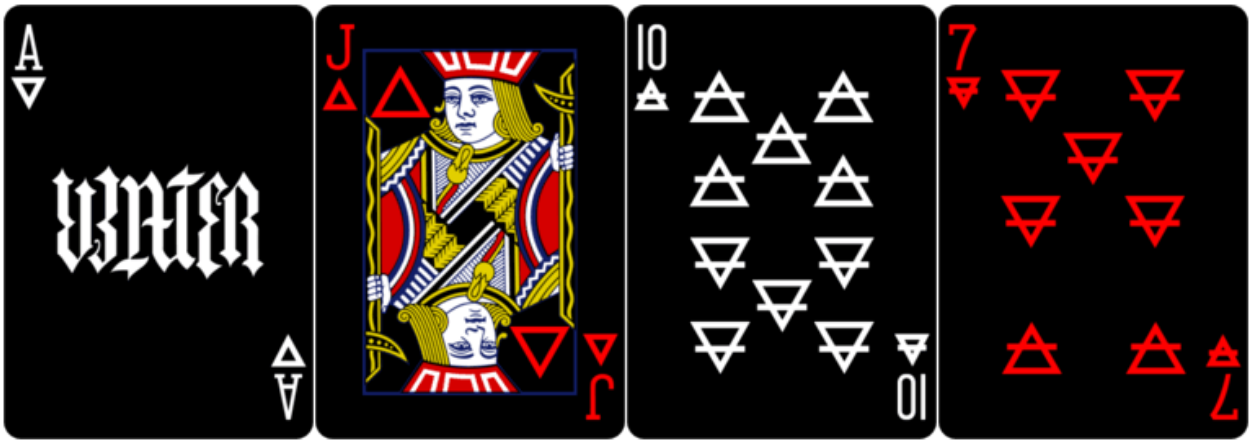


Generate the Angels & Demons deck

Now an example that uses custom pips and custom images for the aces. This deck is based on Dan Brown's Angels & Demons and uses the ambigram images from it for the aces, the advantage being that you can read the card which ever way up you are holding it. The deck also uses the ancient symbols for earth, air, fire and water. Angels are represented by the white pip cards of water and air. Demons are represented by the red pip cards of fire and earth.




```
Phil@Gateway-Desktop ~/CardWork
$ tar zxf AAD.tar.gz

Phil@Gateway-Desktop ~/CardWork
$ cardgen -a -f AAD -i 2 -p AAD -c black --StandardPipHeight 12 --
ImagePipHeight 12 --StandardPipCentreX 30 --StandardPipCentreY 21

Phil@Gateway-Desktop ~/CardWork
$ ./draw.sh
Output created in cards/AAD/
```

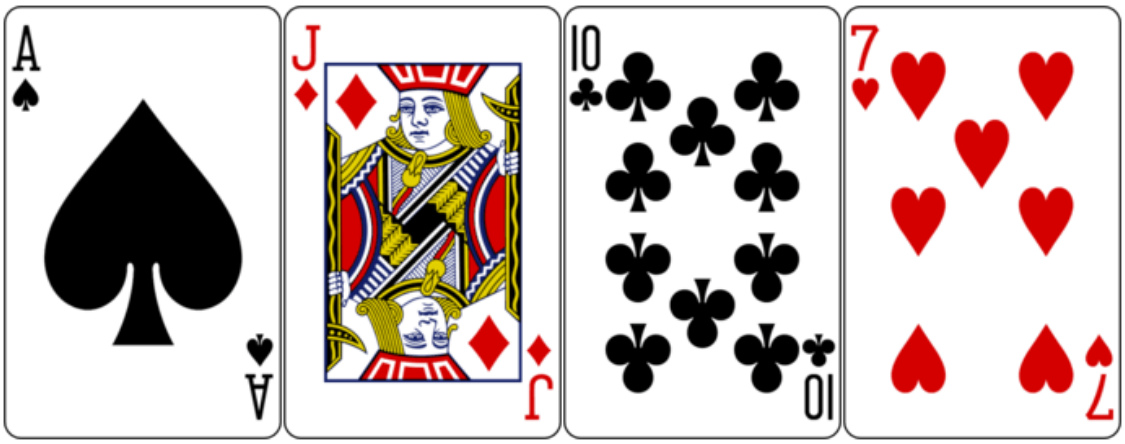


Generate the bridge deck

So far, all cards have been created as poker decks. Poker decks are 3.5 inches by 2.5 inches. The default width is 380 pixels and the default height is 532 pixels. Both of these can be changed using the 'cardgen' parameter -w and -h respectively. The narrower Bridge cards are also 3.5 inches high, but 2.25 inches wide, making it easier to hold more cards.

```
Phil@Gateway-Desktop ~/CardWork
$ cardgen -a -w 342 -o bridge --StandardPipHeight 16 --StandardPipCentreX 27

Phil@Gateway-Desktop ~/CardWork
$ ./draw.sh
Output created in cards/bridge/
```

Here we have set the width, in pixels, using the -w parameter. The pips are also reduced in size and moved over slightly due to the reduced area. Note that CornerPipCentreX is 27% of the width, which is now 342 pixels.



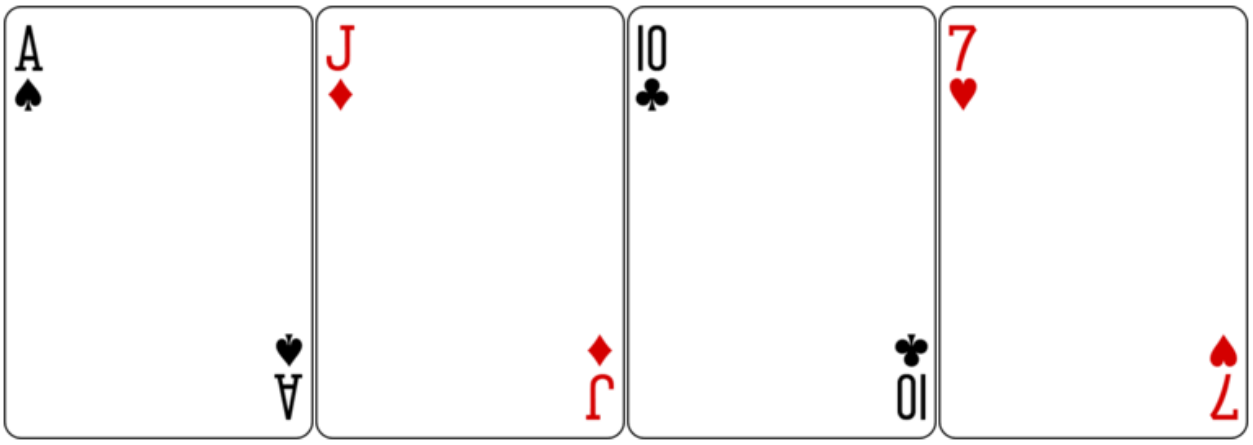
Mad cards

These examples explore some of the less obvious capabilities of 'cardgen'.

Generate the faceless deck

How about creating cards without faces or standard pips? Here we specify a faces directory that does not exist, so no court card images are drawn. We also set StandardPipHeight to 0 so that the standard pips are not drawn.

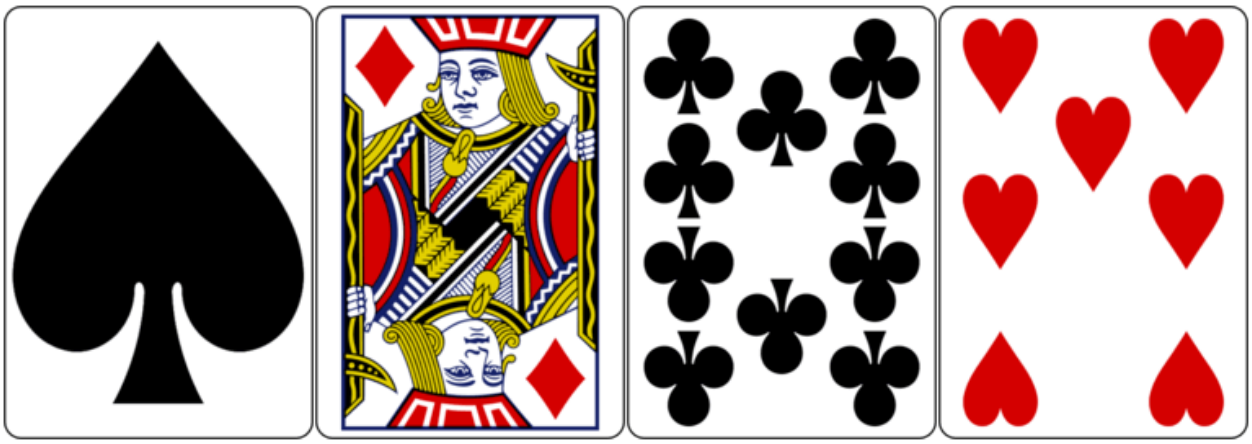
```
Phil@Gateway-Desktop ~/CardWork
$ cardgen -f 0 --StandardPipHeight 0

Phil@Gateway-Desktop ~/CardWork
$ ./draw.sh
Output created in cards/0/
```



Generate the big pips deck

For a long time playing cards didn't have corner pips or indices. For this example, we will turn off the corner pips and indices and make the standard pips and images bigger.

```
Phil@Gateway-Desktop ~/CardWork
$ cardgen -a -o bigpips --StandardPipHeight 22 --StandardPipCentreX 20 --
StandardPipCentreY 14 --CornerPipHeight 0 --IndexHeight 0 --ImageBoarderX 3 -
-ImageBoarderY 2 --ImagePipHeight 19 --ImagePipCentreX 14 --ImagePipCentreY
12

Phil@Gateway-Desktop ~/CardWork
$ ./draw.sh
Output created in cards/bigpips/
```



We have also adjusted the size and position of the pips on the court cards with these parameters:

--ImagePipHeight 19 --ImagePipCentreX 14 --ImagePipCentreY 12

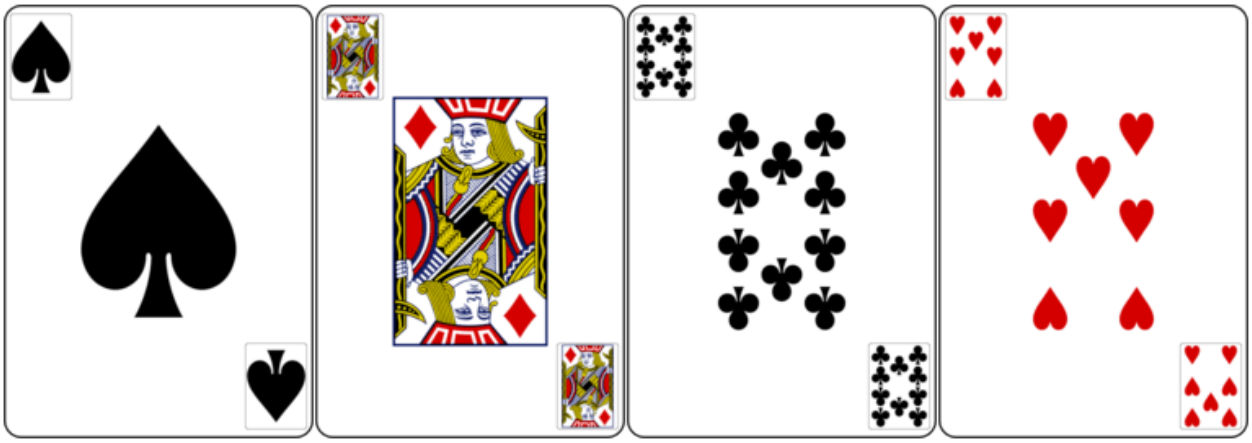
Note the position of the image pip is relative to the top left (and bottom right) corner of the image, not the card, so as the size of the image changes, the position of the pip stays the same, relative to the image. This only holds when the position is explicitly specified, but by default the image pip is adjusted automatically when the image size is changed. The Jumbo index deck example illustrates this.

Generate the old-style indices deck

One of the first implementations of corner indices was to print a small image of the card in one or both corners. Now that we have generated bigpips, we can use them for the indices. Here we make room for the indices by adjusting the size and position of the standard pips and court images. We also turn off the corner pips and increase the size of the indices and move them over to make room.

```
Phil@Gateway-Desktop ~/CardWork
$ cd indices/

Phil@Gateway-Desktop ~/CardWork/indices
$ cp -r ../cards/bigpips/ .

Phil@Gateway-Desktop ~/CardWork/indices
$ cd ..

Phil@Gateway-Desktop ~/CardWork
$ cardgen -a -i bigpips -o oldpips --StandardPipHeight 10 --
StandardPipCentreX 36 --StandardPipCentreY 30 --CornerPipHeight 0 --
IndexHeight 20 --IndexCentreX 12 --IndexCentreY 12 --ImageBoarderX 25

Phil@Gateway-Desktop ~/CardWork
```

```
$ ./draw.sh
Output created in cards/oldpips/
```



What this example illustrates is that the elements of these cards are images. Any of these images can be replaced with any other image. We have seen face images replaced or modified, pips replaced with ancient symbols and indices replaced with images of cards. Use any image you like, but for best results, set the background to transparent. For that, see the 'convert' example below, or use a graphics editor.

Using 'convert' directly

Here are some examples of using the 'convert' command directly from the command line which may be useful when customising your own decks. Of course there are many examples on the ImageMagick web site.

This example converts a .jpg to a .png file and resizes it. Of course, either step could be performed independently:

```
convert original.jpg -resize widthxheight original.png
```

This example replaces the colours black and white, with different colours:

```
convert DQ.png -fuzz 1% -fill black -opaque MidnightBlue -fill white -
opaque gold new.png
```

This example replaces the white background of the King of spades image (SK.png) with transparency. Rather than replacing all white with none, it uses flood fill in specific locations. If a colour is preferred to transparency, replace none with the colour:

```
convert SK.png -alpha set -channel RGBA -fuzz 40% -fill none -floodfill
+30+30 white -floodfill +1180+30 white -floodfill +30+1940 white -
floodfill +1180+1940 white new.png
```

These changes could also be made using a graphics editor.

Notes

All image files used are .png files because they provide transparency. To convert any image file to a .png file use 'convert' from the command line, it is what 'convert' was originally developed for.

The image files use a common brief naming convention of the initial letter of the suit followed by the card index.

If images for the spades indices are not found, clubs will be used because they are usually the same colour. Similarly, if indices for hearts are not found, indices for diamonds will be used.

Acknowledgement

Court cards images were initially created from Chris Aguilar's .svg files:

_____
Vectorized Playing Cards 2.0 - http://sourceforge.net/projects/vector-cards/
Copyright 2015 - Chris Aguilar
Licensed under LGPL 3 - www.gnu.org/copyleft/lesser.html