Visualizing IBM SAN Volume Controller FlashCopy Mappings

Dan Rumney

January 22, 2009

With the introduction of Multiple-Target FlashCopy® and Cascaded FlashCopy to IBM® System Storage™ *SAN Volume Controller* (SVC) , it can become difficult to keep track of which Virtual Disks are mapped to which, and how *FlashCopy Mappings* (FCM) depend upon one another. While submitting **svcinfo**commands or using the SVC Console can provide this information in a textual format, there are times when a diagram provides all the information you need in an easy to understand format.

This paper outlines a method for generating diagrams that link VDisks and FlashCopy Mappings. It also serves as a worked example of automation on the SVC Command Line. This paper assumes familiarity with SVC and the FlashCopy functionality.

1. SVC FlashCopy Mappings

When SVC was introduced in 2003 it included a number of Copy Services: FlashCopy and Remote Copy. FlashCopy is a Point-In-Time Copy Service, whereby the contents of a *Source Virtual Disk* (Source) is copied to a *Target Virtual Disk* (Target), such that the Target is an *exact* copy of the Source, at that point in time. The relationship between a Source and a Target is called a FlashCopy Mapping (FCM). The original implementation was such that a Source could only ever have one active Target. In addition to this, a Target of one FCM could not be the Source of another FCM. A number of FCMs can be gathered together into a *FlashCopy Consistency Group* (FCG) and managed as a single entity, to ensure that all Target VDisks in the FCG represent the exact same point in time.

With the release of SVC 4.2.1, new types of FlashCopy arrangements can be created. A single VDisk can act as the Source to multiple Targets. In addition to this, a VDisk which is acting as the Target of one FCM can also act as the Source of a different FCM. For more details on Copy Services in SVC, see the Redbook *SVC 4.2.1 Advanced Copy Services* [SG24-7574-00] [http://www.redbooks.ibm.com/abstracts/sg247574.html].

In a complex environment, the interactions between FCMs and VDisks can get quite involved. Dependencies between FCMs stem from internal data structures within the cluster rather than the logical connections between VDisks. All of these dependencies can be discovered by submitting the appropriate **svcinfo** commands, but the information is presented in a purely textual way; this provides no insight into the interaction between cluster objects. Example 1, “Sample output from SVC commands, viewing FCM interactions” shows how this output looks.

Example 1. Sample output from SVC commands, viewing FCM interactions

```
IBM_2145:cluster_name:admin>svcinfo lsfcmap -delim :
id:name:source_vdisk_id:source_vdisk_name:target_vdisk_id:target_vdisk_name:
group_id:group_name:status:progress:copy_rate:clean_progress:incremental
0:fcmap0:1039:vdisk1039:1040:vdisk1040:::idle_or_copied:100:93:100:on
1:fcmap1:1041:vdisk1041:1042:vdisk1042:::idle_or_copied:100:30:100:off
2:fcmap2:1043:vdisk1043:1044:vdisk1044:::idle_or_copied:100:88:100:on
3:fcmap3:1045:vdisk1045:1046:vdisk1046:::idle_or_copied:100:36:100:off
4:fcmap4:1046:vdisk1046:1047:vdisk1047:::idle_or_copied:100:96:100:on
IBM_2145:cluster_name:admin>
IBM_2145:cluster_name:admin>svcinfo lsfcmapdependentmaps -delim : 2
fc_id:fc_name
1:fcmap1
3:fcmap3
```

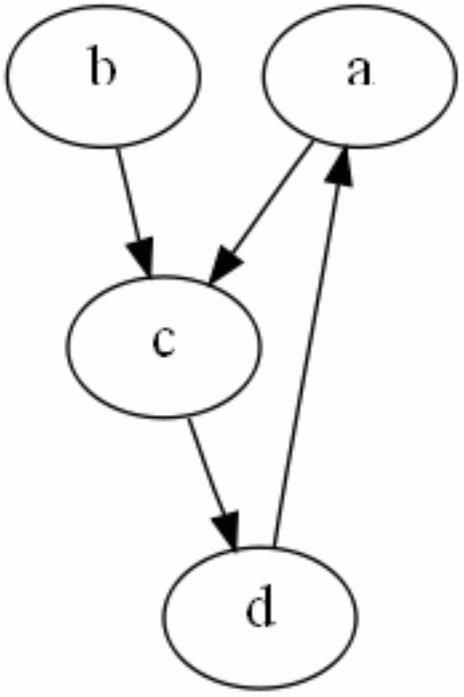
The output from the **svcinfo** command does not lend itself to a swift overview of the cluster state. We look to the DOT language to generate a graphical representation of this information.

2. The DOT Language

The DOT Language is a language used to describe directed and undirected graphs. Once written, the DOT can then be processed by an appropriate program, to render the graph on screen.

A *directed graph* consists of *nodes* and *edges*. A node is a graphical shape which may or may not contain some text. The nodes are interconnected by lines, called edges. In a directed graph, the edges have arrows at one or both ends. Figure 1, “Sample directed graph” shows a sample directed graph. Ellipses `a`,`b`, `c` and `d` are all nodes.

Figure 1. Sample directed graph



The DOT language is quite straightforward; for instance, the DOT required to generate Figure 1, “Sample directed graph” can be seen in Example 2, “DOT language for generating a directed graph”. The nodes are represented by the various letters. In each line in the example, you can see the 'arrow' notation which indicates a directed edge, joining two nodes.

Example 2. DOT language for generating a directed graph

```
digraph EXAMPLE
{
  a -> c;
  b -> c;
  c -> d;
  d -> a;
}
```

A DOT file only *describes* the graph. You must pass this file to a rendering program in order to generate a graphical representation. Once such program is **dot**, which is part of the Graphviz [http://graphviz.org/] package. Once installed, the invocation shown in Example 3, “Invocation to generate graphic from DOT language (Windows)” will generate a GIF file rendering of the sample directed graph described in Example 2, “DOT language for generating a directed graph”.

Example 3. Invocation to generate graphic from DOT language (Windows)

```
dot -T gif -o sampleDigraph.gif -K dot -v sampleDiGraph.dot
```

The capabilities of the DOT language lend themselves directly to the challenge of visualizing FlashCopy Mapping and VDisk relationships. The challenge is to generate the DOT necessary to generate our required visualization. In order to do this, we can use SVC Command Line Scripting.

3. SVC Command Line Scripting

The SVC Command Line Interface is based on a restricted Bash Shell. This provides us with a double-edged opportunity. On one hand, the Bash Shell means that we have the opportunity to execute scripts while logged in to the SVC Command Line. On the other hand, the restricted aspect strongly limits what we can do and results in the need for some imaginative scripting.

The Bash Shell includes a number of 'built-in' commands that can be used, such as:

if
while
for
read
echo

The restrictions on the Shell mean that there is no access to scripting standards such as **sed**, **awk** and **grep**. In addition, IO cannot be redirected to files. However, command output *can* be redirected to other commands, via pipes.

Appendix A, *Graph generating script* contains the script that we'll be discussing for the remainder of this paper. This script creates a CLI function which, when executed, generates DOT language which describes the connections between a set of VDisks and FCMs. The CLI function has one parameter, which is the id of one of the cluster's VDisks. Given a VDisk ID, this command follows the procedure below.

1. Search for any FCMs which have the provided VDisk as a Source or Target

2. For any FCMs found, identify the counterpart VDisks and repeat step 1 for each of the new VDisks

3. For all of the FCMs found, search for all dependant FCMs

4. Generate the DOT language, describing the VDisk and FCM interactions

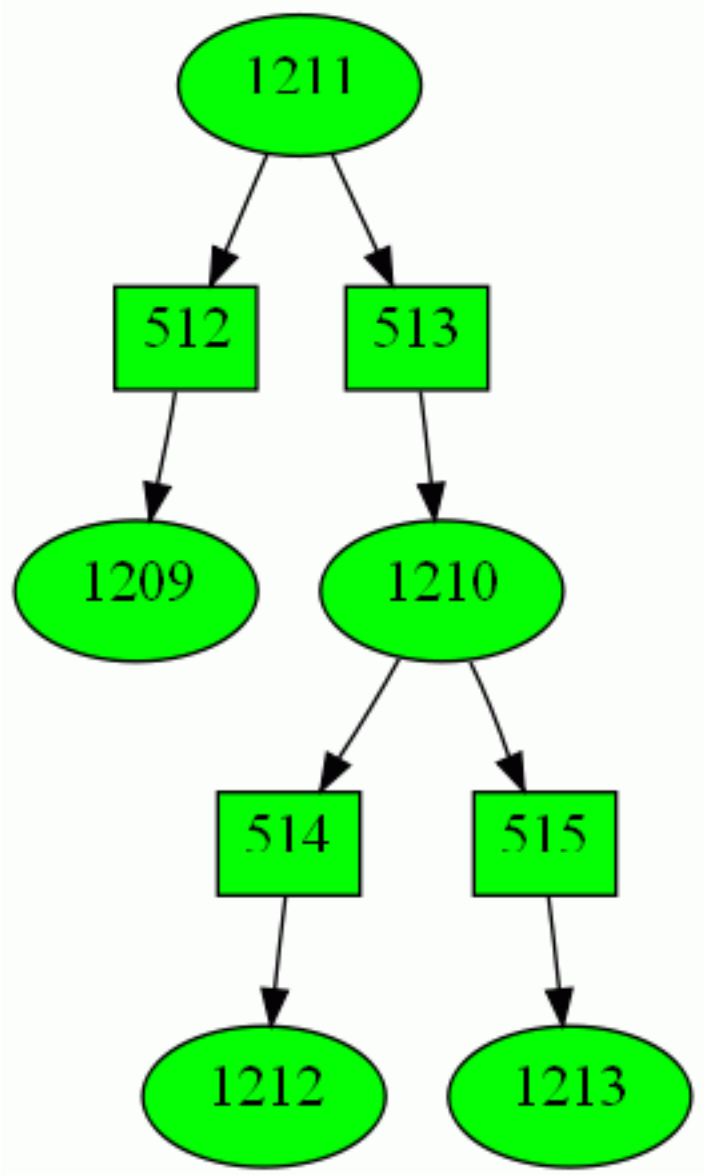
Example 4, “Sample output from script” shows example output from this function. When rendered by **dot**, the graphic shown in Figure 2, “Sample directed graph”

Example 4. Sample output from script

```
IBM_2145:cluster_name:admin>makeFCMapTree 1212
digraph F {
  1210 [style=filled,fillcolor=green]
  514 [shape=box,height=0.4,width=0.4,fillcolor=green,style=filled]
  1210 -> 514
  514 -> 1212
  515 [shape=box,height=0.4,width=0.4,fillcolor=green,style=filled]
  1210 -> 515
  515 -> 1213
  1212 [style=filled,fillcolor=green]
  1213 [style=filled,fillcolor=green]
  1211 [style=filled,fillcolor=green]
  512 [shape=box,height=0.4,width=0.4,fillcolor=green,style=filled]
  1211 -> 512
  512 -> 1209
  513 [shape=box,height=0.4,width=0.4,fillcolor=green,style=filled]
  1211 -> 513
  513 -> 1210
  1209 [style=filled,fillcolor=green]
}
```

The ellipses represent VDisks. The rectangles represent FlashCopy Mappings. The colours of the shapes represent the state of the FlashCopy Mappings and VDisks. In one glance, the interactions between VDisks and FlashCopy Mappings are clear and any issues are immediately obvious.

Figure 2. Sample directed graph



In the next section, we will go through this script line by line and explain what is being done. The techniques used in this script can be used in scripts of your own to perform whatever actions you require. If you prefer, you can skip to Section 5, “Using the script” to learn how to use the script.

4. Script Analysis

When the script in Appendix A, *Graph generating script* is executed, it creates a new function called `makeFCMapTree` in the active SVC CLI session. The script itself has *no* output. Once the function has been created, it is invoked by providing the function name and a VDisk ID. It generates output in the DOT language which can be captured and saved for later rendering. In this section, we analysis the script section by section and explain its function.

4.1. Lines 1-18

The first 18 lines are concerned with setting up variables for use during the main loop.

Lines 1-15. The fifteen first lines simply start the function and set up some variables assigning colours to FCM and VDisk states. These can be changed to suit your needs, with the requirement that all 10 states are present and the colours that are selected are part of the DOT language. Appendix B, *Valid colours in the DOT language* shows the colours supported by DOT.

Line 16. This line outputs the first line of DOT, which indicates that we'll be describing a directed graph. We call it F for FlashCopy, but the name is entirely arbitrary (within the constraints of the DOT language)

Line 17. In this line, we create an array called `$possSrcs`. This array will be treated as a FIFO stack. It will contain a list of VDisks which are possibly acting as the Sources of an FCM. At this point in the execution, we add the VDisk ID that was provided as the sole parameter to the function. In subsequent passes through the main loop, new VDisk IDs may be added, as required.

Line 18. Here, we create an empty array which will keep track of all VDisk IDs which have been processed. This is to ensure that the script processes each VDisk ID once and once only.

4.2. Lines 19-26

These lines start the main loop and gather information about the next VDisk ID in the `$possSrcs` stack.

Line 19. This starts the main loop of this script. The loop will execute as long as there are VDisk IDs in `$possSrcs`. This loop ends on line 56.

Lines 20-21. Lines 20-21 perform the `pop` operation of a stack, removing the first element from `$possSrcs` and assigning it to `$currSrc`.

Lines 22-26. Lines 22-26 show a technique that will be repeated a number of times in this script. This technique is to run an **svcinfo** command and execute a series of commands based on each line of output from the **svcinfo** command.

The technique has the following form:

```
svcinfo xxxx | while read var1 var2 var3 rest; do
          some commands using $var1, $var2, etc
done
```

This technique will take each line of output from the **svcinfo** command and pass it to the **read** built-in command. The read command works in the following way:

`read` [*name* ...]

The **read** command takes a line from STDIN and splits it into words separated according to the *Internal Field Separator* (IFS). By default, IFS is set to whitespace. The first word is assigned to the first `name`, the second to the second `name` and so on. Any leftover words are assigned to the final name with the intervening IFS included.

In the technique shown above, the variables `$var1`, `$var2`, etc can now be used by commands inside the **while** loop. The loop will repeat once per line of output from the **svcinfo** command. The variable `$rest` is needed to capture any remaining values at the end of the line of output.

The command `svcinfo lsvdisk -nohdr -filtervalue id=$currSrc` will generate zero or one line of output. (depending on whether the value in `$currSrc` is an actual VDisk ID).

Line 24. This line acts to 'dereference' the VDisk status. The code `$(eval echo \$$vStatus)` will take the value stored in `$vStatus` and treat it as the name of a variable and try to find the value stored in that. For instance, if `$vStatus` is equal to 'preparing' then this line will look for the value found in `$preparing` and, in this case, return 'yellow'. In this way, we can assign colours to VDisk states.

Line 25. This generates lines of DOT which define a node in the graph. In this instance a VDisk node is generated. The VDisk node is an ellipse, filled in a colour representing the VDisk's state.

4.3. Lines 27-30

These two lines generate an array of FCM Target VDisk IDs and mark the current VDisk ID (held in `$currentSrc`) as having been processed.

Lines 27-29. Lines 27-29 create a new array that holds all of the FCMs which have the current VDisk (represented by `$currSrc`) as the Source VDisk. This array is called `$newTgts`. These three lines shows a method of generating an array directly from the output of an **svcinfo** command. In this particular instance, each line of **svcinfo** output generates 3 array elements:

`$newTgts[L]`	FlashCopy Mapping ID
`$newTgts[L+1]`	Target VDisk ID
`$newTgts[L+2]`	FlashCopy Mapping status

where `L` increases by one per line of **svcinfo** output.

Line 30. Line 30 keeps track of the fact that we have now processed the VDisk ID(as an FCM Source... it may appear later as an FCM Target).

4.4. Lines 31-46

These lines process the FCMs that were placed into `$newTgts` and generate the appropriate DOT to represent them.

Line 31. Lines 31 creates the loop to process `$newTgts`. It creates an index variable which is incremented by 3 for each pass (since 3 elements in an array represent *one* FC Mapping).

Lines 32-35. Lines 32 to 34 simply collect the relevant array elements in to clearer variable names. Line 35 decodes the FC Mapping status into a colour, much like line 24.

Lines 36-39. Lines 36-39 generate DOT language; lines 36 and 37 creates an FC Mapping node, which is a square filled with a colour that represents its state.

Lines 38 and 39 generate the edges that link the FC Mapping node with its Source and Target VDisks.

Lines 40-42. Lines 40-42 look at the FCM Target and determine whether or not it has been processed as a Source. If it has not, it is added to our list of possible Sources: `$possSrcs`. The code in line 41 acts to place this new VDisk ID at the end of the stack.

Lines 43-45. Lines 43-45 look to see if there are any dependencies between this FCM and other FCMs. If there are, a new edge is generated to indicate this.

4.5. Lines 48-55

Lines 48-55 perform a similar task to line 27 and the loop that follows it.

Line 48. Line 48 generates an array of FCM Source VDisk IDs, which have `$currentSrc` as their Target. This array is called `$newSrcs`. Instead of generating any DOT in the loop, however, this loop simply adds the indicated VDisk IDs to the `$possSrcs` array, if they have not previously been processed.

4.6. Lines 56-58

All that remains at this point is to complete the loops, close out the DOT language with a curly bracket and the function is complete.

5. Using the script

There are two simple ways to use the script. It can be executed directly from an interactive CLI session, or it can be used as part of a batch session.

5.1. Interactive session

Adding the `makeFCTreeMap` function to an interactive session is very straightforward. Simply copy the full text of the script into the clipboard and then paste it into the terminal. Once the script has executed, the `makeFCTreeMap` function will be available to you for the remainder of that CLI session.

Once you've passed a VDisk ID to the function, you will need to copy the output from the CLI session and place it into a dot file for rendering.

5.2. Batch session

Adding the `makeFCTreeMap` function to a batch session depends on your SSH client. Here, we will discuss PuTTY for the Windows operating system and **ssh** for Linux or AIX®.

Whichever operating system you use, the output from the script will be returned to the STDOUT stream on your local system. You can redirect this output to a dot file, and then pass it to your rendering application.

5.2.1. Required script changes

When a script is submitted to the SVC Cluster in this way, STDIN is replaced by the contents of the script and executed as if it was typed in manually. Once the end of the file is reached, control returns to the local command line and *not* the SVC command line. Since the normal function of the makeFCTreeMap script is to create a new function in the CLI session (and nothing more), the following modifications are needed to generate output:

- Delete lines 1 and 2
- Delete line 58
- Replace `$1` in line 17 with the ID of the VDisk that you're interested in.

Once these changes have been made, the resulting `script_file` should be submitted to the cluster using one of the methods shown in the next subsections.

5.2.2. PuTTY

The **plink** command comes as part of the PuTTY application. Example 5, “Submitting script to an SVC cluster using plink” shows the command to use to submit a script to an SVC cluster.

Example 5. Submitting script to an SVC cluster using plink

```
plink -l admin -m script_file -i private_key_file cluster_name
```

`script_file`	The file containing the makeFCTreeMap script
`private_key_file`	An SSH private key which corresponds to a public key that has been uploaded to the SVC cluster in question
`cluster_name`	The IP address or DNS name

5.2.3. SSH

The **ssh** command comes with most (if not all) *nix operating systems. Example 6, “Submitting script to an SVC cluster using ssh” shows the invocation required to submit a script to an SVC cluster using **ssh**.

Example 6. Submitting script to an SVC cluster using ssh

```
ssh -i private_key_file -T admin@cluster_name < script_file
```

`script_file`	The file containing the makeFCTreeMap script
`private_key_file`	An SSH private key which corresponds to a public key that has been uploaded to the SVC cluster in question
`cluster_name`	The IP address or DNS name

6. Possible Improvements

The script in Appendix A, *Graph generating script* functions correctly for all possible VDisk IDs, including ones that are not present on the cluster. However, there are some interesting changes that could be made to enhance the script. These are offered as suggestions and are left to the reader to implement:

Possible improvements to script

Handle Multiple VDisk IDs	The current version of the `makeFCTreeMap` only accepts a single VDisk ID. It would be a fairly simple task to change the script to allow the function to accept any number of VDisk IDs. It would be important to check for duplicate VDisk IDs appearing in the `$possSrcs` stack.
Handle FlashCopy Mapping IDs	Expanding this script to support FCM IDs instead of VDisk IDs is the straightforward task of taking the FCM ID, determining the Source VDisk's ID and placing this into the `$possSrcs` stack and then proceeding as before. The challenge lies in making the one script support VDisk IDs *and* FCM IDs.
Handle FlashCopy Consistency Group IDs	Handling FlashCopy Consistency Groups is the natural combination of handling multiple VDisk IDs and handling FlashCopy Mappings, since an FCG is simply a group of FCMs.

The following code will turn an FCG id into an array of FCM ids:
`fcmIDs=(`svcinfo lsfcmap -nohdr -filtervalue group_id=0 | while read fcmid rest; do echo $fcmid; done`)`

A. Graph generating script

The script below has been formatted so that it will fit onto the page. As a result, line continuation operators have been used on lines 22, 27, 28, 36 and 48.

```
 1 makeFCMapTree ()
   {
     # Define the colouring for FC Mapping and VDisk states
     idle_or_copied=green
 5   preparing=yellow
     prepared=green
     copying=green
     stopped=red
     suspended=red
10   stopping=yellow
     online=green
     offline=red
     degraded=yellow

15   # Start the directed graph
     echo "digraph F {";
     possSrcs=($1);
     processed=();
     while [ ${#possSrcs[@]} -gt 0 ]; do
20     currSrc=${possSrcs[0]};
       possSrcs=(${possSrcs[@]:1});
       svcinfo lsvdisk -nohdr -filtervalue id=$currSrc | while read id name \
   iogId iogName vStatus junk; do
         vdkColour=$(eval echo \$$vStatus);
25       echo "$currSrc [style=filled,fillcolor=$vdkColour]";
       done
       newTgts=(`svcinfo lsfcmap -nohdr -filtervalue source_vdisk_id=$currSrc \
   -delim :| while IFS=: read id n srcId srcName tgtId tgtName gId gName \
   status junk; do echo "$id $tgtId $status"; done`);
30     processed[$currSrc]=y;
       for ((i=0; i<${#newTgts[@]};i=$(($i + 3)))); do
         fcm=${newTgts[$i]};
         tgt=${newTgts[$(($i + 1))]};
         status=${newTgts[$(($i + 2))]};
35       colour=$(eval echo \$$status);
         echo "fc$fcm [label=\"$fcm\"shape=box,height=0.4,width=0.4,\
   fillcolor=$colour,style=filled]";
         echo "$currSrc -> fc$fcm";
         echo "fc$fcm -> $tgt";
40       if [ "${processed[$tgt]}" != "y" ]; then
         possSrcs=(${possSrcs[@]} $tgt);
         fi;
         svcinfo lsfcmapdependentmaps -nohdr $fcm | while read fcId fcName; do
```

```
          echo "$fcm -> $fcId [style=dotted]"
45      done
      done;

      newSrcs=(`svcinfo lsfcmap -nohdr -filtervalue target_vdisk_id=$currSrc| \
   while read id name srcId junk; do echo "$srcId "; done`);
50    for src in ${newSrcs[@]};
      do
        if [ "${processed[$src]}" != "y" ]; then
        possSrcs=(${possSrcs[@]} $src);
        fi;
55    done;
    done;
    echo "}";
   }
```

B. Valid colours in the DOT language

The following are acceptable colours in the DOT language: This list can also be found at *Graphviz Color Names* [http://www.graphviz.org/doc/info/colors.html]

```
aliceblue        gray18           grey74           orange1
antiquewhite     gray19           grey75           orange2
antiquewhite1    gray20           grey76           orange3
antiquewhite2    gray21           grey77           orange4
antiquewhite3    gray22           grey78           orangered
antiquewhite4    gray23           grey79           orangered1
aquamarine       gray24           grey80           orangered2
aquamarine1      gray25           grey81           orangered3
aquamarine2      gray26           grey82           orangered4
aquamarine3      gray27           grey83           orchid
aquamarine4      gray28           grey84           orchid1
azure            gray29           grey85           orchid2
azure1           gray30           grey86           orchid3
azure2           gray31           grey87           orchid4
azure3           gray32           grey88           palegoldenrod
azure4           gray33           grey89           palegreen
beige            gray34           grey90           palegreen1
bisque           gray35           grey91           palegreen2
bisque1          gray36           grey92           palegreen3
bisque2          gray37           grey93           palegreen4
bisque3          gray38           grey94           paleturquoise
bisque4          gray39           grey95           paleturquoise1
black            gray40           grey96           paleturquoise2
blanchedalmond   gray41           grey97           paleturquoise3
blue             gray42           grey98           paleturquoise4
blue1            gray43           grey99           palevioletred
blue2            gray44           grey100          palevioletred1
blue3            gray45           honeydew         palevioletred2
blue4            gray46           honeydew1        palevioletred3
blueviolet       gray47           honeydew2        palevioletred4
brown            gray48           honeydew3        papayawhip
brown1           gray49           honeydew4        peachpuff
```

```
brown2          gray50          hotpink          peachpuff1
brown3          gray51          hotpink1         peachpuff2
brown4          gray52          hotpink2         peachpuff3
burlywood       gray53          hotpink3         peachpuff4
burlywood1      gray54          hotpink4         peru
burlywood2      gray55          indianred        pink
burlywood3      gray56          indianred1       pink1
burlywood4      gray57          indianred2       pink2
cadetblue       gray58          indianred3       pink3
cadetblue1      gray59          indianred4       pink4
cadetblue2      gray60          indigo           plum
cadetblue3      gray61          ivory            plum1
cadetblue4      gray62          ivory1           plum2
chartreuse      gray63          ivory2           plum3
chartreuse1     gray64          ivory3           plum4
chartreuse2     gray65          ivory4           powderblue
chartreuse3     gray66          khaki            purple
chartreuse4     gray67          khaki1           purple1
chocolate       gray68          khaki2           purple2
chocolate1      gray69          khaki3           purple3
chocolate2      gray70          khaki4           purple4
chocolate3      gray71          lavender         red
chocolate4      gray72          lavenderblush    red1
coral           gray73          lavenderblush1   red2
coral1          gray74          lavenderblush2   red3
coral2          gray75          lavenderblush3   red4
coral3          gray76          lavenderblush4   rosybrown
coral4          gray77          lawngreen        rosybrown1
cornflowerblue  gray78          lemonchiffon     rosybrown2
cornsilk        gray79          lemonchiffon1    rosybrown3
cornsilk1       gray80          lemonchiffon2    rosybrown4
cornsilk2       gray81          lemonchiffon3    royalblue
cornsilk3       gray82          lemonchiffon4    royalblue1
cornsilk4       gray83          lightblue        royalblue2
crimson         gray84          lightblue1       royalblue3
cyan            gray85          lightblue2       royalblue4
cyan1           gray86          lightblue3       saddlebrown
cyan2           gray87          lightblue4       salmon
cyan3           gray88          lightcoral       salmon1
cyan4           gray89          lightcyan        salmon2
darkgoldenrod   gray90          lightcyan1       salmon3
darkgoldenrod1  gray91          lightcyan2       salmon4
darkgoldenrod2  gray92          lightcyan3       sandybrown
darkgoldenrod3  gray93          lightcyan4       seagreen
darkgoldenrod4  gray94          lightgoldenrod   seagreen1
darkgreen       gray95          lightgoldenrod1  seagreen2
darkkhaki       gray96          lightgoldenrod2  seagreen3
darkolivegreen  gray97          lightgoldenrod3  seagreen4
darkolivegreen1 gray98          lightgoldenrod4  seashell
darkolivegreen2 gray99          lightgoldenrodyellow seashell1
darkolivegreen3 gray100         lightgray        seashell2
darkolivegreen4 green           lightgrey        seashell3
darkorange      green1          lightpink        seashell4
darkorange1     green2          lightpink1       sienna
```

```
darkorange2      green3           lightpink2       sienna1
darkorange3      green4           lightpink3       sienna2
darkorange4      greenyellow      lightpink4       sienna3
darkorchid       grey             lightsalmon      sienna4
darkorchid1      grey0            lightsalmon1     skyblue
darkorchid2      grey1            lightsalmon2     skyblue1
darkorchid3      grey2            lightsalmon3     skyblue2
darkorchid4      grey3            lightsalmon4     skyblue3
darksalmon       grey4            lightseagreen    skyblue4
darkseagreen     grey5            lightskyblue     slateblue
darkseagreen1    grey6            lightskyblue1    slateblue1
darkseagreen2    grey7            lightskyblue2    slateblue2
darkseagreen3    grey8            lightskyblue3    slateblue3
darkseagreen4    grey9            lightskyblue4    slateblue4
darkslateblue    grey10           lightslateblue   slategray
darkslategray    grey11           lightslategray   slategray1
darkslategray1   grey12           lightslategrey   slategray2
darkslategray2   grey13           lightsteelblue   slategray3
darkslategray3   grey14           lightsteelblue1  slategray4
darkslategray4   grey15           lightsteelblue2  slategrey
darkslategrey    grey16           lightsteelblue3  snow
darkturquoise    grey17           lightsteelblue4  snow1
darkviolet       grey18           lightyellow      snow2
deeppink         grey19           lightyellow1     snow3
deeppink1        grey20           lightyellow2     snow4
deeppink2        grey21           lightyellow3     springgreen
deeppink3        grey22           lightyellow4     springgreen1
deeppink4        grey23           limegreen        springgreen2
deepskyblue      grey24           linen            springgreen3
deepskyblue1     grey25           magenta          springgreen4
deepskyblue2     grey26           magenta1         steelblue
deepskyblue3     grey27           magenta2         steelblue1
deepskyblue4     grey28           magenta3         steelblue2
dimgray          grey29           magenta4         steelblue3
dimgrey          grey30           maroon           steelblue4
dodgerblue       grey31           maroon1          tan
dodgerblue1      grey32           maroon2          tan1
dodgerblue2      grey33           maroon3          tan2
dodgerblue3      grey34           maroon4          tan3
dodgerblue4      grey35           mediumaquamarine tan4
firebrick        grey36           mediumblue       thistle
firebrick1       grey37           mediumorchid     thistle1
firebrick2       grey38           mediumorchid1    thistle2
firebrick3       grey39           mediumorchid2    thistle3
firebrick4       grey40           mediumorchid3    thistle4
floralwhite      grey41           mediumorchid4    tomato
forestgreen      grey42           mediumpurple     tomato1
gainsboro        grey43           mediumpurple1    tomato2
ghostwhite       grey44           mediumpurple2    tomato3
gold             grey45           mediumpurple3    tomato4
gold1            grey46           mediumpurple4    transparent
gold2            grey47           mediumseagreen   turquoise
gold3            grey48           mediumslateblue  turquoise1
gold4            grey49           mediumspringgreen turquoise2
```

```
goldenrod         grey50          mediumturquoise   turquoise3
goldenrod1        grey51          mediumvioletred   turquoise4
goldenrod2        grey52          midnightblue      violet
goldenrod3        grey53          mintcream         violetred
goldenrod4        grey54          mistyrose         violetred1
gray              grey55          mistyrose1        violetred2
gray0             grey56          mistyrose2        violetred3
gray1             grey57          mistyrose3        violetred4
gray2             grey58          mistyrose4        wheat
gray3             grey59          moccasin          wheat1
gray4             grey60          navajowhite       wheat2
gray5             grey61          navajowhite1      wheat3
gray6             grey62          navajowhite2      wheat4
gray7             grey63          navajowhite3      white
gray8             grey64          navajowhite4      whitesmoke
gray9             grey65          navy              yellow
gray10            grey66          navyblue          yellow1
gray11            grey67          oldlace           yellow2
gray12            grey68          olivedrab         yellow3
gray13            grey69          olivedrab1        yellow4
gray14            grey70          olivedrab2        yellowgreen
gray15            grey71          olivedrab3
gray16            grey72          olivedrab4
gray17            grey73          orange
```