About ncurses Colors | Linux Journal

by Jim Hall

Why does ncurses support only eight colors?

If you've looked into the color palette available in curses, you may wonder why curses supports only eight colors. The curses.h include file defines these color macros:

```
COLOR_BLACK
COLOR_RED
COLOR_GREEN
COLOR_YELLOW
COLOR_BLUE
COLOR_MAGENTA
COLOR_CYAN
COLOR_WHITE
```

But why only eight colors, and why these particular colors? At least with the Linux console, if you're running on a PC, the color range's origins are with the PC hardware.

A Brief History of Color

Linux started as a PC operating system, so the first Linux console was a PC running in text mode. And to understand the color palette on the PC console, you need to go all the way back to the old CGA days. In text mode, the PC terminal had a color palette of 16 colors, enumerated 0 (black) to 15 (white). Backgrounds were limited to the first eight colors:

- 0. Black
- 1. Blue
- 2. Green
- 3. Cyan
- 4. Red
- 5. Magenta
- 6. Brown
- 7. White ("Light Gray")
- 8. Bright Black ("Gray")
- 9. Bright Blue
- 10. Bright Green
- 11. Bright Cyan
- 12. Bright Red
- 13. Bright Magenta
- 14. Yellow
- 15. Bright White

These colors go back to CGA, IBM's Color/Graphics Adapter from the earlier PC-compatible computers. This was a step up from the plain monochrome displays; as the name implies, monochrome could display only black or white. CGA could display a limited range of colors.

CGA supports mixing red (R), green (G) and blue (B) colors. In its simplest form, RGB is either "on" or "off". In this case, you can mix the RGB colors in 2x2x2=8 ways. Table 1 shows the binary and decimal representations of RGB.

Table 1. Binary and Decimal Representations of RGB

000	(0)	Black
001	(1)	Blue
010	(2)	Green
011	(3)	Cyan
100	(4)	Red
101	(5)	Magenta
110	(6)	Yellow
111	(7)	White

To double the number of colors, CGA added an extra bit called the "intensifier" bit. With the intensifier bit set, the red, green and blue colors would be set to their maximum values. Without the intensifier bit, each RGB value would be set to a "midrange" intensity. Let's represent that intensifier bit as an extra 1 or 0 in the binary color representation, as iRGB (Table 2).

Table 2. Using the Intensifier Bit

0000	(0)	Black
0001	(1)	Blue
0010	(2)	Green
0011	(3)	Cyan
0100	(4)	Red
0101	(5)	Magenta
0110	(6)	Yellow
0111	(7)	White
1000	(8)	Bright Black
1001	(9)	Bright Blue
1010	(10)	Bright Green
1011	(11)	Bright Cyan
1100	(12)	Bright Red
1101	(13)	Bright Magenta
1110	(14)	Bright Yellow
1111	(15)	Bright White

But there's a problem: 0000 Black and 1000 Black are the same color. There's no red, green or blue color to intensify, so black is black whether or not the "intensifier" bit is set. To get around this limitation, CGA actually implemented a modified iRGB definition, using two intermediate values, at about one-third and two-thirds intensity. Most "normal" mode (0 to 7) colors used values at the two-thirds intensity, with the exception of yellow, which was assigned a one-third green value that turned the color brown or orange. To translate from "normal" mode to "bright" mode, convert zero values to the one-third intensity and two-thirds values to full intensity.

Table 3 shows another iteration of the color table, using 0x0 to 0xF for the color range on each RGB value, with 0x5 and 0xA as the one-third and two-thirds intensities, respectively.

Table 3. Color Table Using 0x0 to 0xF for the Color Range on Each RGB Value with 0x5 and 0xA as the One-Third and Two-Thirds Intensities, Respectively

0000	(#000)	Black
0001	(#00A)	Blue
0010	(#0A0)	Green
0011	(#0AA)	Cyan

0100 (#A00)	Red
0101 (#A0A)	Magenta
0110 (#A50)	Brown
0111 (#AAA)	White
1000 (#555)	Bright Black
1001 (#55F)	Bright Blue
1010 (#5F5)	Bright Green
1011 (#5FF)	Bright Cyan
1100 (#F55)	Bright Red
1101 (#F5F)	Bright Magenta
1110 (#FF5)	Bright Yellow
111 (#FFF)	Bright White

You may wonder why there are only eight background colors. Note that DOS also supported a "Blink" attribute. With this attribute set, your text could blink on and off. The "Blink" bit was encoded at the end of the foreground and background bit-pattern:

```
Bbbbffff
```

That's a full byte! Counting from right to left: four bits to represent the text foreground color (0000 Black to 1111 Bright White), three bits to code the background color (000 Black to 111 White) and one bit for the "Blink" attribute.

And, that's how curses got 16 text colors: eight standard-intensity text colors and eight high-intensity text colors. On the Linux console, these are essentially the same colors used in old DOS systems. That's also why you'll often see "brown" labeled "yellow" in some old DOS programmer references, because at least on DOS systems, it started out as plain "yellow" before the intensifier bit. Similarly, you also may see "gray" represented as "Bright Black", because "gray" is really "black" with the intensifier bit set.

Sample Program

Let me demonstrate the Linux terminal colors with a simple program. This color demo will iterate through all available color combinations using curses.

First, I need a simple function to create all possible color pairs:

```
void init_colorpairs(void)
{
    int fg, bg;
    int colorpair;

    for (bg = 0; bg <= 7; bg++) {
        for (fg = 0; fg <= 7; fg++) {
            colorpair = colornum(fg, bg);
            init_pair(colorpair, curs_color(fg), curs_color(bg));
        }
    }
}
```

The `init_colorpairs()` function also relies on a "translation" function that converts standard-intensity CGA color numbers (0 to 7) to curses color numbers, using the curses constant names like `COLOR_BLUE` or `COLOR_RED`:

```
short curs_color(int fg)
{
    switch (7 & fg) {           /* RGB */
    case 0:                     /* 000 */
        return (COLOR_BLACK);
    case 1:                     /* 001 */
        return (COLOR_BLUE);
    case 2:                     /* 010 */
        return (COLOR_GREEN);
    case 3:                     /* 011 */
        return (COLOR_CYAN);
    case 4:                     /* 100 */
        return (COLOR_RED);
    case 5:                     /* 101 */
        return (COLOR_MAGENTA);
    case 6:                     /* 110 */
        return (COLOR_YELLOW);
    case 7:                     /* 111 */
        return (COLOR_WHITE);
    }
}
```

To create a predictable color pair number for each foreground and background color, I also need a function `colornum()` to set an integer bit pattern based on the classic color byte:

```
int colornum(int fg, int bg)
{
    int B, bbb, ffff;

    B = 1 << 7;
    bbb = (7 & bg) << 4;
    ffff = 7 & fg;

    return (B | bbb | ffff);
}
```

The B bit that usually indicates blinking text is not used in my color demo program, so I always set B to one to guarantee that color pair 0 is never assigned. In curses, color pair 0 is reserved for the default foreground and background colors. That should be white text on a black background, but to be safe, I'll always define my own combination for white on black. For a foreground color 7 (white, binary 111) with background color 0 (black, binary 000), the bit pattern looks like this:

```
10000111
```

This is a decimal value of 135.

After `init_colorpairs()`, my program can set each color combination using a wrapper to the curses function `COLOR_PAIR()`. My wrapper function also turns bold text on or off, using the `A_BOLD` attribute:

```
void setcolor(int fg, int bg)
{
    /* set the color pair (colornum) and bold/bright (A_BOLD) */

    attron(COLOR_PAIR(colornum(fg, bg)));
    if (is_bold(fg)) {
        attron(A_BOLD);
    }
}

void unsetcolor(int fg, int bg)
{
    /* unset the color pair (colornum) and
       bold/bright (A_BOLD) */

    attroff(COLOR_PAIR(colornum(fg, bg)));
    if (is_bold(fg)) {
        attroff(A_BOLD);
    }
}
```

And the `is_bold()` function simply tests if the "intensifier" bit on the iRGB value (foreground colors 8 to 15) is set, using a simple bit mask:

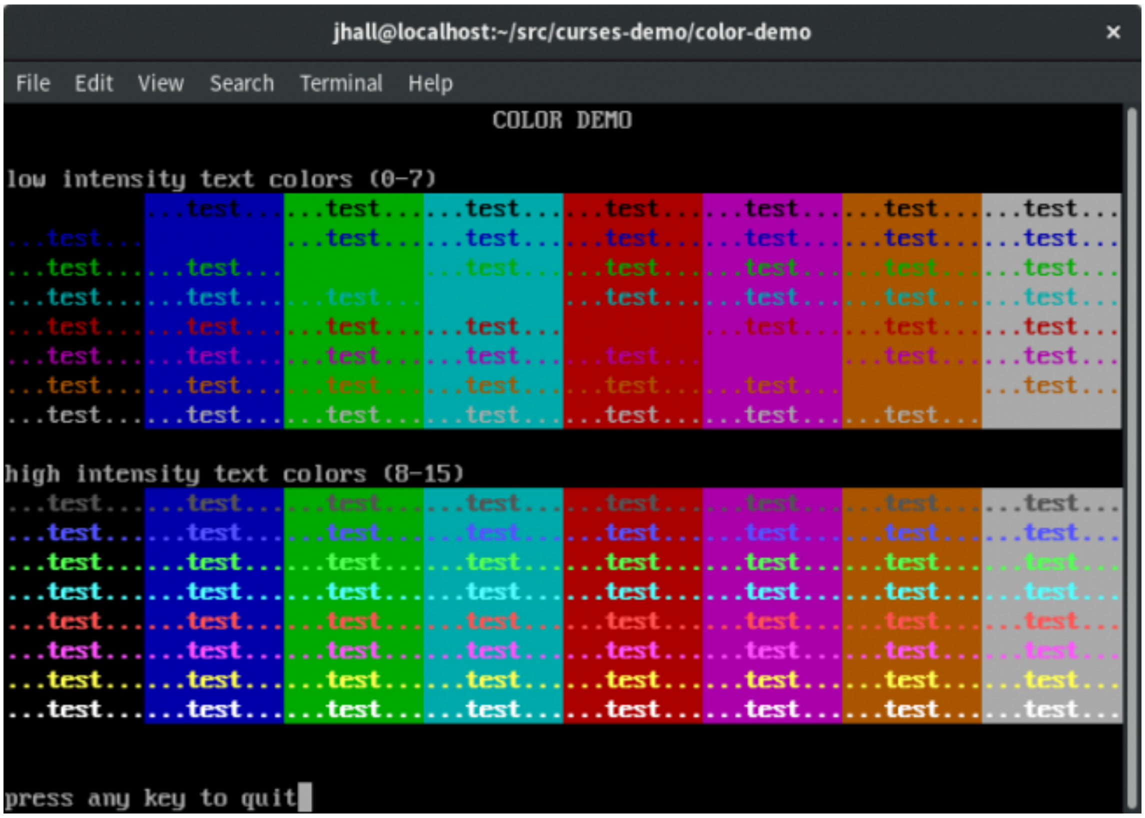
```
int is_bold(int fg)
{
    /* return the intensity bit */

    int i;

    i = 1 << 3;
    return (i & fg);
}
```

With that, creating the color demonstration program is easy:

```
/* color-demo.c */

#include <curses.h>
#include <stdio.h>
#include <stdlib.h>

int is_bold(int fg);
void init_colorpairs(void);
short curs_color(int fg);
int colornum(int fg, int bg);
void setcolor(int fg, int bg);
void unsetcolor(int fg, int bg);

int main(void)
{
    int fg, bg;

    /* initialize curses */

    initscr();
    keypad(stdscr, TRUE);
    cbreak();
    noecho();

    /* initialize colors */

    if (has_colors() == FALSE) {
        endwin();
        puts("Your terminal does not support color");
        exit(1);
    }
    start_color();
    init_colorpairs();

    /* draw test pattern */

    if ((LINES < 24) || (COLS < 80)) {
        endwin();
        puts("Your terminal needs to be at least 80x24");
        exit(2);
    }

    mvaddstr(0, 35, "COLOR DEMO");
    mvaddstr(2, 0, "low intensity text colors (0-7)");
    mvaddstr(12, 0, "high intensity text colors (8-15)");

    for (bg = 0; bg <= 7; bg++) {
        for (fg = 0; fg <= 7; fg++) {
            setcolor(fg, bg);
            mvaddstr(fg + 3, bg * 10, "...test...");
            unsetcolor(fg, bg);
        }

        for (fg = 8; fg <= 15; fg++) {
            setcolor(fg, bg);
            mvaddstr(fg + 5, bg * 10, "...test...");
            unsetcolor(fg, bg);
        }
    }

    mvaddstr(LINES - 1, 0, "press any key to quit");

    refresh();

    getch();
    endwin();

    exit(0);
}
```

Sample Output

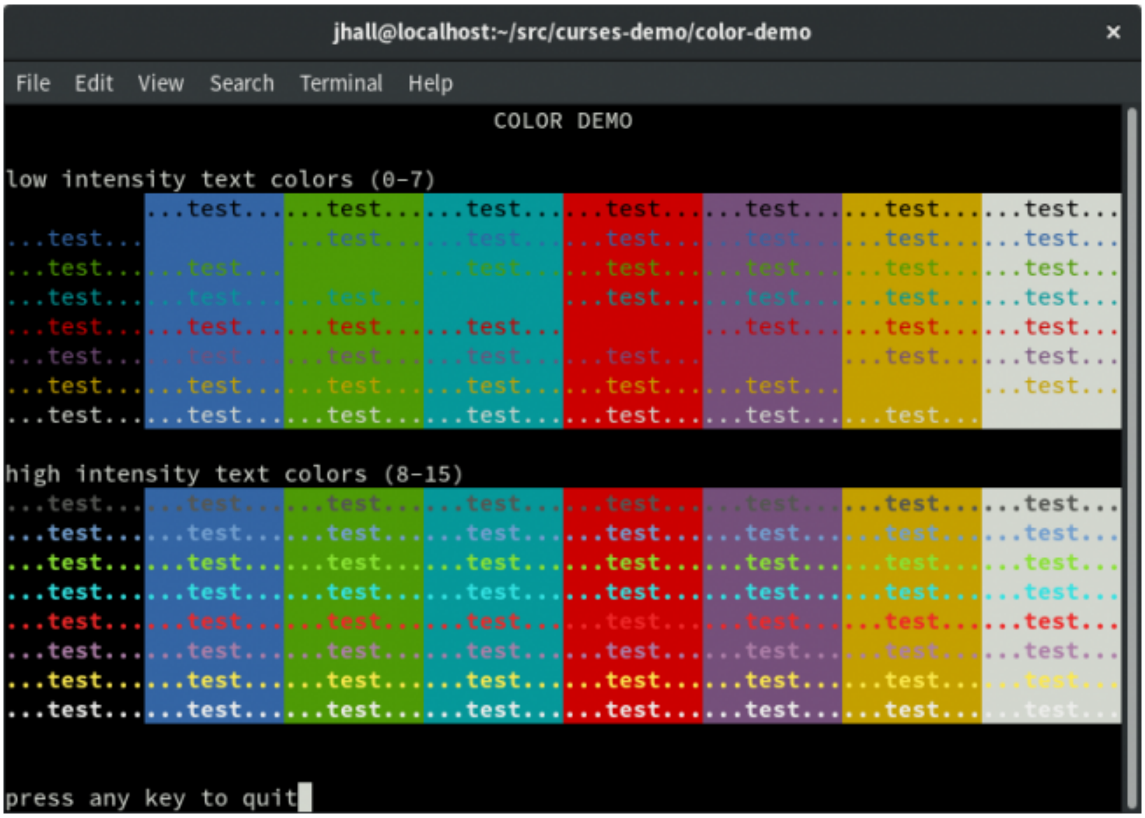
When you run the program, you see all combinations of 16 text colors and eight background colors, for a total of 16x8=128 different color pairs.


jhall@localhost:~/src/curses-demo/color-demo
File Edit View Search Terminal Help

```
                                 COLOR DEMO

low intensity text colors (0-7)
          ...test......test......test......test......test......test......test...
...test...          ...test......test......test......test......test......test...
...test......test...          ...test......test......test......test......test...
...test......test......test...          ...test......test......test......test...
...test......test......test......test...          ...test......test......test...
...test......test......test......test......test...          ...test......test...
...test......test......test......test......test......test...          ...test...
...test......test......test......test......test......test......test...

high intensity text colors (8-15)
...test......test......test......test......test......test......test......test...
...test......test......test......test......test......test......test......test...
...test......test......test......test......test......test......test......test...
...test......test......test......test......test......test......test......test...
...test......test......test......test......test......test......test......test...
...test......test......test......test......test......test......test......test...
...test......test......test......test......test......test......test......test...
...test......test......test......test......test......test......test......test...


press any key to quit
```

Figure 1. Color Demo Console

Figure 1 shows how I've set up my graphics terminal to reflect the text-mode terminal, including the standard text colors. Graphical terminal programs (like GNOME Terminal) support a wide range of colors, because they can leverage the available color palette of the X Window System. Note that you can change the available colors in these programs. Most colors are pretty close to their console counterparts, but some colors look quite different. For example, the default color palette for GNOME Terminal replaces the DOS brown with a yellow color (Figure 2).



```
                              COLOR DEMO

low intensity text colors (0-7)
          ...test......test......test......test......test......test......test...
...test...          ...test......test......test......test......test......test...
...test......test...          ...test......test......test......test......test...
...test......test......test...          ...test......test......test......test...
...test......test......test......test...          ...test......test......test...
...test......test......test......test......test...          ...test......test...
...test......test......test......test......test......test...          ...test...
...test......test......test......test......test......test......test...          

high intensity text colors (8-15)
...test......test......test......test......test......test......test......test...
...test......test......test......test......test......test......test......test...
...test......test......test......test......test......test......test......test...
...test......test......test......test......test......test......test......test...
...test......test......test......test......test......test......test......test...
...test......test......test......test......test......test......test......test...
...test......test......test......test......test......test......test......test...
...test......test......test......test......test......test......test......test...


press any key to quit
```

Figure 2. Color Demo GNOME Terminal

Through colors, you can represent information more clearly. This color demonstration simply iterates through all color combinations to show how each color looks with every other color.

Of course, this example is just color. You can do so much more with curses, depending on what you need your program to do. In a follow-up article, I'll demonstrate other features of the ncurses library, such as how to create windows and frames.

Resources

- Pradeep Padala's NCURSES Programming HOWTO at the Linux Documentation Project
- "Getting Started with ncurses" by Jim Hall, *LJ*, March 2018
- "Creating an Adventure Game in the Terminal with ncurses" by Jim Hall, *LJ*, April 2018
- "Programming in Color with ncurses" by Jim Hall, *LJ*, May 2018