Getting Started with Nextcloud

Review: the Librem 13v2

A Look at GDPR's Massive Impact

LINUX JOURNAL

Since 1994: The ux community

PRIVACY

HOW TO PROTECT YOUR DATA

EFFECTIVE PRIVACY PLUGINS

GIVE YOUR SERVERS SOME PRIVACY WITH TOR HIDDEN SERVICES

INTERVIEW: PRIVATE INTERNET ACCESS GOES OPEN SOURCE

ISSUE 286 | MAY 2018
www.linuxjournal.com

Programming in Color with ncurses

Jim demonstrates color manipulation with `curses` by adding colors to his terminal adventure game.

By Jim Hall

In parts one and two of my article series about programming with the `ncurses` library, I introduced a few `curses` functions to draw text on the screen, query characters from the screen and read from the keyboard. To demonstrate several of these functions, I created a simple adventure game in `curses` that drew a game map and player character using simple characters. In this follow-up article, I show how to add color to a `curses` program.

Drawing on the screen is all very well and good, but if it's all white-on-black text, your program might seem dull. Colors can help convey more information—for example, if your program needs to indicate success or failure. In such a case, you could display text in green or red to help emphasize the outcome. Or, maybe you simply want to use colors to "snazz" up your program to make it look prettier.

In this article, I use a simple example to demonstrate color manipulation via the `curses` functions. In my previous article, I wrote a basic adventure-style game that lets you move a player character around a crudely drawn map. However, the map was entirely black and white text, relying on shapes to suggest water (~) or mountains (^), so let's update the game to use colors.

Color Essentials

Before you can use colors, your program needs to know if it can rely on the terminal to display the colors correctly. On modern systems, this always should be true. But

in the classic days of computing, some terminals were monochromatic, such as the venerable VT52 and VT100 terminals, usually providing white-on-black or green-on-black text.

To query the terminal capability for colors, use the `has_colors()` function. This will return a true value if the terminal can display color, and a false value if not. It is usually used to start an `if` block, like this:

```
if (has_colors() == FALSE) {
    endwin();
    printf("Your terminal does not support color\n");
    exit(1);
}
```

Having determined that the terminal can display color, you then can set up `curses` to use colors with the `start_color()` function. Now you're ready to define the colors your program will use.

In `curses`, you define colors in pairs: a foreground color on a background color. This allows `curses` to set both color attributes at once, which often is what you want to do. To establish a color pair, use `init_pair()` to define a foreground and background color, and associate it to an index number. The general syntax is:

```
init_pair(index, foreground, background);
```

Consoles support only eight basic colors: black, red, green, yellow, blue, magenta, cyan and white. These colors are defined for you with the following names:

- `COLOR_BLACK`
- `COLOR_RED`
- `COLOR_GREEN`
- `COLOR_YELLOW`
- `COLOR_BLUE`

- COLOR_MAGENTA
- COLOR_CYAN
- COLOR_WHITE

Applying the Colors

In my adventure game, I'd like the grassy areas to be green and the player's "trail" to be a subtle yellow-on-green dotted path. Water should be blue, with the tildes in the similar cyan color. I'd like mountains to be grey, but black text on a white background should make for a reasonable compromise. To make the player's character more visible, I'd like to use a garish red-on-magenta scheme. I can define these color pairs like so:

```
start_color();
init_pair(1, COLOR_YELLOW, COLOR_GREEN);
init_pair(2, COLOR_CYAN, COLOR_BLUE);
init_pair(3, COLOR_BLACK, COLOR_WHITE);
init_pair(4, COLOR_RED, COLOR_MAGENTA);
```

To make my color pairs easy to remember, my program defines a few symbolic constants:

```
#define GRASS_PAIR     1
#define EMPTY_PAIR     1
#define WATER_PAIR     2
#define MOUNTAIN_PAIR  3
#define PLAYER_PAIR    4
```

With these constants, my color definitions become:

```
start_color();
init_pair(GRASS_PAIR, COLOR_YELLOW, COLOR_GREEN);
init_pair(WATER_PAIR, COLOR_CYAN, COLOR_BLUE);
```

```
init_pair(MOUNTAIN_PAIR, COLOR_BLACK, COLOR_WHITE);
init_pair(PLAYER_PAIR, COLOR_RED, COLOR_MAGENTA);
```

Whenever you want to display text using a color, you just need to tell `curses` to set that color attribute. For good programming practice, you also should tell `curses` to undo the color combination when you're done using the colors. To set the color, use `attron()` before calling functions like `mvaddch()`, and then turn off the color attributes with `attroff()` afterward. For example, when I draw the player's character, I might do this:

```
attron(COLOR_PAIR(PLAYER_PAIR));
mvaddch(y, x, PLAYER);
attroff(COLOR_PAIR(PLAYER_PAIR));
```

Note that applying colors to your programs adds a subtle change to how you query the screen. Normally, the value returned by `mvinch()` is of type `chtype`. Without color attributes, this is basically an integer and can be used as such. But, colors add extra attributes to the characters on the screen, so `chtype` carries extra color information in an extended bit pattern. If you use `mvinch()`, the returned value will contain this extra color value. To extract just the "text" value, such as in the `is_move_okay()` function, you need to apply a bitwise & with the `A_CHARTEXT` bit mask:

```
int is_move_okay(int y, int x)
{
    int testch;

    /* return true if the space is okay to move into */

    testch = mvinch(y, x);
    return (((testch & A_CHARTEXT) == GRASS)
            || ((testch & A_CHARTEXT) == EMPTY));
}
```

With these changes, I can update the adventure game to use colors:

```
/* quest.c */

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

#define GRASS     ' '
#define EMPTY     '.'
#define WATER     '~'
#define MOUNTAIN  '^'
#define PLAYER    '*'

#define GRASS_PAIR     1
#define EMPTY_PAIR     1
#define WATER_PAIR     2
#define MOUNTAIN_PAIR  3
#define PLAYER_PAIR    4

int is_move_okay(int y, int x);
void draw_map(void);

int main(void)
{
    int y, x;
    int ch;

    /* initialize curses */

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

```
/* initialize colors */

if (has_colors() == FALSE) {
    endwin();
    printf("Your terminal does not support color\n");
    exit(1);
}

start_color();
init_pair(GRASS_PAIR, COLOR_YELLOW, COLOR_GREEN);
init_pair(WATER_PAIR, COLOR_CYAN, COLOR_BLUE);
init_pair(MOUNTAIN_PAIR, COLOR_BLACK, COLOR_WHITE);
init_pair(PLAYER_PAIR, COLOR_RED, COLOR_MAGENTA);

clear();

/* initialize the quest map */

draw_map();

/* start player at lower-left */

y = LINES - 1;
x = 0;

do {

    /* by default, you get a blinking cursor - use it to
       indicate player * */

    attron(COLOR_PAIR(PLAYER_PAIR));
    mvaddch(y, x, PLAYER);
```

```
attroff(COLOR_PAIR(PLAYER_PAIR));
move(y, x);
refresh();

ch = getch();

/* test inputted key and determine direction */

switch (ch) {
case KEY_UP:
case 'w':
case 'W':
    if ((y > 0) && is_move_okay(y - 1, x)) {
        attron(COLOR_PAIR(EMPTY_PAIR));
        mvaddch(y, x, EMPTY);
        attroff(COLOR_PAIR(EMPTY_PAIR));
        y = y - 1;
    }
    break;
case KEY_DOWN:
case 's':
case 'S':
    if ((y < LINES - 1) && is_move_okay(y + 1, x)) {
        attron(COLOR_PAIR(EMPTY_PAIR));
        mvaddch(y, x, EMPTY);
        attroff(COLOR_PAIR(EMPTY_PAIR));
        y = y + 1;
    }
    break;
case KEY_LEFT:
case 'a':
case 'A':
    if ((x > 0) && is_move_okay(y, x - 1)) {
```

```
                attron(COLOR_PAIR(EMPTY_PAIR));
                mvaddch(y, x, EMPTY);
                attroff(COLOR_PAIR(EMPTY_PAIR));
                x = x - 1;
            }
            break;
        case KEY_RIGHT:
        case 'd':
        case 'D':
            if ((x < COLS - 1) && is_move_okay(y, x + 1)) {
                attron(COLOR_PAIR(EMPTY_PAIR));
                mvaddch(y, x, EMPTY);
                attroff(COLOR_PAIR(EMPTY_PAIR));
                x = x + 1;
            }
            break;
        }
    }
    while ((ch != 'q') && (ch != 'Q'));

    endwin();

    exit(0);
}

int is_move_okay(int y, int x)
{
    int testch;

    /* return true if the space is okay to move into */

    testch = mvinch(y, x);
    return (((testch & A_CHARTEXT) == GRASS)
```

```
            || ((testch & A_CHARTEXT) == EMPTY));
}

void draw_map(void)
{
    int y, x;

    /* draw the quest map */

    /* background */

    attron(COLOR_PAIR(GRASS_PAIR));
    for (y = 0; y < LINES; y++) {
        mvhline(y, 0, GRASS, COLS);
    }
    attroff(COLOR_PAIR(GRASS_PAIR));

    /* mountains, and mountain path */

    attron(COLOR_PAIR(MOUNTAIN_PAIR));
    for (x = COLS / 2; x < COLS * 3 / 4; x++) {
        mvvline(0, x, MOUNTAIN, LINES);
    }
    attroff(COLOR_PAIR(MOUNTAIN_PAIR));

    attron(COLOR_PAIR(GRASS_PAIR));
    mvhline(LINES / 4, 0, GRASS, COLS);
    attroff(COLOR_PAIR(GRASS_PAIR));

    /* lake */

    attron(COLOR_PAIR(WATER_PAIR));
    for (y = 1; y < LINES / 2; y++) {
```

```
            mvhline(y, 1, WATER, COLS / 3);
        }
        attroff(COLOR_PAIR(WATER_PAIR));
}
```

Unless you have a keen eye, you may not be able to spot all of the changes necessary to support color in the adventure game. The `diff` tool shows all the instances where functions were added or code was changed to support colors:

```
$ diff quest-color/quest.c quest/quest.c
12,17d11
< #define GRASS_PAIR     1
< #define EMPTY_PAIR     1
< #define WATER_PAIR     2
< #define MOUNTAIN_PAIR  3
< #define PLAYER_PAIR    4
<
33,46d26
<     /* initialize colors */
<
<     if (has_colors() == FALSE) {
<    endwin();
<    printf("Your terminal does not support color\n");
<    exit(1);
<     }
<
<     start_color();
<     init_pair(GRASS_PAIR, COLOR_YELLOW, COLOR_GREEN);
<     init_pair(WATER_PAIR, COLOR_CYAN, COLOR_BLUE);
<     init_pair(MOUNTAIN_PAIR, COLOR_BLACK, COLOR_WHITE);
<     init_pair(PLAYER_PAIR, COLOR_RED, COLOR_MAGENTA);
<
61d40
```

```
<    attron(COLOR_PAIR(PLAYER_PAIR));
63d41
<    attroff(COLOR_PAIR(PLAYER_PAIR));
76d53
<            attron(COLOR_PAIR(EMPTY_PAIR));
78d54
<            attroff(COLOR_PAIR(EMPTY_PAIR));
86d61
<            attron(COLOR_PAIR(EMPTY_PAIR));
88d62
<            attroff(COLOR_PAIR(EMPTY_PAIR));
96d69
<            attron(COLOR_PAIR(EMPTY_PAIR));
98d70
<            attroff(COLOR_PAIR(EMPTY_PAIR));
106d77
<            attron(COLOR_PAIR(EMPTY_PAIR));
108d78
<            attroff(COLOR_PAIR(EMPTY_PAIR));
128,129c98
<     return (((testch & A_CHARTEXT) == GRASS)
<        || ((testch & A_CHARTEXT) == EMPTY));
---
>     return ((testch == GRASS) || (testch == EMPTY));
140d108
<     attron(COLOR_PAIR(GRASS_PAIR));
144d111
<     attroff(COLOR_PAIR(GRASS_PAIR));
148d114
<     attron(COLOR_PAIR(MOUNTAIN_PAIR));
152d117
<     attroff(COLOR_PAIR(MOUNTAIN_PAIR));
154d118
```

```
<     attron(COLOR_PAIR(GRASS_PAIR));
156d119
<     attroff(COLOR_PAIR(GRASS_PAIR));
160d122
<     attron(COLOR_PAIR(WATER_PAIR));
164d125
<     attroff(COLOR_PAIR(WATER_PAIR));
```

Let's Play—Now in Color

The program now has a more pleasant color scheme, more closely matching the original tabletop gaming map, with green fields, blue lake and imposing gray mountains. The hero clearly stands out in red and magenta livery.

Figure 1. A Simple Tabletop Game Map, with a Lake and Mountains

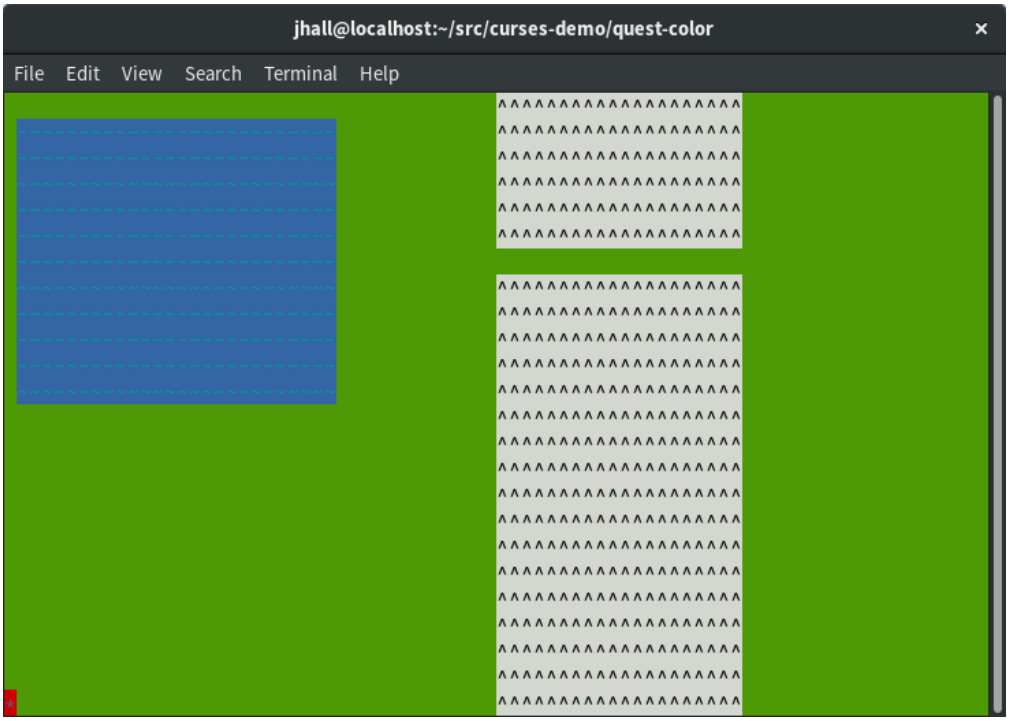

Figure 2. The player starts the game in the lower-left corner.

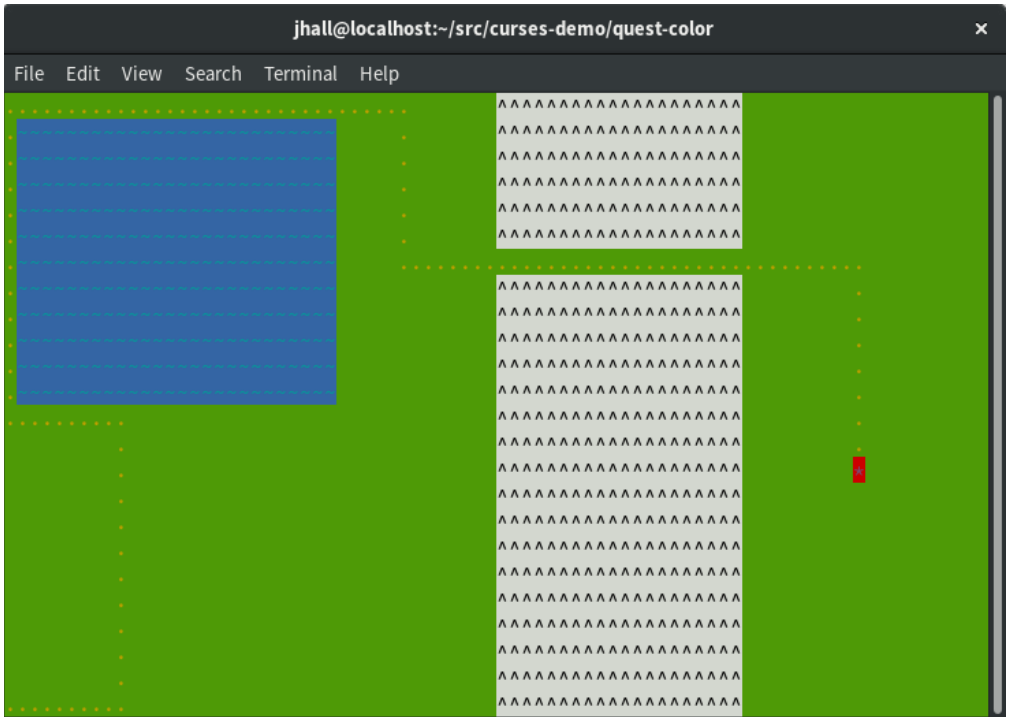

Figure 3. The player can move around the play area, such as around the lake, through the mountain pass and into unknown regions.

With colors, you can represent information more clearly. This simple example uses colors to indicate playable areas (green) versus impassable regions (blue or gray). I hope you

will use this example game as a starting point or reference for your own programs. You can do so much more with `curses`, depending on what you need your program to do.

In a follow-up article, I plan to demonstrate other features of the `ncurses` library, such as how to create windows and frames. In the meantime, if you are interested in learning more about `curses`, I encourage you to read Pradeep Padala's NCURSES Programming HOWTO, at the Linux Documentation Project. ■

Jim Hall is an advocate for free and open-source software, best known for his work on the FreeDOS Project, and he also focuses on the usability of open-source software. Jim is the Chief Information Officer at Ramsey County, Minnesota.