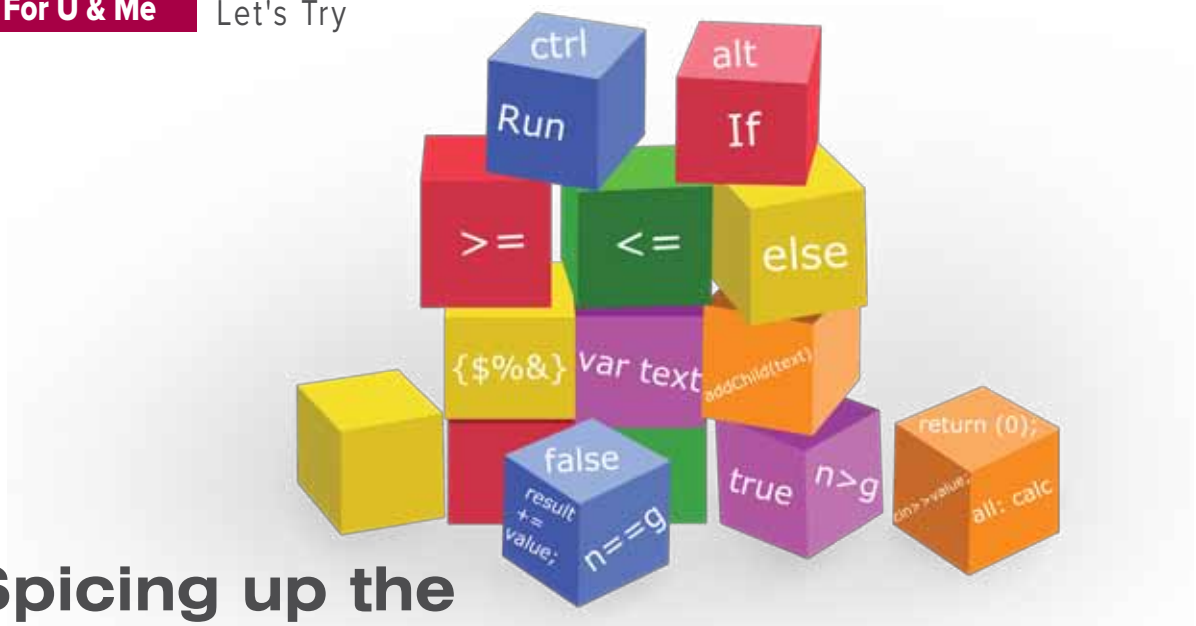

Spicing up the Console for Fun and Profit

Part—1

Command-line terminals and the shell have been an integral part of all *NIX systems from the beginning. The shell is the universal interface between a *NIX system and its users, and the terminal is the medium for it. This article is the first in a two-part series, and shows various ways to spice up different elements of the text console, like prompts, colours or themes to enhance your day-to-day interaction with *NIX systems.

There were different types of hardware terminal devices to interact with the shell in the earlier days of UNIX, but today, these have been mostly replaced by virtual consoles and other emulated terminals provided by various desktop environments. In fact, nowadays, the terminal and shell are just called the text console, mostly due to their mutually complementary relationship. Many different flavours of the shell as well as the terminal were developed in different flavours of *NIX systems. Some popular shells are *bash*, *sh*, *csh*, *ksh*, *zsh* and terminals *xterm*, Linux console, etc. The text console is minimalist, but powerful. Many types of *NIX systems like servers, embedded systems, legacy machines, etc, have neither the need nor the space for resource-hungry, complicated and unreliable graphics environments. Text consoles are the only way to interact with users. Users and developers are still quite dependent on command-line consoles, many decades after the introduction of the first UNIX system.

Over the years, *bash* has emerged as a standard shell on many *NIX systems, especially on GNU/Linux distributions, so the shell used in this article is *bash*. Most modern terminals on GNU/Linux systems are very similar in functionality, so we use the key words 'shell' and 'text console/terminal' interchangeably. Also, I use the word 'terminal' for either *xterm* or the standard GNU/Linux virtual console. I used Puppy Linux 5.1 and the Ubuntu 10.04 64-bit desktop edition to test code and generate the screen-shots presented here.

The default appearance of the text console is a full-screen virtual terminal in non-graphics mode, or an emulated terminal in graphics mode, with a boring default black background and white text, in most cases. However, the console is highly customisable, based on users' tastes and preferences. Let's find out how.

Tweaking *bash* prompts

Different flavours of GNU/Linux present different kinds of *bash* prompts; the default prompt on my Ubuntu text console and terminal emulator is *user@computername* followed by the current working directory, and *$* for a normal user or *#* for the super user. You can customise the content and appearance of your prompts to make them more useful and appealing. *Bash* provides the environmental variable PS1 to customise user prompts. In fact, there are also *PS2*, *PS3*, *PS4* and *PROMPT_COMMAND* environmental variables, relevant to various other aspects of prompt customisation. Just run *OPS1=$PS1; PS1="ItsMyPrompt :) "* in *bash* to quickly view your prompt change. Restore your old prompt with *PS1=$OPS1*.

PS2 determines the prompt shown for the continuation of long command lines. It is shown when you break a long command using the \ character, or press *Enter* without completing the shell construct. Try to type *sdemo="any*

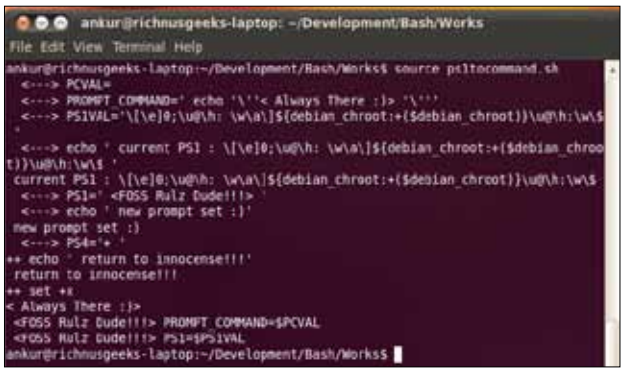

Figure 1: Bash variables to customise the prompt

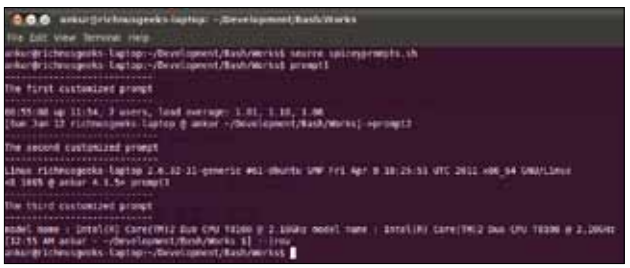

Figure 2: More customised *bash* prompts

number of demo characters and then press *Enter*, to see a continuation prompt (normally >) till you type the closing double quotes ("). You can change *PS2* to show something different, like we did for PS1.

PS3 and *PS4* customise the shell output in case of shell scripts; PS3 sets the prompt for the select command in shell scripts, and PS4 sets the value printed after the PS1 prompt when you enable script execution tracing using *set -x*.

Note: Source files accompanying this article are available for download from the LFY website at http://linuxforu.com/article_source_code/aug11/console.zip

Bash executes the command in *PROMPT_COMMAND* every time (before) it shows the PS1 prompt. You could use this to pre-process or show some information before the prompt. To see the things we've discussed in action, run the accompanying *ps1tocommand.sh* with *source ps1tocommand.sh*, or *. ps1tocommand.sh* (the dot is a short-cut for the source command). You have to source the script into your current shell because if you run it in a new shell (e.g., *sh ps1tocommand.sh*), the settings are changed only in that shell, are lost when the script finishes running, and do not affect the prompt displayed in the (parent) shell in your terminal. After sourcing this script, to reset to the previous prompt settings, run *PROMPT_COMMAND=$PCVAL; PS1=$PS1VAL*.

The output of this is shown in Figure 1. If you noticed, the original value of *PS1* contains code like *\u, \w, \$* etc. As per its man page, *bash* provides the backslash-escaped special characters in Table 1 to customise prompt strings.

\a	an ASCII bell character
\d	date in "Weekday Month Date" format
\h	hostname up to the first '.'
\H	hostname
\j	number of jobs currently managed by the shell
\l	basename of the shell's terminal device name
\s	basename of the shell
\t	current time in 24-hour HH:MM:SS format
\T	current time in 12-hour HH:MM:SS format
\@	current time in 12-hour am/pm format
\A	current time in 24-hour HH:MM format
\u	username of the current user
\v	version of *bash*
\V	version + patch level of *bash*
\w	current working directory
\W	basename of the current working directory
\!	history number of the command
\#	command number of the command
\$	a # for super user, otherwise a $

Table 1: PS1 special characters

You can use any quick-running command in prompt variables or scripts, besides these special characters. ('Quick-running', since the shell has to interpret these every time it displays the prompt.) Also, if you want to put a lot of text in your prompt, it's better to divide the content between *PROMPT_COMMAND* and *PS1*. Source into your *bash* shell the accompanying *spiceyprompts.sh*, then *run prompt1, prompt2, prompt3* to see some more prompts that combine various shell commands, escape characters, etc. Finally, run *rov* to restore the original prompt.

Figure 2 shows the output after doing this. All the examples used till now only affect the shell instance in which they are typed or invoked. To make settings 'permanent' after testing them out, add the commands to your *~/.bashrc* profile.

ANSI escape sequences and *bash* glorification

Now what about using colours in the shell? Almost all consoles, whether physical or terminal emulators, support various character combinations that control text formatting, colour and other aspects. These are known as ANSI escape sequences. We see these used in console-mode GUI applications like installers, text dialogue utilities, and tool-kits like *ncurses*, *newt*, etc. This section is a hands-on introduction to ANSI escape sequences to colourise the console. We will explore *ncurses* and *newt* in the second part of the series.

These escape sequences control various aspects of the text to be displayed, and also the text cursor and graphics modes. The most commonly used format of the terminal escape sequence is *\033[[text colour;][text background;][text attribute]m[text]\033[m*. A listing of the most common numbers to choose the text colour, background and attribute, is shown in Table 2. You can change only the setting you're

Figure 3: Colourful patterns through ANSI escape sequences

interested in, like the colour of the text, and leave the text background and attribute. Also, the order of the parameter values in escape sequences does not matter—you can put the value of the background before that of colour, or the attribute before background, etc.

Text attribute	Text colour	Text background
0 normal	30 black	40 black
1 bold	31 red	41 red
4 underscore	32 green	42 green
7 reverse video	33 yellow	43 yellow
	34 blue	44 blue
	35 magenta	45 magenta
	36 cyan	46 cyan
	37 white	47 white

Table 2: ANSI escape sequence values

There are a lot of escape sequences—check the links in the *References* section—but not everything is guaranteed to work with *bash* and the GNU/Linux terminal; besides, some sequences are non-portable and obsolete. I verified that the values in Table 2 work on all my GNU/Linux distributions. Run the accompanying *testes.py* with *python testes.py* and see how these simple-looking escape sequences turn your console into a colourful graffiti wall.

The output of the script is shown in Figure 3. You can combine these concepts to tweak your shell prompts and turn your otherwise boring shell into a beautiful environment limited only by your imagination. Run *OPS1=$PS1; PS1="\[\033[31;1m<\d \H @ \u \w> \033[m\]"* in a text console, as an example. Please note that the ending \033[m is required, to reset the attributes after the prompt, else your command text will have the same attributes. You can also do terminal cursor movements through escape sequences like those in Table 3. Run the accompanying *curtest.sh* to see some of these, along with colour escape sequences, in action. I'll show a cleaner version of the text console cursor movements in a later section on *tput*.

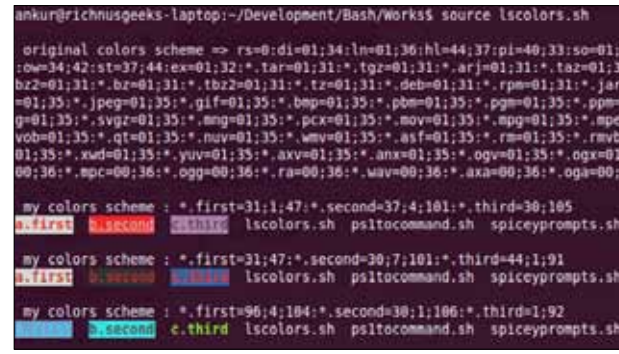

Figure 4: Directory listing with customised colours

Escape	Cursor movements
\033[<y>;<x>H	move cursor to x column, y line
\033[<n>A	move cursor n lines up
\033[<n>B	move cursor n line down
\033[<n>C	move cursor n columns forward
\033[<n>D	move cursor n columns backward
\033[2J	clear screen
\033[K	erase to end of line
\033[s	save cursor position
\033[u	restore cursor position

Table 3: Cursor movement sequences

Now let's play around with one more aspect before the next section. An environment variable dictates the colours of various entries when you run the *ls* command, with colour enabled in your terminal. Most GNU/Linux distributions have aliases set in *~/.bashrc* to enable listings in colour in *bash*, and if this does not work, check the *ls* man page to find out how to enable it. The variable is *LS_COLORS*, and you can tweak it very easily. Run *dircolors* to see the code that sets the listing colours for your shell. (If dircolors is not available, run *echo $LS_COLORS*.) The basic pattern in *LS_COLORS* is *type=[text colour;][text background;][text attribute]*. You can decode the contents of LS_COLOR based on information in Table 4, and any previous knowledge about escape sequence colour codes, with the additional colours in Table 5.

Content	Type
di	directory
fi	file
ln	symbolic link
pi	FIFO file
so	socket file
bd	block (buffered) special file
cd	character (unbuffered) special file
or	symbolic link pointing to a non-existent file (orphan)
mi	non-existent file referred by a symbolic link (visible when you type *ls -l*)
ex	executable file ('x' set in permission)
**.ext*	file with a particular extension (e.g., **.c, *.cpp, *.ogg etc*.)

Table 4: LS_COLORS settings

90	dark grey
91	light red
92	light green
93	yellow
94	light blue
95	light purple
96	turquoise
100	dark grey background
101	light red background
102	light green background
103	yellow background
104	light blue background
105	light purple background
106	turquoise background

Table 5: Additional colours

Now run the accompanying *lscolors.sh* (output shown in Figure 4) to see how to display listings of three files with some random extensions with customised colouring schemes. Again, once you find a colour scheme you are satisfied with, put the *LS_COLORS* setting in *~/.bashrc* to make it permanent.

You can explore ANSI escape sequences in detail at the links provided in the *References* section. Also, I encourage you to explore a package named *Bashish*, also in References. It is a theme environment for text terminals, and can change colours, fonts, transparency and background images on a per-application basis, according to its home page.

Console terminal manipulation with *tput*

Till now, we've used raw escape sequences, but now let's accomplish more tweaks with a utility called *tput*. This is a higher-level shell utility to control various aspects like text background, foreground, attributes, and to manipulate various cursor movements without issuing escape sequences. It simplifies escape-sequence operations, and provides a portable way to manipulate the capabilities of various console terminals. Table 6 has various *tput* arguments and their uses:

Argument	Capability
setab [0-7]	set background colour using ANSI escape value
setb [0-7]	set background colour
setaf [1-7]	set foreground colour using ANSI escape value
setf [1-7]	set foreground colour
bold	set bold mode
rev	set reverse mode
sgr0	turn off all attributes
cup y x	move cursor to x, y location (0, 0 is top left)
sc	save cursor position
rc	restore cursor
lines	get number of lines of the terminal
cols	get number of columns of the terminal
cub n	move n characters left
cuf n	move n characters right
clear	clear screen

Table 6: Tput arguments

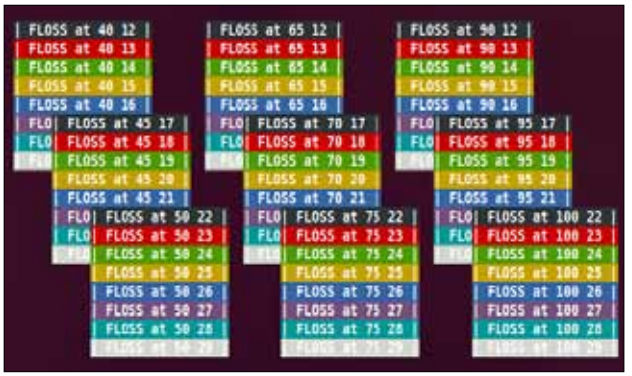

Figure 5: Overlapping rectangles formed by tput

The *setab* and *setaf* arguments support the same colour schemes as the ANSI escape sequences; the only difference is the index, ranging from 0-7. The colour index table for *setb* and *setf* is different, and is shown in Table 7.

Colour	Index
Black	0
Blue	1
Green	2
Cyan	3
Red	4
Magenta	5
Yellow	6
White	7

Table 7: Setb indexes

The output of the accompanying *tputops.py* (try it!) is shown in Figure 5. If you relate the appearance of overlapping rectangles to overlapping windows in the console graphics-mode applications that you often encounter in GNU/Linux, you are correct. Those applications work upon the basic concepts uncovered in this article. Explore the link for *tput* in References to create a full-fledged graphics menu-driven console application without any tool-kit.

See you next month, when we carry on! END

References

- ANSI escape code Wikipedia entry: *http://en.wikipedia.org/wiki/ANSI_escape_sequence*
- Bash Prompt HowTo: *http://www.gilesorr.com/bashprompt/howto/*
- Bashish theme environment: *http://bashish.sourceforge.net/*
- IBM DeveloperWorks *tput* article: *http://ibm.co/nVilHc*

By: Ankur Kumar Sharma

The author is a software developer and researcher. He likes to sing and play classic rock songs on the guitar, read self-help books, write, and explore all interesting things in his spare time. He blogs at *www.richnusgeeks.com.*