Using ANSI Color Codes to Colorize Your Bash Prompt on Linux

Author Michael
Published 2012, Nov 19, 08:34 am
Category Tech › Linux
Tags Bash, CLI, Linux, PS1, Shell

So, why do it? Colorizing your Bash prompt not only gives you super geek cred, it also has the potential to boost your productivity and put a smile on your face. Take a look.

This should also work with the Mac shell, but I don't know for sure. Leave a comment if it works.

---

update: various corrections, added "Checking State" and "256 (8-bit) Colors" sections.

The goal is to create a design similar to the examples below, but with more artistic merit. To accomplish this, we must understand several things, including: special characters, some bash scripting, and ANSI colors.

Box-

```
╔═ * ═ 12:24 PM Tue Jan 01 ═ 0 ═ bdjnk@PowerMAP ═ /usr/share/icons ═ 16 files, 460K ═╗
╚═» badcommand
bash: badcommand: command not found

╔═ * ═ 12:24 PM Tue Jan 01 ═ 0 ═ bdjnk@PowerMAP ═ /usr/share/icons ═ 16 files, 460K ═╗
╚═» gvim &
[1] 30109

╔═ * ═ 12:24 PM Tue Jan 01 ═ 1 ═ bdjnk@PowerMAP ═ /usr/share/icons ═ 16 files, 460K ═╗
╚═» su -
Password:

╔═ * ═ 12:24 PM Tue Jan 01 ═ 0 ═ root@PowerMAP ═ ~ ═ 0 files, 68K ═╗
╚═» exit
logout
[1]+  Done                    gvim

╔═ * ═ 12:25 PM Tue Jan 01 ═ 0 ═ bdjnk@PowerMAP ═ /usr/share/icons ═ 16 files, 460K ═╗
╚═»
```

Drawing Characters

Unicode provides an extensive range of symbols and special characters we can use to make our Bash prompt awesome. In my case, I am using the special box drawing characters to add cutesy little arrows and double lines to the first example.

```
* 12:27 PM Tue Jan 01 0 bdjnk@PowerMAP ~ 17 files, 492K
> badcommand
bash: badcommand: command not found

* 12:27 PM Tue Jan 01 0 bdjnk@PowerMAP ~ 17 files, 492K
> gvim &
[1] 30372

* 12:27 PM Tue Jan 01 1 bdjnk@PowerMAP ~ 17 files, 492K
> su -
Password:

* 12:27 PM Tue Jan 01 0 root@PowerMAP ~ 0 files, 68K
> exit
logout
[1]+  Done                    gvim

* 12:27 PM Tue Jan 01 0 bdjnk@PowerMAP ~ 17 files, 492K
>
```

The Bash PS1 Variable

All of the content and the appearance of the Bash prompt is managed by the PS1 variable, which is defined in ~/.bashrc for each user, and in /etc/*bashrc for users without their own definition.

While the PS1 variable can become incredibly complex, it can also be very simple. Here is the default that came with my Arch Linux install:

```
PS1='[u@h W]$ '
```

This creates a simple prompt which displays:

```
[bdjnk@PowerMAP icons]$
```

Let's discuss how that works.

Some PS1 Codes

These are generally called Bash prompt "escape sequences", as they consist of a backslash (the standard escape character) followed by a regular character. Although there are many available, I will just list the one's I've used:

`u` - the current user's username
`h` - the current machine's hostname
`j` - the number of jobs managed by this shell
`@` - the current time (12 hour format)
`d` - the current date
`w` - path of the current working directory
`W` - just the current working directory
`e` - an ASCII escape character (033)
`n` - adds a newline character

Summary

We can now clearly see how `'[u@h W]$ '` becomes `[bdjnk@PowerMAP icons]$` because it makes use of some basic PS1 codes plus some plain text. No problem.

Checking State

There are two states I want to track. The exit status of the previously run command, and whether or not I'm logged in as root. You can see from the examples that the PS1 is aware of these circumstances. Here's how:

Exit Status

In Linux, an exit code of anything other than 0 generally indicates some kind of error. To get the exit code, we need to take a look at the `$?` variable. Try running various commands, then running `echo $?` right after from the same prompt. This should be 0 if the command completed successfully, and something other than 0 if it didn't.

For our purposes, we need to test the exit status. We can use the form `if [ $? -eq 0 ]; then echo 'good'; else echo 'bad'; fi` replacing the echo commands with whatever we want to do in each case.

Root Prompt

In Linux, the root user receives the user id 0. The current user id for a given prompt is stored in the variable `$EUID`. Thus we test for root with `if [ ${EUID} -eq 0 ]; then echo 'root'; else echo 'user'; fi`.

Gathering Data

Linux is blessed with a plethora of fast little command line utilities, perfect for gathering data to use in your Bash prompt. You'll notice I use the full path of each command. This allows the system to use the utility without knowing anything about the PATH variable.

Number of Files

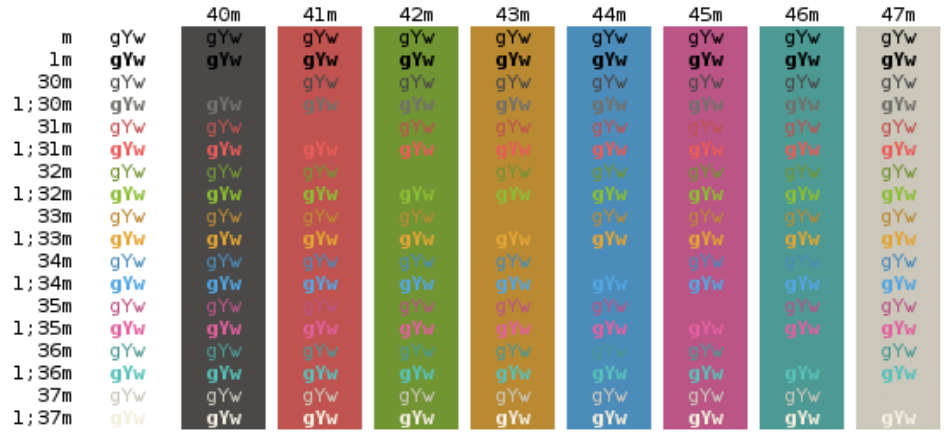
I first use `/bin/ls -1` to generate a list of all the files (and folders, which are also called files in Linux) in the current directory. I then use `|` to pipe (pass) that data to `/usr/bin/wc -l` which counts the number of newlines. This provides the number of files in the current directory.

The complete command is `/bin/ls -1 | /usr/bin/wc -l`

Size of Contents

I first use `/bin/ls -lah` which lists, in **l**ong form, **a**ll the files in the current directory, using **h**uman readable size information. I then use `|` to pipe that data to `/usr/bin/grep -m 1 total` which outputs the first (a **m**aximum of **1**) line containing the word "total". Finally, I use `/bin/sed 's/total //'` to replace the text "total " with nothing, thus discarding it and retaining just the size of the current directory's contents.

The complete command is `/bin/ls -lah | /usr/bin/grep -m 1 total | /bin/sed 's/total //'`



ANSI Colors

There are 16 ANSI Colors, which are actually 8 colors, each having "normal" and "bright" intensity variants. The colors are black, red, green, yellow, blue, magenta, cyan, and white. In certain circumstances bright intensity could be actually brighter, or, in the case of an xterm, it could be bold.

Usage Syntax

To set a font to the color and style desired, we use the syntax `\033[#m` where `#` can be a valid set of semicolon separated numbers. The `\033[` is a control sequence initiator which begins an escape sequence. In our case the sequence is a set of "`m`" terminated select graphic rendition parameters for rendering the text as desired. More details.

If that makes no sense, just concentrate on which numbers to use. Font color makes use of 30 to 37. Background color makes use of 40 to 47. A 0 (or no value given) for the font color resets it to the default, which depends on the setup. For additional rendering options, 1 sets the font to bright / bold, 4 sets it to underlined, 7 sets it to negative colors.

This might help. Run the command: `echo -e "testing \033[0;37;41mCOLOR1\033[1;35;44mCOLOR2\033[m"`

This should output `testing COLOR1COLOR2` with `COLOR1` being white (light gray) text on a red background, and `COLOR2` being bold / bright magenta text on a blue background. All subsequent text should be normally colored.

Try playing around with that echo command until you feel a bit more comfortable.

Reference Table

Obviously you don't want to come back to this page every time you want to change your Bash colors. Here is the shell script (origin) used to generate the lovely color table:

```
#!/bin/bash
#
# This file echoes a bunch of color codes to the terminal to demonstrate
# what's available. Each line is the color code of one forground color,
# out of 17 (default + 16 escapes), followed by a test use of that color
# on all nine background colors (default + 8 escapes).
#
T='gYw'   # The test text
echo -e "\n                 40m     41m     42m     43m     44m     45m     46m     47m";
for FGs in '    m' '   1m' '  30m' '1;30m' '  31m' '1;31m' '  32m' '1;32m' '  33m' '1;33m' '  34
m' '1;34m' '  35m' '1;35m' '  36m' '1;36m' '  37m' '1;37m';
  do FG=${FGs// /}
  echo -en " $FGs \033[$FG  $T  "
  for BG in 40m 41m 42m 43m 44m 45m 46m 47m;
    do echo -en "$EINS \033[$FG\033[$BG  $T \033[0m\033[$BG \033[0m";
  done
  echo;
done
echo
```

Finally, the Goal

Here is the code shared by both:

`c="\[\033["`

This is the PS1 used to create the slightly steam-punk looking Bash prompt, using box-drawing characters:

```
PS1="\n${p}╔╣ ${c}1;3\$(if [ \$? -eq 0 ]; then echo '2'; else echo '1'; fi)m\]*$p ╠ ${c}36m\]\@
\d$p ╠ ${c}35m\]\j$p ╠ ${c}30m\]\u${c}33m\]@\h$p ╠ ${c}1;34m\]\w$p ╠ ${c}32m\]\$(/bin/ls -1 | /u
sr/bin/wc -l) files, \$(/bin/ls -lah | /usr/bin/grep -m 1 total | /bin/sed 's/total //')${c}m\]$
p ╠═╗\n╚═${c}1;3\$(if [ ${EUID} -eq 0 ]; then echo '1'; else echo '4'; fi)m\]»${c}m\] "
```

And this is the PS1 used to create the more compact and somber looking Bash prompt:

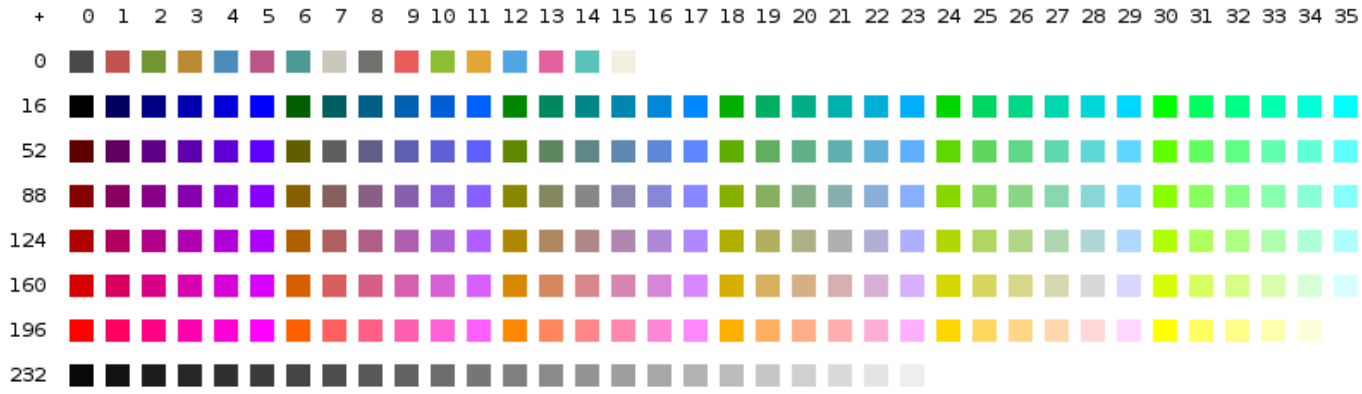
```
n="${c}m\]"
PS1="$n\n${c}1;3\$(if [ \$? -eq 0 ]; then echo '2'; else echo '1'; fi);40m\]*$n ${c}36;40m\]\@ \
d$n ${c}35;40m\]\j$n ${c}37;40m\]\u${c}33m\]@\h$n ${c}1;34;40m\]\w$n ${c}32;40m\]\$(/bin/ls -1 |
 /usr/bin/wc -l) files, \$(/bin/ls -lah | /usr/bin/grep -m 1 total | /bin/sed 's/total //')${c}m
\]\n${c}1;3\$(if [ ${EUID} -eq 0 ]; then echo '1'; else echo '4'; fi);47m\]>$n "
```

Hopefully, at this point you are reading these lines of seeming gibberish like plain English. If not, play more. You'll get it.

256 (8-bit) Colors

It is also possible (in these newfangled modern times) to use a full 256 colors in most consoles. The color table below can be used to determine the code for each color by adding the column and row number. In order to make use of these codes, we use the syntax `33[38;5;#m` for the foreground (text) and `33[48;5;#m` for the background, or in a single statement like `33[38;5;#;48;5;#m` to set both at once, where `#` is an 8-bit (0-255) color code.

The best way to understand is to try it out. Run the command: `echo -e "testing \033[38;5;196;48;5;21mCOLOR1\033[38;5;208;48;5;159mCOLOR2\033[m"`



Reference Table

Once again, you most likely don't want to come back here just to look up a 256 color code. This is my own code. It's shared under CC BY.

```
#!/bin/bash
#
# generates an 8 bit color table (256 colors) for
# reference purposes, using the \033[48;5;${val}m
# ANSI CSI+SGR (see "ANSI Code" on Wikipedia)
#
echo -en "\n   +  "
for i in {0..35}; do
  printf "%2b " $i
done

printf "\n\n %3b  " 0
for i in {0..15}; do
  echo -en "\033[48;5;${i}m  \033[m "
done
```

```
#for i in 16 52 88 124 160 196 232; do
for i in {0..6}; do
  let "i = i*36 +16"
  printf "\n\n %3b  " $i
  for j in {0..35}; do
    let "val = i+j"
    echo -en "\033[48;5;${val}m  \033[m "
  done
done

echo -e "\n"
```

See Also

Bash $PS1 Generator

Bash Shell PS1: 10 Examples to Make Your Linux Prompt like Angelina Jolie

ArchWiki - Color Bash Prompt

8 Useful and Interesting Bash Prompts

Comments

dahasak **2013, Jan 16, 03:33 pm**

> Thanks heaps for this article - it really saved me hours, if not days.
> Cheers
> Dahasak

Wallybear **2013, Jan 17, 12:01 am**

> Works just right on OS X.
> Thanks!

Raunaq **2014, Dec 29, 10:54 pm**

> Thanks for this. It certainly helped me colorize my Mac Shell. So yeah, it works with the Mac shell too.

ton **2015, Nov 18, 07:13 pm**

> Awesome post. Also you might want to explain and emphasis the surrounding escape sequence "\ [" and "\ ]" a bit as it would avoid some pitfall.
>
> The text was wrapping to the next line early on long commands on one version of my prompt and adding the bash escape sequence "\ [" and "\ ]" fixed it. Not sure if this is limited to bash or not. [no space between the two character]

Harpagophytum **2016, Mar 10, 04:34 am**

> Bonjour,
>
> Pile poil ce que je cherchais depuis longtemps.
>
> Un grand merci...

Ramesh **2016, Jun 10, 02:22 pm**

> Finally found a good code for shell colors. Works very well in Cygwin!!!
> Thank you

Ramesh **2016, Jun 10, 02:32 pm**

> Very nice code. Works very well in Cygwin & Redhat !!!
>
> Thank you

ChrisK **2016, Oct 20, 06:35 pm**

> Thanks loads!!!

Add a comment

Name

Email

Website Spam trap. Leave empty.

Comment

preview