

tput

Хотя наша среда командной строки, безусловно, мощная, ей может не хватать визуальной привлекательности. Наши терминалы не могут создать богатую среду графического пользовательского интерфейса, но это не значит, что мы обречены всегда смотреть на простые символы на простом фоне.

В этом приключении мы рассмотрим `tput` команду, используемую для управления нашим терминалом. С его помощью мы можем изменять цвет текста, применять эффекты и вообще делать вещи ярче. Что еще более важно, мы можем использовать `tput` для улучшения человеческого фактора в наших сценариях. Например, мы можем использовать цветовые и текстовые эффекты для лучшего представления информации нашим пользователям.

Доступность

`tput` является частью `ncurses` пакета и поставляется с большинством дистрибутивов Linux.

Что он делает / как это работает

Давным-давно, когда компьютеры были централизованы, пользователи интерактивных компьютеров общались с удаленными системами с помощью физического терминала или программы-эмулятора терминала, запущенной в какой-либо другой системе. В период своего расцвета существовало много типов терминалов, и все они использовали разные последовательности управляющих символов для управления своими экранами и клавиатурами.

Когда мы запускаем сеанс терминала в нашей системе Linux, эмулятор терминала устанавливает переменную `TERM` среды с именем *типа терминала*. Если мы посмотрим `TERM`, мы увидим это:

```
[me@linuxbox ~]$ echo $TERM
xterm
```

В этом примере мы видим, что наш тип терминала называется “`xterm`”, предполагая, что наш терминал ведет себя как классическая программа-эмулятор X terminal `xterm`. Другими распространенными типами терминалов являются “`linux`” для консоли Linux и “`screen`”, используемые терминальными мультиплексорами, такими как `screen` и `tmux`. Хотя мы чаще всего будем сталкиваться с этими 3 типами, на самом деле существуют тысячи различных типов терминалов. Наша система Linux содержит базу данных с именем *terminfo*, которая их описывает. Мы можем изучить типичную запись *terminfo*, используя `infocmp` команду, за которой следует имя типа терминала:

```
[me@linuxbox ~]$ infocmp screen
#   Reconstructed via infocmp from file: /lib/terminfo/s/screen
screen|VT 100/ANSI X3.64 virtual terminal,
    am, km, mir, msgr, xenl,
    colors#8, cols#80, it#8, lines#24, ncv@, pairs#64,
    acsc=++\,\,--..00`aaffgghhiijjkkllmmnnnooppqrrssttuuvvwwxxy
    bel=^G, blink=\E[5m, bold=\E[1m, cbt=\E[Z, civis=\E[?25l,
    clear=\E[H\E[J, cnorm=\E[34h\E[?25h, cr=^M,
    csr=\E[%i%p1%d;%p2%dr, cub=\E[%p1%dD, cub1=^H,
    cud=\E[%p1%dB, cud1=^J, cuf=\E[%p1%dC, cuf1=\E[C,
    cup=\E[%i%p1%d;%p2%dH, cuu=\E[%p1%dA, cuu1=\EM,
    cvvis=\E[34l, dch=\E[%p1%dP, dch1=\E[P, dl=\E[%p1%dM,
    dl1=\E[M, ed=\E[J, el=\E[K, el1=\E[1K, enacs=\E(B\E)0,
    flash=\Eg, home=\E[H, ht=^I, hts=\EH, ich=\E[%p1%d@,
    il=\E[%p1%dL, il1=\E[L, ind=^J, is2=\E)0, kbs=\177,
    kcbt=\E[Z, kcub1=\EOD, kcud1=\EOB, kcufl=\EOC, kcuu1=\EOA,
    kdch1=\E[3~, kend=\E[4~, kf1=\EOP, kf10=\E[21~,
    kf11=\E[23~, kf12=\E[24~, kf2=\EQO, kf3=\EOR, kf4=\EOS,
    kf5=\E[15~, kf6=\E[17~, kf7=\E[18~, kf8=\E[19~, kf9=\E[20~,
    khome=\E[1~, kich1=\E[2~, kmous=\E[M, knp=\E[6~, kpp=\E[5~,
    nel=\EE, op=\E[39;49m, rc=\E8, rev=\E[7m, ri=\EM, rmacs=^O,
    rmcup=\E[?1049l, rmir=\E[4l, rmkx=\E[?1l\E>, rmso=\E[23m,
    rmul=\E[24m, rs2=\Ec\E[?1000l\E[?25h, sc=\E7,
    setab=\E[4%p1%dm, setaf=\E[3%p1%dm,
    sgr=\E[0%?%p6%t;1%;%?%p1%t;3%;%?%p2%t;4%;%?%p3%t;7%;%?%p4%t;
    sgr0=\E[m\017, smacs=^N, smcup=\E[?1049h, smir=\E[4h,
    smkx=\E[?1h\E=, smso=\E[3m, smul=\E[4m, tbc=\E[3g,
```

Приведенный выше пример представляет собой запись terminfo для типа терминала “экран”. То, что мы видим на выходе infocmp, – это разделенный запятыми список имен возможностей терминала или *capabilities*. Некоторые из возможностей являются автономными – например, первые несколько в списке, – в то время как другим присваиваются зашифрованные значения. Возможности автономного терминала указывают на то, что терминал может делать. Например, возможность “am” указывает на то, что терминал имеет автоматическое правое поле. Возможности терминала с присвоенными значениями содержат строки, которые интерпретируются терминалом как команды. Значения, начинающиеся с “\E” (который представляет собой escape-символ), представляют собой последовательности управляющих кодов, которые заставляют терминал выполнять действие, такое как перемещение курсора в указанное местоположение или установка цвета текста.

tput Команда может использоваться для проверки конкретной возможности или для вывода назначенного значения. Вот несколько примеров:

```
tput longname
```

Это выводит полное имя текущего типа терминала. Мы можем указать другой тип терминала, включив -Топцию. Здесь мы запросим полное имя типа терминала с именем “screen”:

```
tput -T screen longname
```

Мы можем запросить значения из базы данных `terminfo`, например, количество поддерживаемых цветов и количество столбцов в текущем терминале:

```
tput colors  
tput cols
```

Мы можем протестировать определенные возможности. Например, чтобы узнать, поддерживает ли текущий терминал “все” (стирание цвета фона – это означает, что очистка или стирание текста будет выполняться с использованием текущего определенного цвета фона), мы вводим:

```
tput bce && echo "True"
```

Мы можем отправлять инструкции на терминал. Например, для перемещения курсора на позицию на 20 символов вправо и на 5 строк вниз:

```
tput cup 5 20
```

В базе данных `terminfo` определено много разных типов терминалов, и существует множество заглавных имен терминалов. Справочная страница `terminfo` содержит полный список. Обратите внимание, однако, что в общей практике существует лишь относительно небольшое количество имен заголовков, поддерживаемых всеми типами терминалов, с которыми мы, вероятно, столкнемся в системах Linux.

Чтение атрибутов терминала

Для следующих имен `tput` заголовков выводит значение в стандартный вывод:

Имя заголовка	Описание
<code>longname</code>	Полное имя типа терминала
<code>lines</code>	Количество строк в терминале
<code>cols</code>	Количество столбцов в терминале
<code>colors</code>	Количество доступных цветов

Имена возможностей

Значения `lines` и `cols` являются динамическими. То есть они обновляются по мере изменения размера окна терминала. Вот удобный псевдоним, который создает команду для просмотра текущего размера нашего окна терминала:

```
alias term_size=`echo "Rows=$(tput lines) Cols=$(tput cols)"`
```

Если мы определим этот псевдоним и выполним его, мы увидим отображаемый размер текущего терминала. Если мы затем изменим размер окна терминала и выполним псевдоним во второй раз, мы увидим, что значения были обновлены.

Одна интересная функция, которую мы можем использовать в наших сценариях, – это сигнал SIGWINCH. Этот сигнал отправляется каждый раз, когда изменяется размер окна терминала. Мы можем включить обработчик сигнала (т. Е. Ловушку) в наши скрипты, чтобы обнаружить этот сигнал и действовать в соответствии с ним:

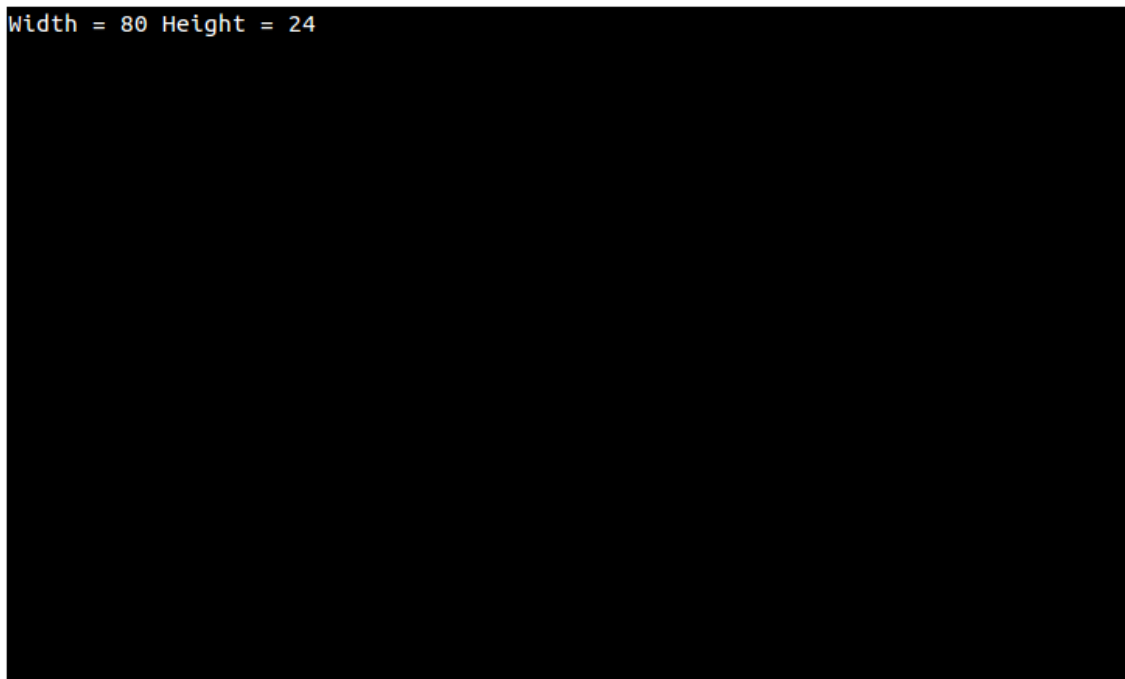
```
#!/bin/bash
# term_size2 - Dynamically display terminal window size

redraw() {
    clear
    echo "Width = $(tput cols) Height = $(tput lines)"
}

trap redraw WINCH

redraw
while true; do
    :
done
```

С помощью этого скрипта мы запускаем пустой бесконечный цикл, но поскольку мы устанавливаем ловушку для сигнала SIGWINCH, каждый раз, когда изменяется размер окна терминала, ловушка срабатывает и отображается новый размер терминала. Чтобы выйти из этого скрипта, мы вводим Ctrl-c.



term_size2

Управление курсором

приведенные ниже заголовки выводят строки, содержащие управляющие коды, которые инструктируют терминал управлять курсором:

Имя заголовка	Описание
---------------	----------

sc	Сохраните положение курсора
rc	Восстановите положение курсора
home	Переместите курсор в верхний левый угол (0,0)
cup <row> <col>	Переместите курсор в строку position, col
cud1	Переместите курсор на 1 строку вниз
cuu1	Переместите курсор на 1 строку вверх
civis	Установите курсор, чтобы он был невидимым
cnorm	Установите курсор в нормальное состояние

Названия элементов управления курсором

Мы можем изменить наш предыдущий скрипт, чтобы использовать позиционирование курсора и размещать размеры окна в центре по мере изменения размера терминала:

```
#!/bin/bash
# term_size3 - Dynamically display terminal window size
#               with text centering

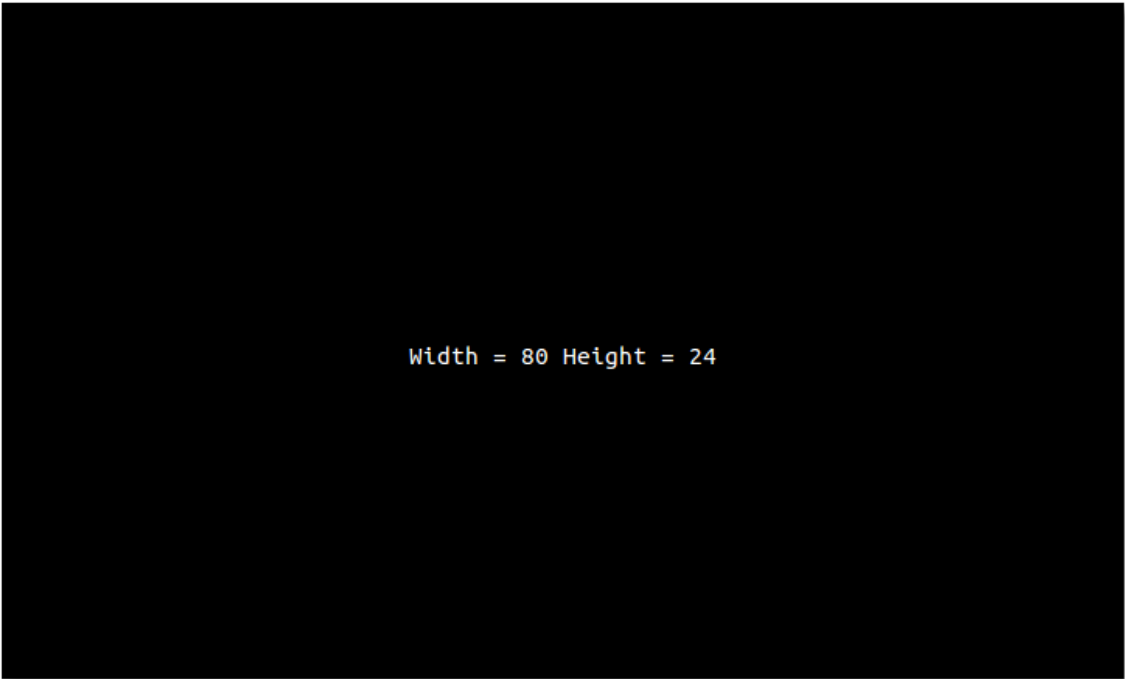
redraw() {
    local str width height length

    width=$(tput cols)
    height=$(tput lines)
    str="Width = $width Height = $height"
    length=${#str}
    clear
    tput cup $((height / 2)) $(((width / 2) - (length / 2)))
    echo "$str"
}

trap redraw WINCH

redraw
while true; do
    :
done
```

Как и в предыдущем сценарии, мы устанавливаем ловушку для сигнала SIGWINCH и запускаем бесконечный цикл. Функция перерисовки в этом скрипте немного сложнее, поскольку она должна вычислять центр окна терминала каждый раз, когда его размер изменяется.



term_size3

Текстовые эффекты

Как и имена заголовков, используемые для управления курсором, следующие имена заголовков выводят строки управляющих кодов, которые влияют на то, как наш терминал отображает текстовые символы:

Имя заголовка	Описание
<code>bold</code>	Начать выделенный жирным шрифтом текст
<code>smul</code>	Начать подчеркнутый текст
<code>rmul</code>	Завершить подчеркнутый текст
<code>rev</code>	Запуск обратного видео
<code>blink</code>	Начать мигающий текст
<code>invis</code>	Запустить невидимый текст
<code>smso</code>	Запуск “выделенного” режима
<code>rmso</code>	Завершение режима “выдающийся”
<code>sgr0</code>	Отключите все атрибуты
<code>setaf</code>	Установите цвет переднего плана

<code><value></code> Имя заголовка	Описание
<code>setab</code> <code><value></code>	Установите цвет фона

Названия текстовых эффектов

Некоторые функции, такие как подчеркивание и выделение, имеют заглавные буквы для включения и выключения атрибута, в то время как у других есть только заглавное имя для включения атрибута. В этих случаях `sgr0` может использоваться для возврата отображения текста в “нормальное” состояние. Вот простой скрипт, который демонстрирует обычные текстовые эффекты:

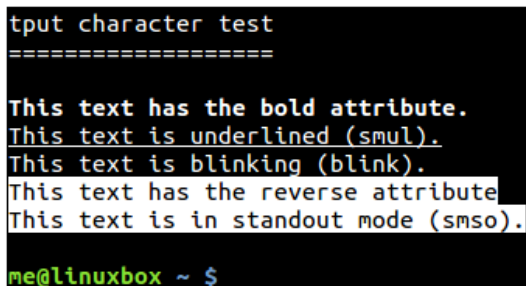
```
#!/bin/bash

# tput_characters - Test various character attributes

clear

echo "tput character test"
echo "====="
echo

tput bold;   echo "This text has the bold attribute.";      tput smul;   echo "This text is underlined (smul).";      tput blink;  echo "This text is blinking (blink).";          tput rev;    echo "This text has the reverse attribute";      tput smso;   echo "This text is in standout mode (smso)."; tput sgr0
echo
```



```
tput character test
=====

This text has the bold attribute.
This text is underlined (smul).
This text is blinking (blink).
This text has the reverse attribute
This text is in standout mode (smso).

me@linuxbox ~ $
```



tput_characters

Text Color

Most terminals support 8 foreground text colors and 8 background colors (though some support as many as 256). Using the setaf and setab capabilities, we can set the foreground and background colors. The exact rendering of colors is a little hard to predict. Many desktop managers impose “system colors” on terminal windows, thereby modifying foreground and background colors from the standard. Despite this, here are what the colors should be:

Value	Color
0	Black
1	Red
2	Green
3	Yellow
4	Blue
5	Magenta
6	Cyan
7	White
8	Not used
9	Reset to default color

Text colors

The following script uses the setaf and setab capabilities to display the available foreground/background color combinations:

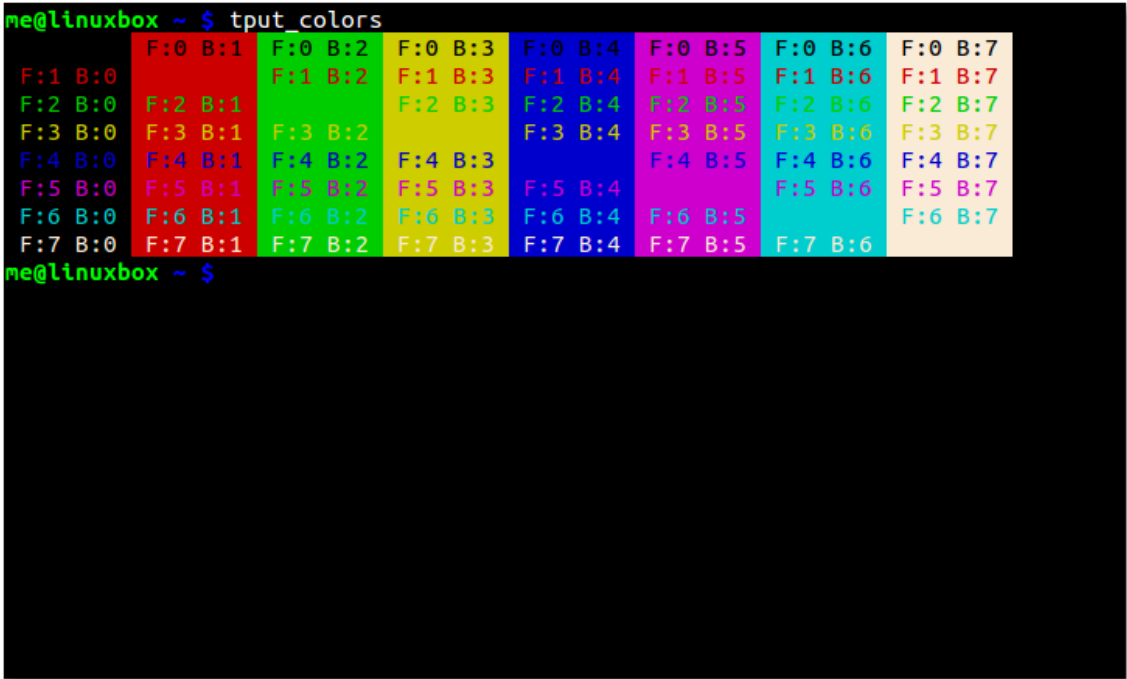
```
#!/bin/bash

# tput_colors - Demonstrate color combinations.

for fg_color in {0..7}; do
    set_foreground=$(tput setaf $fg_color)
    for bg_color in {0..7}; do
        set_background=$(tput setab $bg_color)
        echo -n $set_background$set_foreground
```



```
printf ' F:%s B:%s ' $fg_color $bg_color
done
echo $(tput sgr0)
done
```



tput_colors

Clearing the Screen

These capnames allow us to selectively clear portions of the terminal display:

Capname	Description
smcup	Save screen contents
rmcup	Restore screen contents
el	Clear from the cursor to the end of the line
ell	Clear from the cursor to the beginning of the line
ed	Clear from the cursor to the end of the screen
clear	Clear the entire screen and home the cursor

Screen erasure capnames

Using some of these terminal capabilities, we can construct a script with a menu and a separate output area to display some system information:

```
#!/bin/bash

# tput_menu: a menu driven system information program
```

```

BG_BLUE="$(tput setab 4)"
BG_BLACK="$(tput setab 0)"
FG_GREEN="$(tput setaf 2)"
FG_WHITE="$(tput setaf 7)"

# Save screen
tput smcup

# Display menu until selection == 0
while [[ $REPLY != 0 ]]; do
    echo -n ${BG_BLUE}${FG_WHITE}
    clear
    cat <<- _EOF_
        Please Select:

        1. Display Hostname and Uptime
        2. Display Disk Space
        3. Display Home Space Utilization
        0. Quit

_EOF_

    read -p "Enter selection [0-3] > " selection

    # Clear area beneath menu
    tput cup 10 0
    echo -n ${BG_BLACK}${FG_GREEN}
    tput ed
    tput cup 11 0

    # Act on selection
    case $selection in
        1) echo "Hostname: $HOSTNAME"
            uptime
            ;;
        2) df -h
            ;;
        3) if [[ $(id -u) -eq 0 ]]; then
                echo "Home Space Utilization (All Users)"
                du -sh /home/* 2> /dev/null
            else
                echo "Home Space Utilization ($USER)"
                du -s $HOME/* 2> /dev/null | sort -nr
            fi
            ;;
        0) break
            ;;
        *) echo "Invalid entry."
            ;;
    esac
    printf "\n\nPress any key to continue."
    read -n 1
done

# Restore screen

```

```
tput rmcup
echo "Program terminated."
```

```
Please Select:

1. Display Hostname and Uptime
2. Display Disk Space
3. Display Home Space Utilization
0. Quit

Enter selection [0-3] > 2

Filesystem      Size  Used Avail Use% Mounted on
rootfs          5.8G  4.0G  1.6G   73% /
/dev/root       5.8G  4.0G  1.6G   73% /
devtmpfs        215M    0  215M    0% /dev
tmpfs           44M   260K   44M    1% /run
tmpfs           5.0M    0   5.0M    0% /run/lock
tmpfs           88M    0   88M    0% /run/shm
/dev/mmcblk0p5  60M   9.6M   50M   17% /boot

Press any key to continue.
```

tput_menu

Making Time

For our final exercise, we will make something useful; a large character clock. To do this, we first need to install a program called banner. The banner program accepts one or more words as arguments and displays them like so:

```
[me@linuxbox ~]$ banner "BIG TEXT"
#####      ###      #####      #####      #####      #      #      #####
#      #      #      #      #      #      #      #      #      #
#      #      #      #      #      #      #      #      #      #
#####      #      #      #####      #      #####      #      #
#      #      #      #      #      #      #      #      #      #
#      #      #      #      #      #      #      #      #      #
#####      ###      #####      #      #####      #      #      #
```

This program has been around for a long time and there are several different implementations. On Debian-based systems (such as Ubuntu) the package is called “sysvbanner”, on Red Hat-based systems the package is called simply “banner”. Once we have banner installed we can run this script to display our clock:

```
#!/bin/bash

# tclock - Display a clock in a terminal

BG_BLUE="$(tput setab 4)"
FG_BLACK="$(tput setaf 0)"
```

```

FG_WHITE="$(tput setaf 7)"

terminal_size() { # Calculate the size of the terminal

    terminal_cols="$(tput cols)"
    terminal_rows="$(tput lines)"
}

banner_size() {

    # Because there are different versions of banner, we need
    # calculate the size of our banner's output

    banner_cols=0
    banner_rows=0

    while read; do
        [[ ${#REPLY} -gt $banner_cols ]] && banner_cols=${#REPLY}
        ((++banner_rows))
    done < <(banner "12:34 PM")
}

display_clock() {

    # Since we are putting the clock in the center of the term
    # we need to read each line of banner's output and place i
    # right spot.

    local row=$clock_row

    while read; do
        tput cup $row $clock_col
        echo -n "$REPLY"
        ((++row))
    done < <(banner "$(date +%I:%M %p)")
}

# Set a trap to restore terminal on Ctrl-c (exit).
# Reset character attributes, make cursor visible, and restor
# previous screen contents (if possible).

trap 'tput sgr0; tput cnorm; tput rmcup || clear; exit 0' SIGHUP

# Save screen contents and make cursor invisible
tput smcup; tput civis

# Calculate sizes and positions
terminal_size
banner_size
clock_row=$((terminal_rows - banner_rows) / 2)
clock_col=$((terminal_cols - banner_cols) / 2)
progress_row=$((clock_row + banner_rows + 1))
progress_col=$((terminal_cols - 60) / 2)

# In case the terminal cannot paint the screen with a backgr

```

```

# color (tmux has this problem), create a screen-size string
# spaces so we can paint the screen the hard way.

blank_screen=
for ((i=0; i < (terminal_cols * terminal_rows); ++i)); do
    blank_screen="${blank_screen} "
done

# Set the foreground and background colors and go!
echo -n ${BG_BLUE}${FG_WHITE}
while true; do

    # Set the background and draw the clock

    if tput bce; then # Paint the screen the easy way if bce i
        clear
    else # Do it the hard way
        tput home
        echo -n "$blank_screen"
    fi
    tput cup $clock_row $clock_col
    display_clock

    # Draw a black progress bar then fill it in white
    tput cup $progress_row $progress_col
    echo -n ${FG_BLACK}
    echo -n "#####"
    tput cup $progress_row $progress_col
    echo -n ${FG_WHITE}

    # Advance the progress bar every second until a minute is
    for ((i = $(date +%S); i < 60; ++i)); do
        echo -n "#"
        sleep 1
    done
done

```





tclock script in action

Our script paints the screen blue and places the current time in the center of the terminal window. This script does not dynamically update the display's position if the terminal is resized (that's an enhancement left to the reader). A progress bar is displayed beneath the clock and it is updated every second until the next minute is reached, when the clock itself is updated.

One interesting feature of the script is how it deals with painting the screen. Terminals that support the "bce" capability erase using the current background color. So, on terminals that support bce, this is easy. We simply set the background color and then clear the screen. Terminals that do not support bce always erase to the default color (usually black).

To solve this problem, our this script creates a long string of spaces that will fill the screen. On terminal types that do not support bce (for example, screen) the background color is set, the cursor is moved to the home position and then the string of spaces is drawn to fill the screen with the desired background color.

Summing Up

Using tput, we can easily add visual enhancements to our scripts. While it's important not to get carried away, lest we end up with a garish, blinking mess, adding text effects and color can increase the visual appeal of our work and improve the readability of information we present to our users.

Further Reading

- The terminfo man page contains the entire list of terminal capabilities defined terminfo database.
- On most systems, the /lib/terminfo and /usr/share/terminfo directories contain the all of the terminals supported by terminfo.
- [Bash Hacker's Wiki](#) has a good entry on the subject of text effects using tput. The page also has some interesting example scripts.
- [Greg's Wiki](#) contains useful information about setting text colors using tput.
- [Bash Prompt HOWTO](#) discusses using tput to apply text effects to the shell prompt.

© 2000–2022, [William E. Shotts, Jr.](#) Verbatim copying and distribution of this entire article is permitted in any medium, provided this copyright notice is preserved.

Linux® is a registered trademark of Linus Torvalds.

