

第21章 创建屏幕输出

用户可以使用 shell 脚本创建交互性的、专业性强的屏幕输出。要实现这一点，系统上需要一个彩色监视器和 tput 命令。

本章内容有：

- tput 命令。
- 使用转义序列和产生控制码。
- 使用颜色。

作者写这本书时，遇到了 tput 命令的三种不同变形。至今为止最好的是 GNU tput，如果没有这个版本，首先下载它并安装在你的系统里。tput 使用文件 /etc/terminfo 或 /etc/termcap，这样就可以在脚本中使用终端支持的大部分命令了。

虽然 tput 不识别颜色设置，但是可以使用控制字符实现这一点。

21.1 tput

在使用 tput 前，需要在脚本或命令行中使用 tput 命令初始化终端。

```
$ tput init
```

tput 产生三种不同的输出：字符型、数字型和布尔型（真 / 假）。以下分别介绍其使用功能。

21.1.1 字符串输出

下面是大部分常用字符串：

名 字	含 义
bel	警铃
blink	闪烁模式
bold	粗体
civis	隐藏光标
clear	清屏
cnorm	不隐藏光标
cup	移动光标到屏幕位置 (x , y)
el	清除到行尾
ell	清除到行首
smso	启动突出模式
rmso	停止突出模式
smul	开始下划线模式
rmul	结束下划线模式
sc	保存当前光标位置
rc	恢复光标到最后保存位置
sgr0	正常屏幕
rev	逆转视图

21.1.2 数字输出

以下是大部分常用数字输出。

名 字	含 义
cols	列数目
it	tab设置宽度
lines	屏幕行数

21.1.3 布尔输出

在tput中只有两种布尔操作符。

名 字	含 义
chts	光标不可见
hs	具有状态行

21.2 tput用法

上面讲过了可能用到的tput的大多数常用名。现在学习在一段脚本中使用tput。

21.2.1 设置tput命令

可以取得所有tput名字输出，将其保存为更有意义的变量名。格式如下：

```
variable_name='tput name'
```

21.2.2 使用布尔输出

可以在if语句中使用布尔型tput输出。

```
STATUS_LINE='tput hs'
if $STATUS_LINE; then
    echo "your terminal has a status line"
else
    echo "your terminal has NO status line"
fi
```

21.2.3 在脚本中使用tput

以下脚本设置tput bel和cl为更有意义的变量名。

```
$ pg tput1
#!/bin/sh
BELL='tput bel'
CLEAR='tput cl'
```

```
echo $BELL
echo $CLEAR
```

下面脚本改变两个视图属性，并将光标关闭和打开。

```
$ pg tput2
#!/bin/sh
```

```

BOLD=`tput bold`
REV=`tput rev`
NORMAL=`tput sgr0`
CURSOR_OFF=`tput civis`
CURSOR_ON=`tput cnorm`
tput init

#turn cursor off, highlight text, reverse some text, cursor on
echo $CURSOR_OFF
echo "${BOLD} WELCOME TO THE PIZZA PLACES${NORMAL}"
echo -e "\n${REV} WE ARE OPEN 7 DAYS A WEEK${NORMAL}"
echo $CURSOR_ON

```

21.2.4 产生转义序列

注意，如果正在使用一个仿真器，要使光标不可见，这个操作可能会有问题。这是因为：

1) 一些仿真器并不捕获使光标不可见的控制字符。必须要求正在使用的软件仿真的制作者修改源脚本以关闭光标。

2) tput civis命令的一些旧版本工作不正常。

关闭光标的控制字符是？25l（这是字母l），将之打开是？25h。

所有控制字符均以转义序列开始。通常转义键后紧跟字符[。然后实际序列打开或关闭某终端属性。

可以使用两种不同的方法产生转义序列。下面的列表依据用户系统列出两种方法。第三种方法对于UNIX和Linux支持的变量均有效，因为控制序列嵌在echo语句中。本书将使用这种方法。

要发送一转义序列以关闭光标：

LINUX/BSD	echo -e "\033[?25l"
System V	echo "\033[?25l"
Generic method	echo "<CTRL-V><ESCAPE>[?25l"

\033为转义键取值，\通知echo命令接下来是一个八进制值。例如要反馈一个@字符，键入：

```
echo "@" 或者 echo -e "\100"
```

对于系统v，使用

```
echo "\100"
```

结果是一样的。

命令clear表示清屏并发送光标到屏幕左上角，此位置一般也称为home。在一个VT终端范围实现此功能所需序列为ESCIIJ，可以使用echo语句发送这一序列。

System V	echo "\033[2J"
LINUX/BSD	echo -e "\033[2J"

对于嵌入在文本中的任何控制字符，不要试图剪切和粘贴，因为这样会失去其特殊含义。

例如，要插入控制字符，打开光标，方法如下：

```
echo '<CTRL-V> hit the<ESCAPE> key then [?25h'
```

即先击<CTRL-V>，再击退格键，确保这不是一个仿真器。然后加入一小段脚本将之

打开和关闭。可以将这段脚本编成一个函数或者在后面几页找一下这段脚本。

```
$ pg cursor
#!/bin/sh

# cursor on|off
# turns the cursor on or off for the vt100, 200, 220, meth220
# note : will work on normal tty connec.. if'ie on some win emulations
# check TERM env for your type !
_OPT=$1
if [ $# -ne 1 ]; then
    echo "Usage: `basename $0` cursor [on|off]"
    exit 1
fi

case "$_OPT" in
on|ON|On)
    # turn it on (cursor)
    ON=`echo ^[[?25h`
    echo $ON
    ;;
off|OFF|Off)
    # turn it off (cursor)
    OFF=`echo ^[[?25l`
    echo $OFF
    ;;
*)
    echo "Usage: cursor on|off"
    exit 1
    ;;
esac
```

21.2.5 光标位置

可以用tput将光标放在屏幕任意位置。格式为：

```
cup r c
```

r为从上至下屏幕行数，c为穿过屏幕列数。

最好将之编成函数，这样就可以把行和列的值传递给它。

```
xy()
{
    #_R= row, _C=column
    _R=$1
    _C=$2
    tput cup $_R $_C
}

clear
xy 1 5
echo -n "Enter your name : "
read NAME
xy 2 5
echo -n "Enter your age : "
read AGE
```

当然再传递一个字符串给它也很合适。以下是稍加改动后的函数脚本。

```
xy()
{
#_R= row, _C=column
_R=$1
_C=$2
_TEXT=$3
tput cup $_R $_C
echo -n $_TEXT
}
```

这可以像下面这样调用：

```
xy 5 10 "Enter your password : "
read CODE
```

21.2.6 在屏幕中心位置显示文本

在屏幕中心位置显示文本不是很麻烦。首先从 tput 中得到列数，然后算出所提供的字符串长度，从 tput 列数中减去该值，结果再除以 2，所得结果可用于显示的字符串的行数。

以下脚本实现此功能。只需稍加改动即可从文件中读取各行并在屏幕中间位置显示文本。

输入一些字符，点击回车键，文本将显示在屏幕中间第 10 行。

```
echo -n "input string : "
read STR
# quick way of getting length of string
LEN=`echo $STR | wc -c`
COLS=`tput cols`
NEW_COL=`expr \(${COLS} - $LEN\) / 2`
xy 10 $NEW_COL
echo $STR
```

将上述脚本编成函数，并带有两个参数：文本和行数，这样调用更加灵活，函数如下：

```
centertext()
{
_ROW=$1
_STR=$2
# quick way of getting length of string
LEN=`echo $_STR | wc -c`
COLS=`tput cols`
_NEW_COL=`expr \(${COLS} -- $LEN\) / 2`
xy $_ROW $_NEW_COL
echo $_STR
}
```

可使用下述格式调用上述函数：

```
centertext 15 "THE MAIN EVENT"
```

或者用字符串作参数：

```
centertext 15 $1
```

21.2.7 查找终端属性

下面脚本使用 tput 访问 terminfo，显示前面提到过的 tput 命令下的一些终端转义码。

```
$ pg termpu
#!/bin/sh
# termpu
```

```
#init tput for your terminal
tput init

clear

echo " tput <> terminfo"
infocmp -1 $TERM | while read LINE
do
    case $LINE in
        bel*) echo "$LINE: sound the bell" ;;
        blink*) echo "$LINE: begin blinking mode" ;;
        bold*) echo "$LINE: make it bold" ;;
        el*) echo "$LINE: clear to end of line" ;;
        civis*) echo "$LINE: turn cursor off" ;;
        cnorm*) echo "$LINE: turn cursor on" ;;
        clear*) echo "$LINE: clear the screen" ;;
        kcuu1*) echo "$LINE: up arrow" ;;
        kcub1*) echo "$LINE: left arrow" ;;
        kcufl*) echo "$LINE: right arrow" ;;
        kcud1*) echo "$LINE: down arrow" ;;
    esac
done
```

命令infocmp从terminfo数据库中抽取终端信息，如果要查看终端定义文件的完整列表，可使用命令：

```
$ infocmp $TERM
```

以下是termput脚本的终端输出：

```
$ termput
tput <> terminfo
bel=^G,: sound the bell
blink=E[5m,: begin blinking mode
bold=E[lm,: make it bold
civis=E[?25l,: turn cursor off
clear=E[HE[J,: clear the screen
cnorm=E[?25h,: turn cursor on
el=E[K,: clear to end of line
ell=E[lK,: clear to end of line
kcub1=E[D,: left arrow
kcud1=E[B,: down arrow
kcufl=E[C,: right arrow
kcuu1=E[A,: up arrow
```

21.2.8 在脚本中使用功能键

使用cat命令可以查看发送的任意特殊键控制序列（F1，上箭头等），键入cat -v，然后按任意控制键，回车，在下一行就可以知道终端发送了什么功能键。结束后按 <Ctrl-c>退出。

下面的例子运行cat命令，输入键为F1（^[OP]），F2([OQ]），上箭头^[^[A]。

```
$ cat -v
^[OP^[OQ^[^[A
<CTRL-C>
```

有了这些信息，就可以在脚本中插入这些字符作为用户选择的另外一些方法。

下面脚本识别F1、F2和箭头键，取值可能不同，因此要先用cat命令查看用户终端控制键发送的取值。

```
$ pg control_keys
#!/bin/sh
# control_keys
# to insert use '<CTRL-V><ESCAPE>sequence'
uparrowkey='^[A'
downarrowkey='^[B'
leftarrowkey='^[D'
rightarrowkey='^[C'
f1key='^[OP'
f2key='^[OQ'

echo -n " Press a control key then hit return"
read KEY

case $KEY in
$uparrowkey) echo "UP Arrow"
;;
$downarrowkey) echo "DOWN arrow"
;;
$leftarrowkey) echo "LEFT arrow"
;;
$rightarrowkey) echo "RIGHT arrow"
;;
$f1key) echo "F1 key"
;;
$f2key) echo "F2 key"
;;
*) echo "unknown key $key"
;;
esac
```

21.2.9 使用颜色

对域使用颜色可以使数据输入屏幕看起来更加专业。下面将使用的颜色是 ANSI标准颜色，并不是所有颜色都适合于所有系统。下面列出了大部分常用颜色。

1. 前景色：

数 字	颜 色	数 字	颜 色
30	黑色	34	蓝色
31	红色	35	紫色
32	绿色	36	青色
33	黄（或棕）色	37	白（或灰）色

2. 背景色：

数 字	颜 色	数 字	颜 色
40	黑色	44	青色
41	红色	45	蓝色
42	绿色	46	青色
43	黄（或棕）色	47	白（或灰）色

显示前景或背景颜色格式为：

```
<ESCAPE> [background_number;foreground_number m
```

21.2.10 产生颜色

产生颜色需要在 echo 语句中嵌入控制字符。这种方法适用于带有彩色终端的任何系统。与在控制字符里一样，可以在 echo 语句里使用转义序列产生颜色。

要产生一个黑色背景加绿色前景色：

```
LINUX/BSD      echo -e "\033[40;32m"
System V       echo "\033[40;32m"
Generic method echo "<CTRL-V><ESCAPE>[40;32m"
```

一般方法是先击<Ctrl-v>，然后是<ESCAPE>键，接着是[40;32m。本书使用这种方法。可能发现将颜色设置与 echo 语句放在一个 case 语句里，然后将之编成一个函数，这样做最好。下面是作者编写的颜色函数。

```
colour()
{
# format is background;foregroundm
case $1 in
black_green)
echo '^[[40;32m'
;;
black_yellow)
echo '^[[40;33m'
;;
black_white)
echo '^[[40;37m'
;;
black_cyan)
echo '^[[40;36m'
;;
red_yellow)
echo '^[[41;32m'
;;
black_blue)
echo '^[[40;34m'
;;
esac
}
```

要调用颜色 red-yellow (红色背景，黄色前景)，方法如下：

```
colour red-yellow
```

在脚本中可以这样使用颜色：

```
colour what_ever
echo something
# now change to a different colour
colour what_ever
echo something
```

作者终端的缺省屏幕颜色是黑色和白色。但是如果要用黑色背景加绿色前景，可插入一个 echo 语句，同时将之放入用户 .profile 文件中。

图21-2显示加入颜色设置后的基本输出屏幕。这种颜色看起来更加吸引人

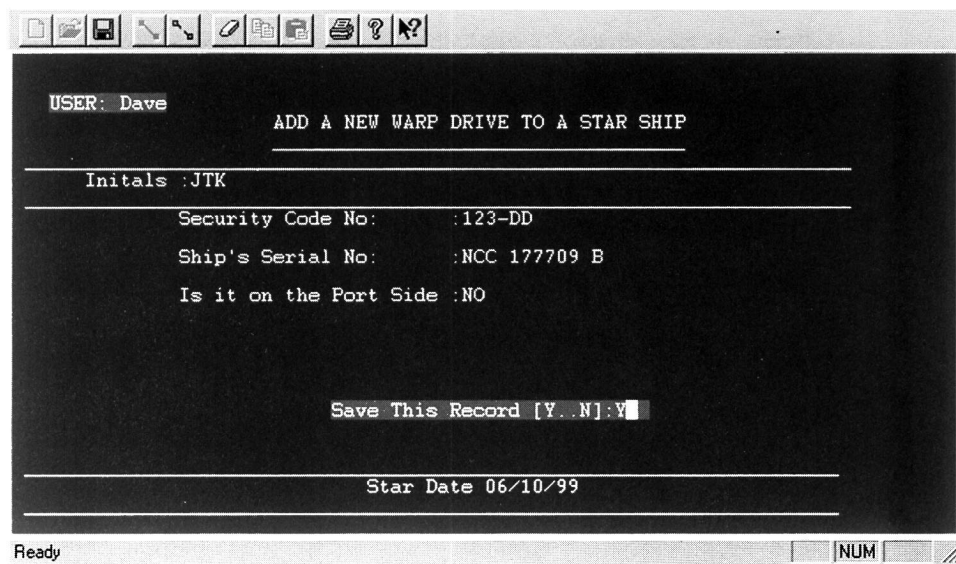


图21-1 下述脚本使用颜色和tput命令所得的屏幕概貌

下面是显示图21-1屏幕的脚本：

```
$ pg colour_scr
#!/bin/sh
# colour_scr
tput init
MYDATE=`date +%D`
colour()
{
# format is background;foregroundm
case $1 in
black_green)
echo '^[[40;32m'
;;
black_yellow)
echo '^[[40;33m'
;;
black_white)
echo '^[[40;37m'
;;
black_cyan)
echo '^[[40;36m'
;;
black_red)
echo '^[[40;31m'
;;
esac
}

xy()
# xy
# to call: xy row, column, "text"
# goto xy screen co-ordinates
{
```

```

#_R= row, _C=column
_R=$1
_C=$2
_TEXT=$3
tput cup $_R $_C
echo -n $_TEXT
}

center()
{
# center
# centers a string of text across screen
# to call: center "string" row_number
_STR=$1
_ROW=$2
# crude way of getting length of string
LEN=`echo $_STR | wc -c`

COLS=`tput cols`
HOLD_COL=`expr $COLS - $LEN`
NEW_COL=`expr $HOLD_COL / 2`
tput cup $_ROW $NEW_COL
echo -n $_STR
}

tput clear
colour red_yellow
xy 2 3 "USER: $LOGNAME"
colour black_cyan
center "ADD A NEW WARP DRIVE TO A STAR SHIP" 3
echo -e "\f\f"
center " " 4

colour black_yellow
xy 5 1 " "
xy 7 1 " "
xy 21 1 " "
center "Star Date $MYDATE " 22
xy 23 1 " "

colour black_green
xy 6 6 "Initials :"
read INIT
xy 8 14
echo -n "Security Code No:      :"
read CODE
xy 10 14
echo -n "Ship's Serial No:      :"
read SERIAL
xy 12 14
echo -n "Is it on the Port Side : "
read PORT

colour red_yellow
center " Save This Record [Y..N]: " 18
read ans

```

```
#reset to normal
colour black_white
```

如你所见，这个脚本没有经过验证，这样也行，因为这里脚本的目标只是显示怎样为屏幕上色。

21.2.11 创建精致菜单

在讲述while循环时曾经创建过菜单，现在增加菜单脚本，菜单将具有下列选项：

```
1 : ADD A RECORD
2 : VIEW A RECORD
3 : PAGE ALL RECORDS
4 : CHANGE A RECORD
5 : DELETE A RECORD
P : PRINT ALL RECORDS
H : Help screen
Q : Exit Menu
```

本脚本使用read_char函数，使用户在选择菜单选项时不必敲入回车键。trap命令（本书后面提到）用于忽略信号2、3和15，这样将防止用户试图跳出菜单。

此菜单还有一些控制访问形式。授权用户可以修改和删除记录，其余用户只能增加，查看或打印记录。带有访问级别的有效用户列表保存在文件 priv.user中。

用户运行菜单时，如果菜单名在文件中不存在，将被告之不能运行此应用并且退出。

只出于显示目的，系统命令就替换了实际的选项执行操作。执行时我们会发现用户 root，dave和matty不能修改数据库文件，而peter和louise可以。

```
$ pg priv.user
# prov.user access file for apps menu
# edit this at your own risk !!!!
# format is USER AMEND/DELETE records
# example root yes means yes root can amend or delete recs
# "      dave no  means no dave cannot amend or delete recs
root no
dave no
peter yes
louise yes
matty no
```

要检查用户权限，首先需要读入列表文件，忽略注释行，将其他行重定向到一个临时文件中。

```
user_level()
{
while read LINE
do
    case $LINE in
        \#*);;
        *) echo $LINE >>$HOLD1
           ;;
        esac
done < $USER_LEVELS

FOUND=false
```

```

while read MENU_USER PRIV
do
    if [ "$MENU_USER" = "$USER" ];
    then
        FOUND=true
        case $PRIV in
            yes|YES)
                return 0
                ;;
            no|NO)
                return 1
                ;;
            esac
        else
            continue
        fi
    done <$HOLD1
    if [ "$FOUND" = "false" ]; then
        echo "Sorry $USER you have not been authorised to use this menu"
        exit 1
    fi
}

```

下一步是读取新形成的格式化文件，变量 FOUND 首先设置为假，临时文件保存名字和权限级别。分别用用户名和权限级别设置为一个变量，然后执行测试文件中名字是否匹配 USER。USER 取值是从脚本开始时 whoami 命令中获得的。如果不匹配，则寻找其他用户，使用命令 continue 循环进入下一步。

处理过程直至所有用户名读取和匹配完毕。如果整个文件中均未找到匹配用户名，脚本末尾的 test 语句捕获权限级别，对一般访问级别为 1，对高级访问权限返回 0。

当用户选择修改或删除记录时，基于上述函数的返回值进行了一项测试。这个例子中 passwd 文件被分类或列出一个目录清单。

```

if user_level; then
    sort /etc/passwd
else
    restrict
fi

```

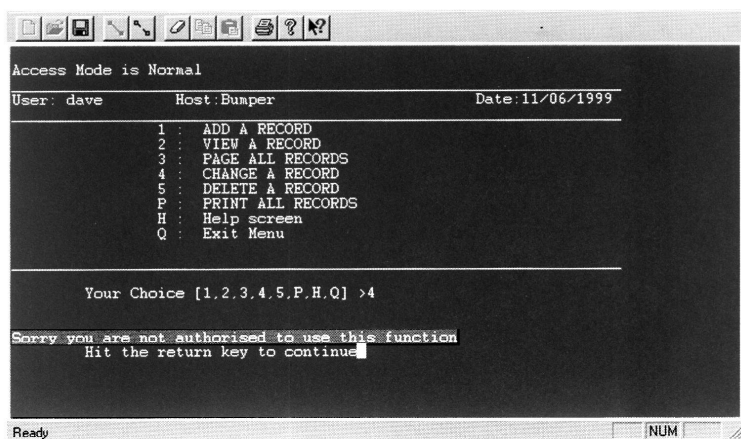


图21-2 带有访问限制的菜单屏幕概貌

Restrict是一个只打印违规操作提示符的函数。

上述测试在一个循环里即可完成，但考虑到脚本清晰性，使用了两个文件的方法，这样调试起来会更容易。

图21-2显示用户dave试图修改记录，但只具有一般权限，因此被提示权限不够。

用户可以选择q或Q退出菜单。退出时调用一个清屏函数。这样做的好处在于可以随意增加要运行的命令，同时也增加脚本的可读性。

脚本如下：

```
$ pg menu2
#!/bin/sh
# menu2
# MAIN MENU SCRIPT
# ignore CTRL-C and QUIT interrupts
trap "" 2 3 15
MYDATE=`date +%d/%m/%Y`
THIS_HOST=`hostname -s`

USER=`whoami`

# user level file
USER_LEVELS=priv.user

# hold file
HOLD1=hold1.$$

# colour function
colour()
{
# format is background;foregroundm
case $1 in
black_green)
echo '^[[40;32m'
;;
black_yellow)
echo '^[[40;33m'
;;
black_white)
echo '^[[40;37m'
;;
black_cyan)
echo '^[[40;36m'
;;
red_yellow)
echo '^[[41;33m'
;;
esac
}

# just read a single key please
get_char()
{
```

```
# get_char
# save current stty settings
SAVEDSTTY=`stty -g`
    stty cbreak
    dd if=/dev/tty bs=1 count=1 2> /dev/null
    stty -cbreak
# restore stty
stty $SAVEDSTTY
}

# turn the cursor on or off
cursor()
{
# cursor
#turn cursor on/off

_OPT=$1
case $_OPT in
on) echo '^[[?25h'
    ;;
off) echo '^[[?25l'
    ;;
*) return 1
    ;;
esac
}

# check what privilege level the user has
restrict()
{
colour red_yellow
echo -e -n "\n\n\007Sorry you are not authorised to use this function"
colour black_green
}

user_level()
{
# user_level
# read in the priv.user file
while read LINE
do
    case $LINE in
    # ignore comments
    \#*);;
    *) echo $LINE >>$HOLD1
        ;;
    esac
done < $USER_LEVELS

FOUND=false
while read MENU_USER PRIV
do
    if [ "$MENU_USER" = "$USER" ];
    then
        FOUND=true
```

```

    case $PRIV in
    yes|YES)
        return 0
        ;;
    no|NO)
        return 1
        ;;
    esac
else
# no match found read next record
    continue
fi
done <$HOLD1
if [ "$FOUND" = "false" ]; then
    echo "Sorry $USER you have not been authorised to use this menu"
    exit 1
fi
}

# called when user selects quit
my_exit()
{
# my_exit
# called when user selects quit!
colour black_white
    cursor on
    rm *.$$
    exit 0
}

tput init
# display their user levels on the screen
if user_level; then
    ACCESS="Access Mode is High"
else
    ACCESS="Access Mode is Normal"
fi

tput init
while :
do
tput clear
colour black_green
cat <<MAYDAY
$ACCESS

```

User: \$USER	Host:\$THIS_HOST	Date:\$MYDATE
--------------	------------------	---------------

```

1 : ADD A RECORD
2 : VIEW A RECORD
3 : PAGE ALL RECORDS
4 : CHANGE A RECORD
5 : DELETE A RECORD
P : PRINT ALL RECORDS
H : Help screen
O : Exit Menu

```

```
MAYDAY
colour black_cyan
echo -e -n "\tYour Choice [1,2,3,4,5,P,H,Q] >"
read CHOICE
CHOICE=`get_char`
case $CHOICE in
1) ls
;;
2) vi
;;
3) who
;;
4) if user_level; then
ls -l |wc
else
restrict
fi
;;
5) if user_level; then
sort /etc/passwd
else
restrict
fi
;;
P|p) echo -e "\n\nPrinting records....."
;;
H|h)
tput clear
cat <<MAYDAY
This is the help screen, nothing here yet to help you!
MAYDAY
;;
Q|q) my_exit
;;
*) echo -e "\t\007unknown user response"
;;
esac
echo -e -n "\tHit the return key to continue"
read DUMMY
done
```

这种菜单可以在profile文件中用exec命令调用，用户不能够退出。它们将始终位于菜单或菜单子选项的应用里面。这对于只使用UNIX或LINUX应用而不关心shell的用户来说是一种好方法。

21.3 小结

使用tput命令可以增强应用外观及脚本的控制。颜色设置可以增加应用的专业性。注意使用颜色不要太过火，这也许对你来说很好，但其他用户使用这段脚本时看到这种屏幕设置也许会感到厌烦。可以使用和读取控制字符来增加脚本的灵活性，特别是对用户击键输入操作更是如此。