

第22章 创建屏幕输入

屏幕输入或数据输入是接受输入（这里指键盘）并验证其有效的能力。如果有效，接受它，如果无效，放弃该输入。

前面讲到了基于一些条件的测试函数，例如字符串长度、字符串是数字或字符型，这一章在脚本中将继续使用这些函数。

本章内容有：

- 验证有效输入。
- 增加、删除、修改和查看记录。
- 修改脚本的工作文件。

本章开始读起来可能有些累人，因此可以先大概看一下，再慢慢细看。验证有效性的代码量很大，这是因为捕获所有错误，脚本必须测试几乎所有可能的错误。

现在在创建一个一般文件以修改系统地过程中逐步实现每一个任务：增加、删除、修改和查看记录。这里也将创建一个个人文件以修改系统。记录文件 DBFILE 保存下列信息：

域	长 度	允 许 输 入	描 述
职员号码	10	数字	雇员代码
名	20	字符	雇员名
姓	20	字符	雇员姓
部门	-	记帐 IT 服务 销售 权利	雇员所在部门

域间用冒号：分隔，例如：

```
<Staff number>:<First name>:<Second name>:<Department>
```

每一个任务即是一个完整脚本。脚本中一小部分复制于前面的两个例子。这样做是因为本章主要用于显示怎样用文件修改系统。刚开始编写脚本时，最令人头疼的事就是将修改的文件或数据库系统放在一起后的文档清理工作。

运行脚本应具有一些菜单选项，与任务或模块相连接或包含在文件里与菜单脚本相关的一系列函数相连接。每一段脚本均执行 trap 命令，信号 2、3 和 15 被忽略。

22.1 增加记录

将记录加入文件，包含以下两个步骤：

- 1) 确认输入有效。
- 2) 将记录写入文件。

第一个任务就是将一些函数放在一起，这些函数测试域是字符型或数字型及域的长度限制，即数据输入有效性检验。有效性检验将用于增加数据和修改数据。幸运的是前面已经编

好这些函数，检测字符串及长度的函数脚本如下：

```
length_check()
{
# length_string
# $1=string, $2= length of string not to exceed this number
_STR=$1
_MAX=$2
_LENGTH=`echo $_STR |awk '{print length($0)}'`
if [ "$_LENGTH" -gt "$_MAX" ]; then
    return 1
else
    return 0
fi
}
```

检测字符串是否为数字型，函数脚本如下：

```
a_number()
#a _number
# $1= string
{
_NUM=$1
_NUM=`echo $1|awk '{if($0~/[^\0-9]/) print "1"}'`
if [ "$_NUM" != "" ]
then
    return 1
else
    return 0
fi
}
```

检测字符串是否为字符型，函数脚本如下：

```
characters()
# characters
# $1 = string
{
_LETTERS_ONLY=$1
_LETTERS_ONLY=`echo $1|awk '{if($0~/[^\a-zA-Z]/) print "1"}'`

if [ "$_LETTERS_ONLY" != "" ]
then
    return 1
else
    return 0
fi
}
```

当域读取完毕时，调用相应函数，测试其返回值。

这里也需要提示以保存屏幕信息直到键入某键删除这些信息，下列函数用到了

read_a_char函数。

```
continue_promptYN()
{
# continue_prompt
echo -n "Hit any key to continue.."
DUMMY=`read_a_char`
}
```

当用户输入雇员代码后，要确保编号还没有用到，此域必须唯一。测试这一点有几种方法，这里使用grep。grep搜寻字符串_CODE中的雇员编号，如果awk返回空值，则不存在匹配编号，函数返回状态码0。函数如下（这里在grep中使用“\$_CODE\>”抽取相应匹配，变量用双引号括起来，如果用单引号，则返回空值）：

```
check_duplicate()
{
# check_duplicate
# check for employee number duplicate
_CODE=$1
MATCH=`grep "$_CODE\>" $DBFILE`
echo $_CODE
if [ "$MATCH" = "" ]; then
    return 0 # no duplicate
else
    return 1 # duplicate found
fi
}
```

以下是检测雇员编号代码段，之后继续讲解其功能。

```
while :
do
echo -n "Employee Staff Number : "
read NUM
# check for input
if [ "$NUM" != "" ]; then
    if a_number $NUM; then
        # number OK
        NUM_PASS=0
    else
        NUM_PASS=1
    fi
    if length_check $NUM 10; then
        # length OK
        LEN_PASS=0
    else
        LEN_PASS=1
    fi
    # now check for duplicates...

    if check_duplicate $NUM; then
        # no duplicates
        DUPLICATE=0
    else
        DUPLICATE=1
        echo "Staff Number: There is already an employee with this number"
        continue_prompt
    fi
    # check all three variables now, they must all be true
    if [ "$LEN_PASS" = "0" -a "$NUM_PASS" = "0" -a "$DUPLICATE" = "0" ]
    then
        break
    else
        echo "Staff Number: Non-Numeric or Too Many Numbers In Field"
```

```

    continue_prompt
fi
else
    echo "Staff Number: No Input Detected, This Field Requires a Number"
    continue_prompt
fi

```

done

所有检测语句均在 while 循环中（实际上每一个数据输入域均在一单独的 while 循环中），如果没有有效数据，提示返回初始读位置。

读完雇员编号，继续检测域中数据是否存在：

```
if [ "$NUM" != "" ]
```

如果域中没有输入数据。则不执行 if 语句 then 部分。else 部分在脚本结尾部分，用于显示下列信息：

```
Staff Number: No Input Detected, This Field Requires a Number
```

then 部分包括对域输入数据的所有有效性检测。假定存在输入，调用 a_number 函数，测试传输字符串是否为一数字，如果是，函数返回 0，如果不是，函数返回 1。基于此返回值，设置指针 NUM_PASS 为 0，表示返回值正确（数字型），设置为 1，表示返回失败（非数字型）。

然后调用函数 length_check，参数为字符串及字符串包含字符最大数目。这里为 10，如果字符串长度小于 10，则返回 0，否则返回 1。指针 LEN_PASS 设置为 0，表示返回成功（长度不超过最大长度），设置为 1，表示返回失败（长度超出最大长度）。

接下来检测是否有重复雇员编号。调用函数 check_duplicate，如果没有发现重复编号，设置指针 DUPLICATE 为 0，最后测试三个指针变量均为 0（无错误），为此使用 AND 测试，如果条件同时成立，执行 then 部分语句。

如果测试通过，则存在有效域。这时处在 while 循环中，因此需要用 break 命令跳出循环。

```
if [ "$LEN_PASS" = "0" -a "$NUM_PASS" = "0" -a "$DUPLICATE" = "0"
]; then
break

```

如果有效性测试任何一部分失败，即长度测试或类型测试之一不通过，返回错误信息并显示在屏幕底部。

```
Staff Number : Non-Numeric or Too Many Numbers In Field
```

验证第 2 和第 3 域有效性，处理过程一样。有效性验证这次在另一个循环中。这次调用 characters 函数，检验域是否只包含字符。下述脚本做名字有效性检验：

```

while :
do
echo -n "Employee's First Name : "
read F_NAME
if [ "$F_NAME" != "" ]; then
    if characters $F_NAME; then
        F_NAME_PASS=0
    else
        F_NAME_PASS=1
    fi
    if length_check $F_NAME 20; then

```

```

        LEN_PASS=0
    else
        LEN_PASS=1
    fi
    if [ "$LEN_PASS" = "0" -a "$F_NAME_PASS" = "0" ]; then

        break
    else
        echo "Staff First Name: Non-Character or Too Many Characters In Field"
        continue_prompt
    fi
else
    echo "Staff First Name: No Input Detected, This Field Requires
        Characters"
    continue_prompt
fi
done

```

使用case语句检验部门域（列表见下面），因为公司只包含5个部门，部门域必须是其中之一。注意对每个部门有三个不同的匹配模式，可以由用户键入部门名称加以验证。如果找到匹配模式，用户跳出case语句，并显示有效部门列表。

```

while :
do
echo -n "Company Department   :"
read DEPART
case $DEPART in
ACCOUNTS|Accounts|accounts) break;;
SALES|Sales|sales)break;;
IT|It|it) break;;
CLAIMS|Claims|claims)break;;
SERVICES|Services|services)break;;
*) echo "Department: Accounts,Sales,IT,Claims,Services";;
esac
done

```

当所有域的有效性验证完成后，将提示用户是否保存此记录，这里使用函数continue_promptYN，前面讲过这个函数，它处理Y或N响应，用户也可以点击回车键表示缺省回答。

如果用户选择N，进入if语句代码段并退出脚本。如果用户输入许多记录，然后在while循环中调用函数以增加记录，这种方法将不会返回菜单或增加记录后退出。

如果用户选择Y，保存记录。将记录加入一个文件的脚本是：

```
echo "$NUM:$F_NAME:$S_NAME:$DEPART">>$DBFILE
```

然后显示信息通知用户记录已存入文件。sleep命令将脚本进程挂起1s，以使用户有足够的时间查看该信息。

域分隔符是冒号，文件以姓域分类，输出存入一临时工作文件，然后文件移入初始文件DBFILE。文件转移操作时，会测试其最后状态，如果存在问题，则通知用户该信息。

加入一个记录后，输出如下：

```
ADD A RECORD
```

```
Employee Staff Number :23233
```

```
Employee's First Name :Peter
Employee's Surname    :Wills
Company Department    :Accounts
```

```
Do You wish To Save This Record [Y..N] [Y]:
saved
```

增加记录后，DBFILE文件内容如下：

```
$ pg DBFILE
32123:Liam:Croad:Claims
2399:Piers:Cross:Accounts
239192:John:Long:Accounts
98211:Simon:Penny:Services
99202:Julie:Sittle:IT
23736:Peter:Wills:Accounts
89232:Louise:Wilson:Accounts
91811:Andy:Wools:IT
```

以下是增加一个记录的完整脚本：

```
$ pg dbase_add
#!/bin/sh
# dbase_add
# add a record
# ignore signals
trap "" 2 3 15
# temp hold files
DBFILE=DBFILE
HOLD1=HOLD1.$$

read_a_char()
{
# read_a_char
# save the settings
SAVEDSTTY=`stty -g`
stty cbreak
dd if=/dev/tty bs=1 count=1 2> /dev/null
stty -cbreak
stty $SAVEDSTTY
}

continue_promptYN()
#to call: continue_prompt "string to display" default_answer
{
# continue_prompt
_STR=$1
_DEFAULT=$2
# check we have the right params
if [ $# -lt 1 ]; then
echo "continue_prompt: I need a string to display"
return 1
fi
while :
do
echo -n "$_STR [Y..N][$_DEFAULT]:"
read _ANS
```

```

# if user hits return set the default and determine the return value
: ${_ANS:=$_DEFAULT}
if [ "$_ANS" = "" ]; then
    case $_ANS in
        Y) return 0 ;;
        N) return 1 ;;
    esac
fi
# user has selected something
case $_ANS in
    y|Y|Yes|YES)
        return 0
        ;;
    n|N|No|NO)
        return 1
        ;;
    *) echo "Answer either Y or N, default is $_DEFAULT"
        ;;
esac
echo $_ANS
done
}

continue_prompt()
{
# continue_prompt
echo -n "Hit any key to continue.."
DUMMY=`read_a_char`
}

length_check()
{
# length_check
# $1=str to check length $2=max length
_STR=$1
_MAX=$2
_LENGTH=`echo $_STR |awk '{print length($0)}'`
if [ "$_LENGTH" -gt "$_MAX" ]; then
    return 1
else
    return 0
fi
}

a_number()
{
# a_number
# call: a_number $1=number
_NUM=$1
_NUM=`echo $1|awk '{if($0~/[^\0-9]/) print "1"}'`
if [ "$_NUM" != "" ]
then
    # errors
    return 1
else

```

```

    return 0
fi
}
characters()
# characters
# to call: char_name string
{
    _LETTERS_ONLY=$1
    _LETTERS_ONLY=`echo $1|awk '{if($0~/[Aa-zA-Z]/) print "1"}'`
    if [ "$_LETTERS_ONLY" != "" ]
    then
        # oops errors
        return 1
    else
        # contains only chars
        return 0
    fi
}

check_duplicate()
{
    # check_duplicate
    # check for employee number duplicate
    # to call: check_duplicate string
    _CODE=$1
    MATCH=`grep "$_CODE>" $DBFILE`
    echo $_CODE
    if [ "$MATCH" = "" ]; then
        return 0 # no duplicate
    else
        return 1 # duplicate found
    fi
}

add_rec()
{
    # add_rec
    # == STAFF NUMBER
    while :
    do
        echo -n "Employee Staff Number :"
        read NUM
        if [ "$NUM" != "" ]; then
            if a_number $NUM; then
                NUM_PASS=0
            else
                NUM_PASS=1
            fi
            if length_check $NUM 10; then
                LEN_PASS=0
            else
                LEN_PASS=1
            fi
        fi
        # now check for duplicates...
        if check_duplicate $NUM; then

```



```
    DUPLICATE=0
else
    DUPLICATE=1
    echo "Staff Number: There is already a employee with this number"
    continue_prompt
fi
if [ "$LEN_PASS" = "0" -a "$NUM_PASS" = "0" -a "$DUPLICATE" = "0" ]
then
    break
else
    echo "Staff Number: Non-Numeric or Too Many Numbers In Field"
    continue_prompt
fi
else
    echo "Staff Number: No Input Detected, This Field Requires a Number"
    continue_prompt
fi
done

# == First Name
while :
do
    echo -n "Employee's First Name : "
    read F_NAME
    if [ "$F_NAME" != "" ]; then
        if characters $F_NAME; then
            F_NAME_PASS=0
        else
            F_NAME_PASS=1
        fi
        if length_check $F_NAME 20; then
            LEN_PASS=0
        else
            LEN_PASS=1
        fi
        # both conditons must be true to get out of this loop
        if [ "$LEN_PASS" = "0" -a "$F_NAME_PASS" = "0" ]; then
            break
        else
            echo "Staff First Name: Non-Character or Too Many Characters In
                Field"
            continue_prompt
        fi
    else
        echo "Staff First Name: No Input Detected, This Field Requires
            Characters"
        continue_prompt
    fi
done

# == Surname
while :
do
    echo -n "Employee's Surname : "
    read S_NAME
```



```

    echo "problems trying to sort the file..check it out"
    exit 1
fi
mv $HOLD1 $DBFILE
if [ $? -ne 0 ]; then
    echo "problems moving the temp sort file..check it out"
    exit 1
fi
else
    echo " record not saved"
    sleep 1
fi

```

22.2 删除记录

要从文件中删除记录，首先要将记录传给用户以确保该记录是正确删除的记录。得到确认后才执行下列任务：

- 1) 查询记录。
- 2) 显示记录。
- 3) 确认删除。
- 4) 修改文件。

首先使用姓域查询记录，一旦从用户处得到需查询的姓，则使用 `grep`或`awk`进行处理，但是因为此文件不会有超过 100个记录，将直接从文件中执行读取操作，进行匹配测试。

如果文件包括超过几百个记录，建议使用 `awk`，因为使用 `awk`比直接从文件中读取数据快得多，同样也比用 `grep`将各域分开存入变量要快一些。

可以使用`awk`或`grep`查询文件`DBFILE`：

```

echo "enter the surname to search "
read STR

```

```

# for awk use this
awk -F: '/$STR/' DBFILE

```

```

# for grep use this
grep "$STR" DBFILE

```

```

or
grep "$STR\" > "DBFILE

```

注意使用`awk`时，变量用单引号括起来，如果不这样做，就不会返回任何数据。

将各域分开，并设置与其对应的变量（记住这里用冒号作分隔符）。必须改变`IFS`设置为冒号。如果不这样，就不能读取记录。改变 `IFS`设置时，最好先保存其设置，以便于脚本完成后再恢复它。

要保存`IFS`，使用：

```

SAVEDIFS=$IFS

```

将其改为冒号：

```

IFS=:

```

当用`IFS`完成操作后，简单的恢复它：

```

IFS=$SAVEDIFS

```

查询记录函数为 `get_rec`，此函数不带参数。

```
get_rec()
{
# get_rec
clear
echo -n "Enter the employee surname : "
read STR
if [ "$STR" = "q" ]; then
    return 1
fi

REC=0
MATCH=no
if [ "$STR" != "" ]; then
    while read CODE F_NAME S_NAME DEPART
    do
        REC=`expr $REC + 1`
        tput cup 3 4
        echo -n " searching record.. $REC"
        if [ "$S_NAME" = "$STR" ]; then
            MATCH=yes
            display_rec
            break
        else
            continue
        fi
    done <DBFILE
else
    echo "Enter a surname to search for or q to quit"
fi
if [ "$MATCH" = "no" ]; then
    no_recs
fi
}
```

用户可以输入姓或q退出任务。一旦输入姓，执行测试以确保输入存在。比较好的测试方法是：

```
if [ "$STR" != "" ]; then
```

然后是：

```
[-2 $STR]
```

第一个测试捕获只键入回车符的用户。第二个测试 0 长度字符串。

使用有意义的变量名从文件中读取各域，读取记录时使用计数，通过计数变化告之用户查询记录时发生某种动作。如果发现匹配模式，调用另一过程显示此域，用户然后使用 `break` 命令跳出循环。如果未找到匹配模式，脚本进入下一循环步。找到匹配记录后，询问用户是否删除记录，缺省回答是 `no`。

```
if continue_promptYN "Do You Wish To DELETE This Record" "N"; then
echo "DEL"
grep -v $STR DBFILE > $HOLD1 2> /dev/null
if [ $? -ne 0 ]; then
    echo "Problems creating temp file $HOLD1..check it out"
```

```
exit 1
fi
...
```

使用grep -v 执行记录删除，并使用字符串 STR（STR保存用户删除的雇员姓）显示所有未匹配记录。

grep命令输出重定向到一临时工作文件中。然后文件移入初始 DBFILE中，删除后执行文件分类，输出重定向到一临时工作文件，然后再移回初始文件 DBFILE。临时工作文件可能首先被分类，然后再移回初始文件，而不是先移后分类。

使用最后状态命令执行测试所有文件移动操作。以下是删除一个记录的输出结果：

```
Enter the employee surname :Wilson
```

```
searching record.. 6
```

```
EMPLOYEE NO: 89232
FIRST NAME : Louise
SURNAME    : Wilson
DEPARTMENT : Accounts
```

```
Do You Wish To DELETE This Record [Y..N] [N]:
```

删除记录的完整脚本如下：

```
$ pg dbase_del
#!/bin/sh
# dbase_del
# delete a record

# trap signals
trap "" 2 3 15

# DATAFILE
DBFILE=DBFILE

# temp files
HOLD1=HOLD1.$$
HOLD2=HOLD2.$$

continue_promptYN()
{
# continue_prompt
_STR=$1
_DEFAULT=$2
# check we have the right params
if [ $# -lt 1 ]; then
echo "continue_prompt: I need a string to display"
return 1
fi
while :
do
echo -n "$_STR [Y..N] [$_DEFAULT]:"
read _ANS
: ${_ANS:=$_DEFAULT}
if [ "$_ANS" = "" ]; then
case $_ANS in
Y) return 0 ;;
N) return 1 ;;
```

```

    esac
fi
case $_ANS in
y|Y|Yes|YES)
    return 0
;;
n|N|No|NO)
    return 1
;;
*) echo "Answer either Y or N, default is $_DEFAULT"
;;
esac
done
}

display_rec()
{
# display_rec
# could use cat << here document
tput cup 5 3
echo "EMPLOYEE NO: $CODE"
echo "FIRST NAME : $F_NAME"
echo "SURNAME    : $S_NAME"
echo "DEPARTMENT : $DEPART"
echo -e "\n\n"
}

no_recs()
{
# no_recs
echo -e "\n\nSorry could not find a record with the name $STR"
}

get_rec()
{
# get_rec
clear
echo -n "Enter the employee surname : "
read STR
if [ "$STR" = "q" ]; then
    return 1
fi

REC=0
MATCH=no
if [ "$STR" != "" ]; then
    while read CODE F_NAME S_NAME DEPART
    do
        REC=`expr $REC + 1`
        tput cup 3 4
        echo -n " searching record.. $REC"
        if [ "$S_NAME" = "$STR" ]; then
            MATCH=yes
            display_rec
            break
        else

```

```

        continue
    fi
done $DBFILE
else
    echo "Enter a surname to search for or q to quit"
fi
if [ "$MATCH" = "no" ]; then
    no_recs
fi
}

SAVEDIFS=$IFS
IFS=:
get_rec
if [ "$MATCH" = "yes" ]; then
    if continue_promptYN "Do You Wish To DELETE This Record" "N"; then
        echo "DEL"
        grep -v $STR DBFILE >$HOLD1 2> /dev/null
        if [ $? -ne 0 ]; then
            echo "Problems creating temp file $HOLD1..check it out"
            exit 1
        fi
        mv $HOLD1 DBFILE
        if [ $? -ne 0 ]; then
            echo "Problems moving temp file..check it out"
            exit 1
        fi
        # sort the file after changes
        sort +2 -t: $DBFILE >$HOLD2 2> /dev/null
        if [ $? -ne 0 ]; then
            echo "problems trying to sort the file..check it out"
            exit 1
        fi
        mv $HOLD2 $DBFILE
        if [ $? -ne 0 ]; then
            echo "problems moving the temp sort file..check it out"
            exit 1
        fi
    else
        echo "no deletion"
        # no deletion
    fi # if wish to delete
fi # if match

# restore IFS settings
IFS=$SAVEDIFS

```

22.3 修改记录

实际上已经编写了修改记录的大部分脚本，这些脚本在记录删除操作中。

找到正确记录后，设置所有该记录域变量到一临时工作文件，然后使用缺省设置变量：

```
: {default_variable=variable}
```

对于不想修改的域，简单输入回车键即可，然后缺省值放入临时工作变量中。在相关域内键入新值即可修改此域。

```
echo -n -e "EMPLOYEE NO: $CODE\n"

echo -n "FIRST NAME : [$F_NAME] >"
read _F_NAME
: ${_FNAME:=$F_NAME}
...
```

使用grep -v执行文件的实际修改操作。除了正在被修改的记录，所有记录重定向到一临时工作文件，这里雇员编号用作grep命令字符串参数：

```
grep -v $CODE $DBFILE >$HOLD1
```

然后提示用户是否保存记录。如果保存，重新修改的记录也写入临时工作文件，然后此临时工作文件移入原文件DBFILE中。

```
echo "$CODE:$F_NAME:$S_NAME:$DEPART" >> $HOLD1
mv $HOLD1 $DBFILE
```

输出被重定向到一临时工作文件，然后将此文件分类，再移回原文件 DBFILE。最后状态命令测试文件移动操作。如果存在问题，告之用户。实际脚本中，在增加记录时执行的有效性测试也同样用于修改记录中。修改一个记录的输出结果如下：

```
Enter the employee surname :Penny
```

```
searching record.. 7
```

```
EMPLOYEE NO: 98211
FIRST NAME : Simon
SURNAME    : Penny
DEPARTMENT : Services
```

```
Is this the record you wish to amend [Y..N] [Y]:
```

```
amending
```

```
EMPLOYEE NO: 98211
```

```
FIRST NAME : [Simon] >
```

```
SURNAME    : [Penny] >
```

```
DEPARTMENT : [Services] >Accounts
```

```
Ready to save this record [Y..N] [Y]:
```

完整脚本如下：

```
$ pg dbasechange
# !/bin/sh
# dbasechange
# amend a record
```

```
# ignore signals
trap "" 2 3 15
```

```
# temp files
DBFILE=DBFILE
HOLD1=HOLD1.$$
HOLD2=HOLD2.$$
```

```
continue_promptYN()
```

```
{
```



```
# continue_prompt
_STR=$1
_DEFAULT=$2
# check we have the right params
if [ $# -lt 1 ]; then
    echo "continue_prompt: I need a string to display"
    return 1
fi
while :
do
    echo -n "$_STR [Y..N] [$_DEFAULT]:"
    read _ANS
    : ${_ANS:=$_DEFAULT}
    if [ "$_ANS" = "" ]; then
        case $_ANS in
            Y) return 0 ;;
            N) return 1 ;;
        esac
    fi
    case $_ANS in
        y|Y|Yes|YES)
            return 0
            ;;
        n|N|No|NO)
            return 1
            ;;
        *) echo "Answer either Y or N, default is $_DEFAULT"
            ;;
    esac
done
}

display_rec()
{
# display_rec
# could use cat << as a here document, but I won't
tput cup 5 3
echo "EMPLOYEE NO: $CODE"
echo "FIRST NAME : $F_NAME"
echo "SURNAME    : $S_NAME"
echo "DEPARTMENT : $DEPART"
echo -e "\n\n"
}

no_recs()
{
# no_recs
echo -e "\n\nSorry could not find a record with the name $STR"
}

get_rec()
{
# get_rec
clear
```

```

echo -n "Enter the employee surname : "
read STR
if [ "$STR" = "q" ]; then
    return 1
fi

REC=0
MATCH=no
if [ "$STR" != "" ]; then
    while read CODE F_NAME S_NAME DEPART
    do
        REC=`expr $REC + 1`
        tput cup 3 4
        echo -n " searching record.. $REC"
        if [ "$S_NAME" = "$STR" ]; then
            MATCH=yes
            display_rec
            break
        else
            continue
        fi
    done $DBFILE
else
    echo "Enter a surname to search for or q to quit"
fi
if [ "$MATCH" = "no" ]; then
    no_recs
fi
}

# main
SAVEDIFS=$IFS
IFS=:
get_rec
if [ "$MATCH" = "yes" ]; then
    if continue_promptYN "Is this the record you wish to amend" "Y"
    then
        echo "amending"
        # cannot change employee code
        echo -n -e "EMPLOYEE NO: $CODE\n"

        echo -n "FIRST NAME : [$F_NAME] >"
        read _F_NAME
        : ${_FNAME:=$F_NAME}

        echo -n "SURNAME : [$S_NAME] >"
        read _S_NAME
        : ${_S_NAME:=$S_NAME}

        echo -n "DEPARTMENT : [$DEPART] >"
        read _DEPART
        : ${_DEPART:=$DEPART}

        grep -v $CODE $DBFILE >$HOLD1
    fi
fi

```

```

if [ $? -ne 0 ]; then
    echo "Problems creating temporary file..check it out"
    exit 1
fi

if continue_promptYN "Ready to save this record" "Y"; then
    echo "$CODE:$_F_NAME:$_S_NAME:$_DEPART" >> $HOLD1
    mv $HOLD1 $DBFILE
    if [ $? -ne 0 ]; then
        echo "Problems moving temporary file...check it out"
    fi
    echo " Record Amended"
    # now sort the file after changes,
    # could have been done in one file movement of course
    sort +2 -t: $DBFILE >$HOLD2 2> /dev/null
    if [ $? -ne 0 ]; then
        echo "problems trying to sort the file..check it out"
        exit 1
    fi
    mv $HOLD2 $DBFILE
    if [ $? -ne 0 ]; then
        echo "problems moving the temp sort file..check it out"
        exit 1
    fi

    else #if amend aborted
        echo "Amend aborted"
        exit 0
    fi
    else # if no amend
        echo "no amending"
        # no deletion
    fi # if wish to delete
fi # if match

```

IFS=\$SAVEDIFS

22.4 查看记录

用户可能要查看所有记录或其中一部分。如果查看所有记录，使用 cat命令和awk，如果记录包含很多域，那么很有必要定量显示输出结果，使其对用户更加实用。

```

if [ "$STR" = "all" ]; then
    echo "Surname   Name   Employee Code"
    echo "_____"
    cat $DBFILE |awk -F: '{print $2"\t"$3"\t\t"$1}' | more
    return 0
fi

```

在删除和修改记录中，已经讲过了怎样显示单一记录，用户有选择的查看记录选项时唯一增加的功能就是打印一个记录。以下脚本段将记录发往打印机：

```
pr <<- MAYDAY
```

```

RECORD No      : $REC
EMPLOYEE NUMBER : $CODE

```

```
EMPLOYEE NAME      : $F_NAME
EMPLOYEE SURNAME    : $$_NAME
EMPLOYEE DEPARTMENT : $DEPART
MAYDAY
```

查看记录时输出结果如下：

```
Enter the employee surname to view or all for all records:Wilson
```

```
searching record.. 8
```

```
EMPLOYEE NO: 89232
FIRST NAME : Peter
SURNAME    : Wilson
DEPARTMENT : IT
```

```
Do You Wish To Print This Record [Y..N] [N]:
```

查看记录的完整脚本如下：

```
$ pg dbaseview
#!/bin/sh
# dbaseview
# view records

# ignore signals
trap "" 2 3 15

# temp files
HOLD1=HOLD1.$$
DBFILE=DBFILE

continue_promptYN()
{
# continue_prompt
_STR=$1
_DEFAULT=$2
# check we have the right params
if [ $# -lt 1 ]; then
    echo "continue_prompt: I need a string to display"
    return 1
fi
while :
do
    echo -n "$_STR [Y..N] [$_DEFAULT]:"
    read _ANS
    : ${_ANS:=$_DEFAULT}
    if [ "$_ANS" = "" ]; then
        case $_ANS in
            Y) return 0 ;;
            N) return 1 ;;
            esac
        fi
        case $_ANS in
            y|Y|Yes|YES)
                return 0
                ;;
            n|N|No|NO)
                return 1
            ;;
        esac
    fi
done
}
```

```

;;
*) echo "Answer either Y or N, default is $_DEFAULT"
;;
esac
done
}

display_rec()
{
# display_rec
# could use cat <<.
tput cup 5 3
echo "EMPLOYEE NO: $CODE"
echo "FIRST NAME : $F_NAME"
echo "SURNAME    : $S_NAME"
echo "DEPARTMENT : $DEPART"
echo -e "\n\n"
}

no_recs()
{
# no_rec
echo -e "\n\nSorry could not find a record with the name $STR"
}

get_rec()
{
# get_rec
clear
echo -n "Enter the employee surname to view or all for all records:"
read STR
if [ "$STR" = "q" ]; then
    return 1
fi
if [ "$STR" = "all" ]; then
    # view all recs
    echo "Surname   Name   Employee Code"
    echo "_____ "
    cat $DBFILE | awk -F: '{print $2"\t"$3"\t\t"$1}' | more
    return 0
fi
REC=0
MATCH=no
if [ "$STR" != "" ]; then
    while read CODE F_NAME S_NAME DEPART
    do
        REC=`expr $REC + 1`
        tput cup 3 4
        echo -n " searching record.. $REC"
        if [ "$S_NAME" = "$STR" ]; then
            # found name
            MATCH=yes
            display_rec
            break
        else
            continue
        fi
    done
fi

```

```
    fi
done <$DBFILE
else
    echo "Enter a surname to search for or q to quit"
fi
if [ "$MATCH" = "no" ]; then
    no_recs
fi
}

# main
SAVEDIFS=$IFS
IFS=:
get_rec
if [ "$MATCH" = "yes" ]; then
    if continue_promptYN "Do You Wish To Print This Record" "N"; then
        lpr <<- MAYDAY

RECORD No: $REC

EMPLOYEE NUMBER      : $CODE
EMPLOYEE NAME        : $F_NAME
EMPLOYEE SURNAME     : $S_NAME
EMPLOYEE DEPARTMENT : $DEPART

MAYDAY
else
    echo "No print of $S_NAME"
    # no print
fi # if wish to print
fi # if match

IFS=$SAVEDIFS
```

22.5 小结

验证用户输入的有效性很重要，也是一种高级技巧。虽然你可能有时知道记录接受输入的内容，但用户通常并不知道。

在计算机工业发展史上有一句老话：送进的是垃圾，出来的肯定是垃圾，但知道这一点时已经太晚了，意即如果没有在脚本中测试垃圾数据，就会输出垃圾信息。