第20章　向脚本传递参数

前面已经讲到如何使用特定变量 $1..$9向脚本传递参数。 $#用于统计传递参数的个数。可以创建一个usage语句，需要时可通知用户怎样以适当的调用参数调用脚本或函数。

本章内容有：

- shift。
- getopts。
- shift和getopts例子。

简单地说，下述脚本框架控制参数开始与停止。脚本需要两个参数，如果没有输入两个参数，那么产生一个usage语句。注意这里使用case语句处理输入脚本的不同参数。

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

usage()
{
echo "usage:‘basename $0‘ start|stop process name"
}

OPT=$1
PROCESSID=$1
if [ $# -ne 2 ]
then
  usage
  exit 1
fi
case $OPT in
start|Start) echo "Starting..$PROCESSID"
  #  some process to go here
  ;;
stop|Stop) echo "Stopping..$PROCESSID"
  # some process to go here
  ;;
*) usage
  ;;
;;
esac
```

执行脚本，输入以下参数，结果为：

```
$ opt start named
Starting..named

$ opt start
usage:opt start|stop process name
```

任何UNIX或LINUX命令均接受一般格式：

命令　选项　文件

选项部分最多可包含12个不同的值。上述脚本中，如果必须控制不同的命令选项，就要加入大量脚本。这里只控制两个选项：开始和停止。

幸运的是shell提供shift命令以帮助偏移选项，使用shift可以去除只使用$1到$9传递参数的限制。

20.1 shift命令

向脚本传递参数时，有时需要将每一个参数偏移以处理选项，这就是 shift命令的功能。它每次将参数位置向左偏移一位，下面用一段简单脚本详述其功能。脚本使用 while循环反馈所有传递到脚本的参数。

```
$ pg opt2
#!/bin/sh
# opt2
loop=0
while [ $# -ne 0 ]    # while there are still arguments
do
  echo $1
done
```

你可能想像，上述脚本一直执行，直到命令行中不再有更多的参数输入。错了，因为没有办法偏移到脚本中下一个参数，将只会反馈出第一个参数。执行结果如下：

```
$ opt2 file1 file2 file3
file1
file1
file1
...
```

20.1.1 shift命令简单用法

使用shift命令来处理传递到脚本的每一个参数。改动后脚本如下：

```
$ pg opt2
#!/bin/sh
# opt2
loop=0
while [ $# -ne 0 ]    # while there are still arguments
do
  echo $1
  shift
done
```

现在再执行，结果将会不同：

```
$ opt2 file1 file2 file3
file1
file2
file3
```

20.1.2 命令行输入的最后一个参数

虽然还没有讲eval命令, 如果需要知道命令行中输入的最后一个参数(通常是一个文件名), 可以有两种选择：使用命令eval echo \$$#；使用shift命令：shift 'expr $# -2'。

20.1.3 使用shift处理文件转换

shift可使控制命令行选项更加容易。下面构造一个转换脚本，使用 tr将文件名转换为大写或小写。

脚本选项为：

-l 用于小写转换。

-u 用于大写转换。

使用shift命令将脚本放在一起以控制 -l和-u选项。脚本的第一版本如下：

```
$ pg tr_case
!/bin/sh
# tr_case
# case conversion
usage()
{
# usage
echo "usage:`basename $0` -[l|u] file [files]" >&2
exit 1
}

if [ $# -eq 0 ]; then
  # no parameters passed !
  usage
fi

while [ $# -gt 0 ]
do
  case $1 in
  -u|-U) echo "-u option specified"
    # do any settings of variables here for lowercase then shift
    shift
    ;;
  -l|-L) echo "-l option specified"
    # do any settings of variables here for uppercase then shift
    shift
    ;;
  *) usage
    ;;
  esac
done
```

首先检查脚本是否有参数，如果没有，打印 usage语句，如果有需要处理的参数，使用case语句捕获每一个传送过来的选项。当处理完此选项后，使用 shift命令搜集命令行中下一选项，如果未发现匹配选项，打印usage语句。

当向脚本传递两个无效参数时，输出如下：

```
$ tr_case -u -l -k
-u option specified
-l option specified
usage:tr_case -[l|u] file [files]
```

下一步就是要用case语句处理选项后传递过来的文件名。为此需改动 case语句。case语句中捕获任意模式*应该为-*以允许传递无效选项，例如 -p或-q。

*模式也匹配传递过来的所有文件名，以便用 for循环处理每一个文件，这里也将使用 -f选项检测文件是否存在。

改动后的case语句如下：

```
case
...
-*) usage
  ;;
*)  if [ -f  $1 ]; then
     FILES=$FILES" " $1   # assign the filenames to a variable
    else
      echo "`basename $0` cannot find the file $1"
    fi
  shift  # get next one !
  ;;
esac
```

还需要指定与选项（-l，-u）相关的变量设置。这些变量是：

TRCASE 保存转换类型（大写或小写）。

EXT 所有文件转换后，大写文件名为.UC，小写为.LC，不保存初始文件状态。

OPT 如果给出此选项，设其为yes，否则为no。如果没有给出此选项，捕获此信息并反馈出来。

其他部分脚本用于实际转换处理，这里即 tr命令。tr命令放在case语句for循环中读取文件名进行处理的脚本末尾部分。

以下为完整脚本：

```
$ pg tr_case
!/bin/sh
# tr_case
# convert files to either upper or lower case
FILES=""
TRCASE=""
EXT=""
OPT=no

# gets called when a conversion fails
error_msg()
{
_FILENAME=$1
echo "`basename $0`: Error the conversion failed on $_FILENAME"
}

if [ $# -eq 0 ]
then
  echo "For more info try `basename $0` --help"
  exit 1
fi
while [ $# -gt 0 ]
do
  case $1 in
  # set the variables based on what option was used
  -u) TRCASE=upper
    EXT=".UC"
```

```
  OPT=yes
  shift
  ;;
-l) TRCASE=lower
  EXT=".LC"
  OPT=yes
  shift
  ;;
-help) echo "convert a file(s) to uppercase from lowercase"
        echo "convert a file(s) from lowercase to uppercase"
        echo "will convert all characters according to the"
        echo " specified command option."
        echo " Where option is"
        echo " -l Convert to lowercase"
        echo " -u Convert to uppercase"
        echo " The original file(s) is not touched. A new file(s)"
        echo "will be created with either a .UC or .LC extension"
        echo "usage: $0 -[l|u] file [file..]"
  exit 0
  ;;
-*) echo "usage: `basename $0` -[l|u] file [file..]"
  exit 1
  ;;

*) # collect the files to process
  if [ -f $1 ]
     then
       # add the filenames to a variable list
       FILES=$FILES" "$1
     else
       echo "`basename $0`: Error cannot find the file $1"
     fi
     shift
     ;;
   esac

done
# no options given ... help the user
if [ "$OPT" = "no" ]
then
  echo "`basename $0`:Error you need to specify an option. No action taken"
  echo " try `basename $0` --help"
  exit 1
fi

# now read in all the file(s)
# use the variable LOOP, I just love the word LOOP
for LOOP in $FILES
do
  case $TRCASE in
  lower) cat $LOOP|tr "[a-z]" "[A-Z]" >$LOOP$EXT
    if [ $? != 0 ]
    then
      error_msg $LOOP
```

```
        else
          echo "Converted file called $LOOP$EXT"
        fi
        ;;
      upper) cat $LOOP|tr "[A-Z]" "[a-z]" >$LOOP$EXT
        if [ $? != 0 ]
        then
          error_msg $LOOP
        else
          echo "Converted file called $LOOP$EXT"
        fi
        ;;
      esac
  done
```

执行上述脚本，给出不同选项，得结果如下：

转换一个不存在的文件：

```
$ tr_case -k cursor
usage: shift1 -[l|u] file [file..]
```

传递不正确选项：

```
$ tr_case cursor
tr_case:Error you need to specify an option. No action taken
 try tr_case -help
```

只键入文件名，希望脚本提示更多帮助信息：

```
$ tr_case
For more info try tr_case -help
```

输入两个有效文件及第三个无效文件：

```
$ tr_case -l cursor sd ascii
tr_case: Error cannot find the file sd
Converted file called cursor.LC
Converted file called ascii.LC
```

使用上述脚本可以将许多文件转换为同样的格式。编写一段脚本，使其控制不同的命令行选项，这种方式编程量很大，是一件令人头疼的事。

假定要写一段脚本，要求控制以下各种不同的命令行选项：

```
命令-l -c 23 -文件1文件2
```

shift命令显得力不从心，这就需要用到getopts命令。

20.2 getopts

getopts可以编写脚本，使控制多个命令行参数更加容易。 getopts用于形成命令行处理标准形式。原则上讲，脚本应具有确认带有多个选项的命令文件标准格式的能力。

20.2.1 getopts脚本实例

通过例子可以更好地理解getopts。以下getopts脚本接受下列选项或参数。

- a 设置变量ALL为true。
- h 设置变量HELP为true。

- f 设置变量FILE为true。
- v 设置变量VERBOSE为true。

对于所有变量设置，一般总假定其初始状态为 false：

```
$ pg getopt1
!/bin/sh
#getopt1

# set the vars
ALL=false
HELP=false
FILE=false
VERBOSE=false

while getopts ahfgv OPTION
do
  case $OPTION in
  a)ALL=true
    echo "ALL is $ALL"
    ;;
  h)HELP=true
    echo "HELP is $HELP"
    ;;
  f)FILE=true
    echo "FILE is $FILE"
    ;;
  v)VERBOSE=true
    echo "VERBOSE is $VERBOSE"
    ;;
  esac
done
```

getopts一般格式为：

```
getopts  option_string variable
```

在上述例子中使用脚本：

```
while getopts ahfgv OPTION
```

可以看出while循环用于读取命令行，option_string为指定的5个选项（-a，-h，-f，-g，-v），脚本中variable为OPTION。注意这里并没有用连字符指定每一单个选项。

运行上述脚本，给出几个有效和无效的选项，结果为：

```
$ getopt1 -a -h
ALL is true
HELP is true
$ getopt1 -ah
ALL is true
HELP is true

$ getopt1 -a -h -p
ALL is true
HELP is true
./getopt1: illegal option -- p
```

可以看出不同选项的结合方式。

20.2.2 getopts使用方式

getopts读取option_string，获知脚本中使用了有效选项。

getopts查看所有以连字符开头的参数，将其视为选项，如果输入选项，将把这与option_string对比，如果匹配发现，变量设置为 OPTION，如果未发现匹配字符，变量能够设置为？。重复此处理过程直到选项输入完毕。

getopts接收完所有参数后，返回非零状态，意即参数传递成功，变量 OPTION保存最后处理参数，一会儿就可以看出处理过程中这样做的好处。

20.2.3 使用getopts指定变量取值

有时有必要在脚本中指定命令行选项取值。 getopts为此提供了一种方式，即在option_string中将一个冒号放在选项后。例如：

```
getopts ahfvc: OPTION
```

上面一行脚本指出，选项 a、h、f、v可以不加实际值进行传递，而选项 c必须取值。使用选项取值时，必须使用变量 OPTARG保存该值。如果试图不取值传递此选项，会返回一个错误信息。错误信息提示并不明确，因此可以用自己的反馈信息屏蔽它，方法如下：

将冒号放在option_string开始部分。

```
while getopts :ahfgvc:  OPTION
```

在case语句里使用？创建一可用语句捕获错误。

```
case
...
...
\?)   # usage statement
  echo "`basename $0` -[a h f v] -[c value] file"
  ;;
esac
```

改动后getopts脚本如下：

```
$ pg getopt1
#!/bin/sh
#getopt1

# set the vars
ALL=false
HELP=false
FILE=false
VERBOSE=false
COPIES=0    # the value for the -c option is set to zero

while getopts :ahfgvc: OPTION
do
  case $OPTION in
  a)ALL=true
    echo "ALL is $ALL"
    ;;
  h)HELP=true
    echo "HELP is $HELP"
    ;;
```

```
    f)FILE=true
      echo "FILE is $FILE"
      ;;
    v)VERBOSE=true
      echo "VERBOSE is $VERBOSE"
      ;;
    c) COPIES=$OPTARG
      echo "COPIES is $COPIES"
    \?)    # usage statement
      echo "`basename $0` -[a h f v] -[c value] file" >&2
      ;;
    esac
 done
```

运行上述脚本，选项-c不赋值，将返回错误，但显示的是脚本语句中的反馈信息：

```
$ getopt1 -ah -c
ALL is true
HELP is true
getopt1 -[a h f v] -[c value] file
```

现在，输入所有合法选项：

```
$ getopt1 -ah -c 3
ALL is true
HELP is true
COPIES is 3
```

20.2.4 访问取值方式

getopts的一种功能是运行后台脚本。这样可以使用户加入选项，指定不同的磁带设备以备份数据。使用getopts实现此任务的基本框架如下：

```
$ pg backups
#!/bin/sh
# backups
QUITE=n
DEVICE=awa
LOGFILE=/tmp/logbackup
usage()
{
echo "Usage: `basename $0` -d [device] -l [logfile] -q"
exit 1
}
if [ $# = 0 ]
then
  usage
fi

while getopts :qd:l: OPTION
do
  case $OPTION in
  q) QUIET=y
    LOGFILE="/tmp/backup.log"
    ;;
  d) DEVICE=$OPTARG
    ;;
  l) LOGFILE=$OPTARG
```

```
    ;;
  \?) usage
    ;;
  esac
done
echo "you chose the following options..I can now process these"
echo "Quite= $QUITE $DEVICE $LOGFILE"
```

上述脚本中如果指定选项d，则需为其赋值。该值为磁带设备路径。用户也可以指定是否备份输出到登录文件中的内容。运行上述脚本，指定下列输入：

```
$ backups -d/dev/rmt0 -q
you chose the following options..I can now process these
Quite= y /dev/rmt0 /tmp/backup.log
```

getopts检查完之后，变量OPTARG取值可用来进行任何正常的处理过程。当然，如果输入选项，怎样进行进一步处理及使该选项有有效值，完全取决于用户。

以上是使用getopts对命令行参数处理的基本框架。

实际处理文件时，使用for循环，就像在tr-case脚本中使用shift命令过滤所有选项一样。

使用getopts与使用shift方法比较起来，会减少大量的编程工作。

20.2.5 使用getopts处理文件转换

现在用所学知识将tr-case脚本转换为getopts版本。命令行选项getopts方法与shift方法的唯一区别是一个VERBOSE选项。

变量VERBOSE缺省取值为no，但选择了命令行选项后， case语句将捕获它，并将其设为yes，反馈的命令是一个简单的if语句。

```
if [ "VERBOSE" = "on" ]; then
  echo "doing upper on $LOOP..newfile called $LOOP$EXT"
fi
```

如果正在使用其他系统命令包，它总是反馈用户动作，只需简单地将包含错误的输出重定向到/dev/null中即可。如：

命令 `>/dev/null 2 >&1`

缺省时VERBOSE关闭（即不显示），使用-v选项可将其打开。例如要用 VERBOSE将myfile文件系列转换为小写，方法如下：

```
tr-case -l -v myfile1 myfile2 ...
```

或者

```
tr-case -v -l myfile1 myfile2 ...
```

可能首先注意的是使用 getopts后脚本的缩减效果。这里用于文件处理的脚本与 shift版本相同。

脚本如下：

```
$ pg tr_case2
#!/bin/sh
#tr_case2
# convert case, using getopts
EXT=""
TRCASE=""
FLAG=""
```

```
OPT="no"
VERBOSE="off"


while getopts :luv OPTION
do
  case $OPTION in
  l) TRCASE="lower"
    EXT=".LC"
    OPT=yes
    ;;
  u) TRCASE="upper"
    EXT=".UC"
    OPT=yes
    ;;
  v) VERBOSE=on
    ;;
  \?) echo "usage: `basename $0`: -[l|u] --v file[s]"
    exit 1 ;;
  esac
done
# next argument down only please
shift `expr $OPTIND - 1`
# are there any arguments passed ???
if [ "$#" = "0" ] || [ "$OPT" = "no" ]
then
  echo "usage: `basename $0`: -[l|u] -v file[s]" >&2
  exit 1
fi
for LOOP in "$@"
do
  if [ ! -f $LOOP ]
  then
    echo "`basename $0`: Error cannot find file $LOOP" >&2
    exit 1
  fi
  echo $TRCASE $LOOP
  case $TRCASE in
  lower) if [ "VERBOSE" = "on" ]; then
           echo "doing..lower on $LOOP..newfile called $LOOP$EXT"
         fi
    cat $LOOP | tr "[a-z]" "[A-Z]" >$LOOP$EXT
    ;;
  upper) if [ "VERBOSE" = "on" ]; then
           echo "doing upper on $LOOP..newfile called $LOOP$EXT"
         fi
    cat $LOOP | tr "[A-Z]" "[a-z]" >$LOOP$EXT
    ;;
  esac
done
```

在脚本中指定命令行选项时，最好使其命名规则与 UNIX或LINUX一致。下面是一些选项及其含义的列表。

选　项	含　义
-a	扩展
-c	计数、拷贝
-d	目录、设备
-e	执行
-f	文件名、强制
-h	帮助
-i	忽略状态
-l	注册文件
-o	完整输出
-q	退出
-p	路径
-v	显示方式或版本

20.3 小结

正确控制命令行选项会使脚本更加专业化，对于用户来说会使之看起来像一个系统命令。本章讲到了控制命令行选项的两种方法， shift和getopts。使用getopts检测脚本的数量远远小于使用shift方法检测脚本的数量。

shift也克服了脚本参数$1..$9的限制。使用shift命令，脚本可以很容易偏移至所有调用参数，因此脚本可以做进一步处理。