第14章　环境和shell变量

为使shell编程更有效，系统提供了一些 shell变量。shell变量可以保存诸如路径名、文件名或者一个数字这样的变量名。 shell将其中任何设置都看做文本字符串。

有两种变量，本地和环境。严格地说可以有 4种，但其余两种是只读的，可以认为是特殊变量，它用于向shell脚本传递参数。

本章内容有：

- shell变量。
- 环境变量。
- 变量替换。
- 导出变量。
- 特定变量。
- 向脚本传递信息。
- 在系统命令行下使用位置参数。

14.1　什么是shell变量

变量可以定制用户本身的工作环境。使用变量可以保存有用信息，使系统获知用户相关设置。变量也用于保存暂时信息。例如：一变量为 EDITOR，系统中有许多编辑工具，但哪一个适用于系统呢？将此编辑器名称赋给 EDITOR，这样，在使用 cron或其他需要编辑器的应用时，这就是你将一直使用的EDITOR取值，并将之用作缺省编辑器。

下面是一个例子，登录的审核系统需要编辑。在菜单中选择此选项时，应用查询 EDITOR变量值，其值为vi。系统知道可使用此编辑器。

另一个例子需要登录数据库系统，键入下列命令：

```
$ isql -Udavet -Pabcd -Smethsys
```

这里-S为正在连接的服务器名称。有一变量 DSQUERY保存服务器名称值。设置服务器名称值到DSQUERY变量，这样如果登录时不使用 -S提供服务器名称，应用将查询 DSQUERY变量，并使用其取值作为服务器名称。需要做的全部工作就是键入下列命令：

```
$ isql -Udavet -Pabcd
```

工作方式同上例。

14.2　本地变量

本地变量在用户现在的 shell生命期的脚本中使用。例如，本地变量 file-name取值为loop.doc，这个值只在用户当前 shell生命期有意义。如果在 shell中启动另一个进程或退出，此值将无效。这个方法的优点就是用户不能对其他的 shell或进程设置此变量有效。

表14-1列出各种实际变量模式

使用变量时，如果用花括号将之括起来，可以防止 shell误解变量值，尽管不必一定要这

样做，但这确实可用。

要设置一本地变量，格式为：

```
$ variable-name=value 或 ${variable-name=value}
```

注意，等号两边可以有空格。如果取值包含空格，必须用双引号括起来。 shell变量可以用大小写字母。

表14-1 变量设置时的不同模式

Variable-name=value	设置实际值到variable-name
Variable-name+value	如果设置了variable-name，则重设其值
Variable-name:?value	如果未设置variable-name，显示未定义用户错误信息
Variable-name?value	如果未设置variable-name，显示系统错误信息
Variable-name:=value	如果未设置variable-name，设置其值
Variable-name:-value	同上，但是取值并不设置到variable-name，可以被替换

14.2.1 显示变量

使用echo命令可以显示单个变量取值，并在变量名前加 $，例如：

```
$ GREAT_PICTURE="die hard"
$ echo ${GREAT_PICTURE}
die hard

$ DOLLAR=99
$ echo ${DOLLAR}
99

$ LAST_FILE=ZLPSO.txt
$ echo ${LAST_FILE}
ZLPSO.txt
```

可以结合使用变量，下面将错误信息和环境变量 LOGNAME设置到变量error-msg。

```
$ ERROR_MSG=" Sorry this file does not exist user $LOGNAME"
$ echo ${ERROR_MSG}
Sorry this file does not exist user dave
```

上面例子中，shell首先显示文本，然后查找变量 $LOGNAME，最后扩展变量以显示整个变量值。

14.2.2 清除变量

使用unset命令清除变量。

```
unset variable-name
```

```
$ PC=enterprise
$ echo ${PC}
enterprise
$ unset PC
$ echo ${PC}
$
```

14.2.3 显示所有本地shell变量

使用set命令显示所有本地定义的shell变量。

```
$ set
...
PWD=/root
SHELL=/bin/sh
SHLVL=1
TERM=vt100
UID=7
USER=dave
dollar=99
great_picture=die hard
last_file=ZLPSO.txt
```

set输出可能很长。查看输出时可以看出 shell已经设置了一些用户变量以使工作环境更加容易使用。

14.2.4 结合变量值

将变量并排可以使变量结合在一起：

```
echo ${variable_name}${variable_name}...
```

```
$ FIRST="Bruce "
$ SURNAME=Willis
$ echo ${FIRST}${SURNAME}
Bruce Willis
```

14.2.5 测试变量是否已经设置

有时要测试是否已设置或初始化变量。如果未设置或初始化，就可以使用另一值。此命令格式为：

```
${variable:-value}
```

意即如果设置了变量值，则使用它，如果未设置，则取新值。例如：

```
$ COLOUR=blue
$ echo "The sky is ${COLOUR:-grey} today"
The sky is blue today
```

变量colour取值blue，echo打印变量colour时，首先查看其是否已赋值，如果查到，则使用该值。现在清除该值，再来看看结果。

```
$ COLOUR=blue
$ unset COLOUR
$ echo "The sky is ${COLOUR:-grey} today"
The sky is grey today
```

上面的例子并没有将实际值传给变量，需使用下述命令完成此功能：

```
${variable:=value}
```

下面是一个更实用的例子。查询工资清单应用的运行时间及清单类型。在运行时间及类型输入时，敲回车键表明用户并没有设置两个变量值，将使用缺省值（ 03:00和Weekly），并传入at命令中以按时启动作业。

```
$ pg vartest
#!/bin/sh
# vartest
echo  "what time do you wish to start the payroll [03:00]:"
read TIME
```

```
echo " process to start at ${TIME:=03:00} OK"
echo  "Is it a monthly or weekly run [Weekly]:"
read RUN_TYPE
echo "Run type is ${RUN_TYPE:=Weekly}"
at -f $RUN_TYPE $TIME
```

在输入域敲回车键，输出结果如下：

```
$ vartest
what time do you wish to start the payroll [03:00]:
 process to start at 03:00 OK
Is it a monthly or weekly run [Weekly]:
Run type is Weekly

warning: commands will be executed using /bin/sh
job 15 at 1999-05-14 03:00
```

也可以编写脚本测试变量是否取值，然后返回带有系统错误信息的结果。下面的例子测试变量file是否取值。

```
$ echo "The file is ${FILES:?}"
sh: files: parameter null or not set
```

以上结果可读性不好，但是可以加入自己的脚本以增加可读性。

```
$ echo "The file is ${FILES:?" sorry cannot locate the variable files"}"

sh: files:  sorry cannot locate the variable files
```

测试变量是否取值，如果未设置，则返回一空串。方法如下：

```
${variable:+value}
```

使用下述方法初始化变量为空字符串。

```
variable=""
$DETINATION=""
```

14.2.6 使用变量来保存系统命令参数

可以用变量保存系统命令参数的替换信息。下面的例子使用变量保存文件拷贝的文件名信息。变量source保存passwd文件的路径，dest保存cp命令中文件目标。

```
$ SOURCE="/etc/passwd"
$ DEST="/tmp/passwd.bak"
$ cp ${SOURCE} ${DEST}
```

下面例子中，变量device保存磁带设备路径，然后用于在mt命令中倒带。

```
$ DEVICE="/dev/rmt/0n"
$ mt -f ${DEVICE} rewind
```

14.2.7 设置只读变量

如果设置变量时，不想再改变其值，可以将之设置为只读方式。如果有人包括用户本人想要改变它，则返回错误信息。格式如下：

```
variable-name=value
readonly variable-name
```

下面的例子中，设置变量为系统磁带设备之一的设备路径，将之设为只读，任何改变其

值的操作将返回错误信息。

```
$ TAPE_DEV="/dev/rmt/0n"
$ echo ${TAPE_DEV}
/dev/rmt/0n
$ readonly TAPE_DEV
$ TAPE_DEV="/dev/rmt/1n"
sh: TAPE_DEV: read-only variable
```

要查看所有只读变量，使用命令readonly即可。

```
$ readonly
declare -r FILM="Crimson Tide"
declare -ri PPID="1"
declare -r TAPE_DEV="/dev/rmt/0n"
declare -ri UID="0"
```

14.3 环境变量

环境变量用于所有用户进程（经常称为子进程）。登录进程称为父进程。shell中执行的用户进程均称为子进程。不像本地变量（只用于现在的 shell）环境变量可用于所有子进程，这包括编辑器、脚本和应用。

环境变量可以在命令行中设置，但用户注销时这些值将丢失，因此最好在 .profile文件中定义。系统管理员可能在 /etc/profile文件中已经设置了一些环境变量。将之放入 profile文件意味着每次登录时这些值都将被初始化。

传统上，所有环境变量均为大写。环境变量应用于用户进程前，必须用 export命令导出。环境变量与本地变量设置方式相同。

14.3.1 设置环境变量

```
VARIABLE-NAME=value；export VARIABLE-NAME
```

在两个命令之间是一个分号，也可以这样写：

```
VARIABLE-NAME=value
Export VARIABLE-NAME
```

14.3.2 显示环境变量

显示环境变量与显示本地变量一样，例子如下：

```
$ CONSOLE=tty1; export CONSOLE
$ echo $CONSOLE
tty1

$ MYAPPS=/usr/local/application; export MYAPPS
$ echo $MYAPPS
/usr/local/application
```

使用env命令可以查看所有的环境变量。

```
$ env
HISTSIZE=1000
HOSTNAME=localhost.localdomain
LOGNAME=dave
MAIL=/var/spool/mail/root
TERM=vt100
HOSTTYPE=i386
```

```
PATH=/sbin:/bin:/usr/sbin:/usr/bin:/usr/X11R6/bin:/root/bin:
CONSOLE=tty1
HOME=/home/dave
ASD=sdf
SHELL=/bin/sh
PS1=$
USER=dave
...
```

14.3.3 清除环境变量

使用unset命令清除环境变量：

```
$ unset MYAPPS
$ echo $MYAPPS

$
```

14.3.4 嵌入shell变量

Brourne shell有一些预留的环境变量名，这些变量名不能用作其他用途。通常在/etc/profile中建立这些嵌入的环境变量，但也不完全是，这取决于用户自己。以下是嵌入 shell变量列表。

1. CDPATH

改变目录路径变量，保留一系列由冒号隔开的路径名，用于cd命令。如果设置了CDPATH，cd一个目录时，首先查找CDPATH，如果CDPATH指明此目录，则此目录成为当前工作目录。例子如下：

```
$ CDPATH=:/home/dave/bin:/usr/local;export CDPATH.
```

如果要

```
$ cd apps
```

cd命令首先在CDPATH中查找目录列表，如果发现apps，则它成为当前工作目录。

2. EXINIT

EXINIT变量保存使用vi编辑器时的初始化选项。例如，调用 vi时，要显示行号，且在第10个空格加入tab键，命令为：

```
$ EXINIT='set nu tab=10';export EXINIT
```

3. HOME

HOME目录，通常定位于 passwd文件的倒数第 2列，用于保存用户自身文件。设置了HOME目录，可以简单使用cd命令进入它。

```
$ HOME=/home/dave; export HOME
$ pwd
$ /usr/local
$ cd
$ pwd
$ /home/dave
```

也可以用

```
$ cd $ HOME
```

4. IFS

IFS用作shell指定的缺省域分隔符。原理上讲域分隔符可以是任意字符，但缺省通常为空格、新行或tab键。IFS在分隔文件或变量中各域时很有用。下面的例子将 IFS设置为冒号，然后echo PATH变量，给出一个目录分隔开来的可读性很强的路径列表。

```
$ export IFS=:
$ echo $PATH
/sbin /bin /usr/sbin /usr/bin /usr/X11R6/bin /root/bin
```

要设置其返回初始设置：

```
$ IFS=<space><tab>; export IFS
```

这里<space><tab>为空格和tab键。

5. LOGNAME

此变量保存登录名，应该为缺省设置，但如果没有设置，可使用下面命令完成它：

```
$ LOGNAME=`whoami`; export LOGNAME
$ echo $LOGNAME
dave
```

6. MAIL

MAIL变量保存邮箱路径名，缺省为 /var/spool/mail/<login name>。shell周期性检查新邮件，如果有了新邮件，在命令行会出现一个提示信息。如果邮箱并不在以上指定位置，可以用MAIL设置。

```
$ MAIL=/usr/mail/dave; export MAIL
```

7. MAILCHECK

MAILCHECK缺省每60s检查新邮件，但如果不想如此频繁检查新邮件，比如设为每 2m，使用命令：

```
$ MAILCHECK=120; export MAILCHECK
```

8. MAILPATH

如果有多个邮箱要用到MAILPATH，此变量设置将覆盖MAIL设置。

```
$ MAILPATH=/var/spool/dave:/var/spool/admin; export MAILPATH
```

上面的例子中，MAIL检测邮箱dave和admin。

9. PATH

PATH变量保存进行命令或脚本查找的目录顺序，正确排列这个次序很重要，可以在执行命令时节省时间。你一定不想在已知命令不存在的目录下去查找它。通常情况，最好首先放在HOME目录下，接下来是从最常用到一般使用到不常用的目录列表次序。如果要在当前工作目录下查询，无论在哪儿，均可以使用句点操作。目录间用冒号分隔，例如：

```
$ PATH=$HOME/bin:.:/bin:/usr/bin; export PATH
```

使用上面的例子首先查找 HOME/bin目录，然后是当前工作目录，然后是 /bin，最后是/usr/bin。

PATH可以在系统目录下/etc/profile中设置，也可以使用下面方法加入自己的查找目录。

```
$ PATH=$PATH:/$HOME/bin; export PATH
```

这里使用了/etc/profile中定义的PATH，并加入$HOME/bin和当前工作目录。一般来说，在查找路径开始使用当前工作目录不是一个好办法，这样很容易被其他用户发现。

10. PS1

基本提示符包含shell提示符，缺省对超级用户为 #，其他为$。可以使用任何符号作提示

符，以下为两个例子：

```
$ PS1="star trek:"; export PS1
star trek:
$ PS1="->" ; export PS1
->
```

11. PS2

PS2为附属提示符，缺省为符号> 。PS2用于执行多行命令或超过一行的一个命令。

```
$ PS2="@:"; export PS2
$ for loop in *
@:do
@:echo $loop
...
```

12. SHELL

SHELL变量保存缺省shell，通常在/etc/passwd中已设置，但是如有必要使用另一个 shell，可以用如下方法覆盖当前shell：

```
$ echo $SHELL
/bin/sh
```

13. TERMINFO

终端初始化变量保存终端配置文件的位置。通常在 /usr/lib/terminfo或/usr/share/terminfo

```
$ TERMINFO=/usr/lib/terminfo;export TERMINFO
```

14. TERM

TERM变量保存终端类型。设置TERM使应用获知终端对屏幕和键盘响应的控制序列类型，常用的有vt100、vt200、vt220-8等。

```
$ TERM=vt100;export TERM
```

15. TZ

时区变量保存时区值，只有系统管理员才可以更改此设置。例如：

```
$ echo $TZ
GMT2EDT
```

返回值表明正在使用格林威治标准时间，与 GMT时差为0，并作EDT保存。

14.3.5 其他环境变量

还有一些预留的环境变量。其他系统或命令行应用将用到它们。以下是最常用的一些，注意这些值均未有缺省设置，必须显示说明。

1. EDITOR

设置编辑器，最常用。

```
$ EDITOR=vi;export EDITOR
```

2. PWD

当前目录路径名，用cd命令设置此选项。

3. PAGER

保存屏幕翻页命令，如pg、more，在查看man文本时用到此功能。

```
$ PAGER='pg -f -p%d';export PAGER
```

4. MANPATH

保存系统上man文本的目录。目录间用冒号分隔。

```
$ MANPATH=/usr/apps/man:/usr/local/man;export MANPATH
```

5. LPDEST或PRINTER

保存缺省打印机名，用于打印作业时指定打印机名。

```
$ LPDEST=hp3si-systems
```

14.3.6 set命令

在$HOME.profile文件中设置环境变量时，还有另一种方法导出这些变量。使用 set命令-a选项，即set -a指明所有变量直接被导出。不要在 /etc/profile中使用这种方法，最好只在自己的$HOME.profile文件中使用。

```
$ pg .profile
#.profile
set -a
MAIL=/usr/mail/${LOGNAME:?}
PATH=$PATH:$HOME:bin
#
EDITOR=vi
TERM vt220
ADMIN=/usr/adm
PS1="`hostname`>"
```

14.3.7 将变量导出到子进程

shell新用户碰到的问题之一是定义的变量如何导出到子进程。前面已经讨论过环境变量的工作方式，现在用脚本实现它，并在脚本中调用另一脚本（这实际上创建了一个子进程）。

以下是两个脚本列表father和child。

father脚本设置变量film，取值为A Few Good Men，并将变量信息返回屏幕，然后调用脚本child，这段脚本显示第一个脚本里的变量 film，然后改变其值为Die Hard，再将其显示在屏幕上，最后控制返回father脚本，再次显示这个变量。

```
$ pg father
#!/bin/sh
# father script.
echo "this is the father"
FILM="A Few Good Men"
echo "I like the film :$FILM"
# call the child script
child
echo "back to father"
echo "and the film is :$FILM"

$ pg child
#!/bin/sh
# child
echo "called from father..i am the child"
echo "film name is :$FILM"
FILM="Die Hard"
echo "changing film to :$FILM"
```

看看脚本显示结果。

```
$ father
this is the father
I like the film :A Few Good Men
called from father..i am the child
film name is :
changing film to :Die Hard
back to father
and the film is :A Few Good Men
```

因为在father中并未导出变量film，因此child脚本不能将film变量返回。

如果在father脚本中加入export命令，以便child脚本知道film变量的取值，这就会工作：

```
pg father
#!/bin/sh
# father script.
echo "this is the father"
FILM="A Few Good Men"
echo "I like the film :$FILM"
# call the child script
# but export variable first
export FILM
child
echo "back to father"
echo "and the film is :$FILM"

$ father2
this is the father
I like the film :A Few Good Men
called from father..i am the child
film name is :A Few Good Men
changing film to :Die Hard
back to father
and the film is :A Few Good Men
```

因为在脚本中加入了 export命令，因此可以在任意多的脚本中使用变量 film，它们均继承了film的所有权。

不可以将变量从子进程导出到文进程， 然面通过重定向就可做到这一点

14.4 位置变量参数

本章开始提到有4种变量，本地、环境，还有两种变量被认为是特殊变量，因为它们是只读的。这两种变量即为位置变量和特定变量参数。先来看一看位置变量。

如果要向一个shell脚本传递信息，可以使用位置参数完成此功能。参数相关数目传入脚本，此数目可以任意多，但只有前 9个可以被访问，使用 shift命令可以改变这个限制。本书后面将讲到shift命令。参数从第一个开始，在第 9个结束；每个访问参数前要加 $符号。第一个参数为0，表示预留保存实际脚本名字。无论脚本是否有参数，此值均可用。

如果向脚本传送Did You See The Full Moon信息，下面的表格讲解了如何访问每一个参数。

$0	$1	$2	$3	$4	$5	$6	$7	$8	$9
脚本名字	Did	You	See	The	Full	Moon			

14.4.1 在脚本中使用位置参数

在下面脚本中使用上面的例子。

```
$ pg param
#!/bin/sh
# param
echo "This is the script name        : $0"
echo "This is the first parameter    : $1"
echo "This is the second parameter   : $2"
echo "This is the third parameter    : $3"
echo "This is the fourth parameter   : $4"
echo "This is the fifth parameter    : $5"
echo "This is the sixth parameter    : $6"
echo "This is the seventh parameter  : $7"
echo "This is the eighth parameter   : $8"
echo "This is the ninth parameter    : $9"
```

```
$ param Did You See The Full Moon
This is the script name        : ./param
This is the first parameter    : Did
This is the second parameter   : You
This is the third parameter    : See
This is the fourth parameter   : The
This is the fifth parameter    : Full
This is the sixth parameter    : Moon
This is the seventh parameter  :
This is the eighth parameter   :
This is the ninth parameter    :
```

这里只传递6个参数，7、8、9参数为空，正像预计的那样。注意，第一个参数表示脚本名，当从脚本中处置错误信息时，此参数有很大作用。

下面的例子返回脚本名称。

```
$ pg param2
#!/bin/sh
echo "Hello world this is $0 calling"
```

```
$ param2
Hello world this is ./param2 calling
```

注意$0返回当前目录路径，如果只返回脚本名，在 basename命令下参数设为$0，刚好得到脚本名字。

```
$ pg param2
#!/bin/sh
echo "Hello world this is `basename $0` calling"
```

```
$ param2
Hello world this is param2 calling
```

14.4.2 向系统命令传递参数

可以在脚本中向系统命令传递参数。下面的例子中，在 find命令里，使用 $1参数指定查找文件名。

```
$  pg findfile
#!/bin/sh
# findfile
find / -name $1 -print

$ findfile passwd
/etc/passwd
/etc/uucp/passwd
/usr/bin/passwd
```

另一个例子中，以$1向grep传递一个用户id号，grep使用此id号在passwd中查找用户全名。

```
$ pg who_is
#!/bin/sh
# who_is
grep $1 passwd | awk -F: {print $4}'

$ who_is seany
Seany Post
```

14.4.3 特定变量参数

既然已经知道了如何访问和使用shell脚本中的参数，多知道一点相关信息也是很有用的，有必要知道脚本运行时的一些相关控制信息，这就是特定变量的由来。共有 7个特定变量，见表14-2。

表14-2 特定shell变量

$#	传递到脚本的参数个数
$*	以一个单字符串显示所有向脚本传递的参数。与位置变量不同，此选项参数可超过 9个
$$	脚本运行的当前进程ID号
$!	后台运行的最后一个进程的进程ID号
$@	与$#相同，但是使用时加引号，并在引号中返回每个参数
$-	显示shell使用的当前选项，与set命令功能相同
$?	显示最后命令的退出状态。0表示没有错误，其他任何值表明有错误。

现在来修改脚本param并替换各种特定变量，与以前的例子不同，用不同的传递文本重新运行脚本。

```
$ pg param
#!/bin/sh
# allparams
echo "This is the script name          : $0"
echo "This is the first parameter      : $1"
echo "This is the second parameter     : $2"
echo "This is the third parameter      : $3"
echo "This is the fourth parameter     : $4"
echo "This is the fifth parameter      : $5"
echo "This is the sixth parameter      : $6"
echo "This is the seventh parameter    : $7"
echo "This is the eighth parameter     : $8"
echo "This is the ninth parameter      : $9"
echo "The number of arguments passed   :$#"
echo "Show all arguments               : $*"
```

```
echo "Show me my process ID            : $$"
echo "Show me the arguments in quotes  :" "$@"
echo "Did my script go with any errors :$?"


$ param Merry Christmas Mr Lawrence
This is the script name           :./param
This is the first parameter       : Merry
This is the second parameter      : Christmas
This is the third parameter       : Mr Lawrence
This is the fourth parameter      :
This is the fifth parameter       :
This is the sixth parameter       :
This is the seventh parameter     :
This is the eighth parameter      :
This is the ninth parameter       :
The number of arguments passed    :3
Show all arguments                : Merry Christmas Mr Lawrence
Show me my process ID             : 630
Show me the arguments in quotes   : "Merry" "Christmas" "Mr Lawrence"
Did my script go with any errors :0
```

特定变量的输出使用户获知更多的脚本相关信息。可以检查传递了多少参数，进程相应的ID号，以免我们想杀掉此进程。

14.4.4 最后的退出状态

注意，$?返回0。可以在任何命令或脚本中返回此变量以获得返回信息。基于此信息，可以在脚本中做更进一步的研究，返回 0意味着成功，1为出现错误。

下面的例子拷贝文件到/tmp，并使用$?检查结果。

```
$ cp ok.txt /tmp
$ echo $?
0
```

现在尝试将一个文件拷入一个不存在的目录：

```
$ cp ok.txt /usr/local/apps/dsf
cp: cannot create regular file '/usr/local/apps/dsf': No such file or
directory


$ echo $?
1
```

使用$?检验返回状态，可知脚本有错误，但同时发现 cp：cannot...，因此检验最后退出状态已没有必要。在脚本中可以用系统命令处理输出格式，要求命令输出不显示在屏幕上。为此可以将输出重定向到 /dev/null，即系统bin中。现在怎样才能知道脚本正确与否？好，这时可以用最后退出状态命令了。请看上一个例子的此形式的实际操作结果。

```
$ cp ok.txt /usr/local/apps/dsf >/dev/null 2>&1
$ echo $?
1
```

通过将包含错误信息的输出重定向到系统 bin中，不能获知最后命令返回状态，但是通过使用$!，（其返回值为1）可知脚本失败。

检验脚本退出状态时，最好将返回值设置为一个有意义的名字，这样可以增加脚本的可读性。

```
$ cp ok.txt /usr/local/apps/dsf >/dev/null 2>&1
$ cp_status=$?
$ echo $cp_status
1
```

14.5 小结

变量可以使shell编程更容易。它能够保存输入值并提高效率。 shell变量几乎可以包含任何值。特定变量增强了脚本的功能并提供了传递到脚本的参数的更多信息。