

## 第26章 shell 工 具

本章将讨论以下内容：

- 创建以日期命名的文件及临时文件。
- 信号。
- trap命令以及如何捕获信号。
- eval命令。
- logger命令。

### 26.1 创建保存信息的文件

任何脚本都应该能够创建临时文件或日志文件。在运行脚本做备份时，最好是保存一个日志文件。这些日志文件通常在文件系统中保留几周，过时将被删除。

在开发脚本的时候，可能总要创建一些临时的文件。在正常运行脚本的时候，也要使用临时文件保存信息，以便作为另外一个进程的输入。可以使用 cat命令来显示一个临时文件的内容或把它打印出来。

#### 26.1.1 使用date命令创建日志文件

在创建日志文件时，最好能够使它具有唯一性，可以按照日志文件创建的日期和时间来识别这些文件。我们可以使用 date命令做到这一点。这样就能够使日期和时间成为日志文件名中的一部分。

为了改变日期和时间的显示格式，可以使用如下的命令：

```
date option + %format
```

使用加号 ‘+’ 可以设置当前日期和时间的显示格式。下面的例子将日期以日、月、年的格式显示：

```
$ date +%d%m%y  
090699
```

下面是一些常用的日期格式：

```
$ date +%d-%m-%y  
09-06-99
```

```
$ date +%A%e" "%B" "%Y  
Wednesday 9 June 1999
```

下面的命令可以使时间按照 hh:mm 的格式显示：

```
$ date +%R  
10:07
```

```
$ date +%A" "%R" "%p  
Wednesday 10:09 AM
```

下面的命令可以显示完整的时间：

```
$ date +%T
```

10:29:41

```
$ date +%A "%T"  
Wednesday 10:31:19
```

注意，如果希望在日期和时间的显示中包含空格，要使用双引号。

在文件名中含有日期的一个简单办法就是使用置换。把含有你所需要的日期格式的变量附加在相应的日志文件名后面即可。

在下面的例子中我们创建了两个日志文件，一个使用了 dd, mm, yy 的日期格式，另一个使用了 dd, hh, mm 的时间格式。

下面就是这个脚本。

```
$ pg log  
#!/bin/sh  
# log  
#  
MYDATE=`date +%d%m%y`  
# append MYDATE to the variable LOGFILE that holds the actual filename of  
the log.  
LOGFILE=/logs/backup_log.$MYDATE  
# create the file  
>$LOGFILE  
  
MYTIME=`date +%d%R`  
LOGFILE2=/logs/admin_log.$MYTIME  
# create the file  
>$LOGFILE2
```

运行上面的脚本后，得到这样两个日志文件。

```
backup_log.090699  
admin_log.0910:18
```

### 26.1.2 创建唯一的临时文件

在本书的前面讨论特殊变量时，曾介绍变量 \$\$，该变量中保存有你所运行的当前进程的进程号。可以使用它在我们运行的脚本中创建一个唯一的临时文件，因为该脚本在运行时的进程号是唯一的。我们只要创建一个文件并在后面附加上 \$\$ 即可。在脚本结束时，只需删除带有 \$\$ 扩展的临时文件即可。Shell 将会把 \$\$ 解析为当前的进程号，并删除相应的文件，而不会影响以其他进程号做后缀的文件。

在命令行中输入如下的命令：

```
$ echo $$  
281
```

这就是当前的进程号，如果你执行这个命令，看到的结果可能会有所不同。现在如果我创建另一个登录进程并输入同样的命令，将会得到一个不同的进程号，因为我已经启动了一个新的进程。

```
$ echo $$  
382
```

下面的例子中，创建了两个临时文件，并进行了相应的操作，最后在结束时删除了这些文件。

```
$ pg tempfiles
#!/bin/sh
# tempfiles
# name the temp files
HOLD1=/tmp/hold1.$$
HOLD2=/tmp/hold2.$$

# do some processing using the files
df -tk >$HOLD1
cat $HOLD1 >$HOLD2
# now delete them
rm /tmp/*.$$
```

当上面的脚本运行时，将会创建这样两个文件：

```
hold1.408
hold2.408
```

在执行 `rm /tmp/*.$$` 时，shell 实际上将该命令解析为 `rm /tmp/*.408`。

记住，该进程号只在当前进程中唯一。例如，如果我再次运行上面的脚本，将会得到一个新的进程号，因为我已经创建了一个新的进程。

如果文件有特殊用途的话，那么创建含有日期的文件，就可以使你很容易地查找到它们。而且还可以很容易地按照日期删除文件，因为这样一眼就能看出哪个文件是最新的，哪个文件是最“旧”的。

还可以使用这种方法来快速地创建临时文件，它们在当前进程中是唯一的。在脚本结束之前，也很容易删除这些临时文件。

## 26.2 信号

信号就是系统向脚本或命令发出的消息，告知它们某个事件的发生。这些事件通常是内存错误，访问权限问题或某个用户试图停止你的进程。信号实际上是一些数字。下表列出了最常用的信号及它们的含义。

信 号	信 号 名	含 义
1	SIGHUP	挂起或父进程被杀死
2	SIGINT	来自键盘的中断信号，通常是 <CTRL-C>
3	SIGQUIT	从键盘退出
9	SIGKILL	无条件终止
11	SIGSEGV	段（内存）冲突
15	SIGTERM	软件终止（缺省杀进程信号）

还有信号 0，我们前面在创建 `.logout` 文件时已经遇到过。该信号为“退出 shell”信号。为了发出信号 0，只要从命令行键入 `exit`，或在一个进程或命令行中使用 <CTRL-D> 即可。

发送信号可以使用如下的格式：

```
kill [-signal no:] signal name process ID
```

使用 `kill` 命令时不带任何信号或名字意味着使用缺省的信号 15。

可以使用如下的命令列出所有的信号：

```
$ kill -l
1) SIGHUP      2) SIGINT      3) SIGQUIT     4) SIGILL
```

5) SIGTRAP	6) SIGIOT	7) SIGBUS	8) SIGFPE
9) SIGKILL	10) SIGUSR1	11) SIGSEGV	12) SIGUSR2
13) SIGPIPE	14) SIGALRM	15) SIGTERM	17) SIGCHLD
18) SIGCONT	19) SIGSTOP	20) SIGTSTP	21) SIGTTIN
22) SIGTTOU	23) SIGURG	24) SIGXCPU	25) SIGXFSZ
26) SIGVTALRM	27) SIGPROF	28) SIGWINCH	29) SIGIO
30) SIGPWR			

### 26.2.1 杀死一个进程

发送信号 1 将使一个进程重新读入配置文件。例如，你在运行域名服务（DNS）守护进程 named，现在你对域名数据库文件做了某些修改，这时不需要杀死该守护进程再重新启动，只需使用 kill -1 命令向其发送信号 1。Named 进程将重新读入它的配置文件。

下面的例子向系统中一个名为 mon\_web 的进程发送信号 9（无条件终止）来杀死它。首先使用 ps 命令得到相应的进程号。

```
$ ps -ef | grep mon_web | grep -v root
157 ? S 0:00 mon_web
```

如果系统不支持 ps -ef 命令，那么可以使用 ps xa。为了杀死该进程，我可以使用下面的两种方法之一：

```
kill -9 157
```

或

```
kill -s SIGKILL 157
```

在有些系统中，不必使用 -s，例如：kill SIGKILL 157。

下面的脚本将根据进程名来杀死一个进程，拟被杀死的进程名作为该脚本的一个参数。在执行相应的命令之前，将会首先检查是否存在这样的进程。在这里使用 grep 命令来匹配相应的进程名。如果匹配成功，则向用户提示进程已经找到，并询问用户是否杀死该进程。最后使用 kill -9 命令杀死相应的进程。

下面就是该脚本。

```
$ pg pskill
#!/bin/sh
# pskill
HOLD1=/tmp/hold1.$$
PROCESS=$1
usage()
{

# usage
echo "Usage : `basename $0` process_name"
exit 1
}
if [ $# -ne 1 ]; then
    usage
fi

case $1 in
*)
# grep the process, do not include our script in the output from ps
# extract fields 1 and 6, redirect to a temp file
```

```

ps x | grep $PROCESS | grep -v $0 | awk '{print $1"\t" $6}'>$HOLD1
# ps -ef |..      if ps x does not work
;;
esac

# is the file there??
if [ ! -s $HOLD1 ]; then
    echo "No processes found..sorry"
    exit 1
fi

# read in the contents from the temp file and display the fields
while read LOOP1 LOOP2
do
    echo $LOOP1 $LOOP2
done <$HOLD1
echo -n "Are these the processes to be killed ? [y..n] >"
read ANS

case $ANS in
Y|y) while read LOOP1 LOOP2
do
    echo $LOOP1
    kill -9 $LOOP1
done <$HOLD1
rm /tmp/*.$$
;;
N|n);;
esac

```

运行该脚本将会产生如下的输出：

```

$ pskill web
1760 ./webmon
1761 /usr/apps/web_col
Are these the processes to be killed ? [y..n] >y

```

```

1760
1761
[1]+  Killed                  webmon

```

在使用该脚本时，要确信存在相应的进程：

```

$ pskill web
No processes found..sorry

```

### 26.2.2 检测信号

有些信号可以被应用程序或脚本捕获，并依据该信号采取相应的行动。另外一些信号不能被捕获。例如，如果一个命令收到了信号9，就无法再捕捉其他信号。

在编写shell脚本时，只需关心信号1、2、3和15。当脚本捕捉到一个信号后，它可能会采取下面三种操作之一：

- 1) 不采取任何行动，由系统来进行处理。
- 2) 捕获该信号，但忽略它。

3) 捕获该信号，并采取相应的行动。

大多数的脚本都使用第一种处理方法，这也是到目前为止本书中所有脚本所采取的处理方法。

如果想要采取另外两种处理方法，必须使用 trap 命令。

## 26.3 trap

trap 可以使你在脚本中捕捉信号。该命令的一般形式为：

```
trap name signal(s)
```

其中，name 是捕捉到信号以后所采取的一系列操作。实际生活中，name 一般是一个专门用来处理所捕捉信号的函数。Name 需要用双引号（“ ”）引起来。Signal 就是待捕捉的信号。

脚本在捕捉到一个信号以后，通常会采取某些行动。最常见的行动包括：

- 1) 清除临时文件。
- 2) 忽略该信号。
- 3) 询问用户是否终止该脚本的运行。

下表列出了一些最常见的 trap 命令用法：

trap "" 2 3	忽略信号 2 和信号 3，用户不能终止该脚本
trap "commands" 2 3	如果捕捉到信号 2 或 3，就执行相应的 commands 命令
trap 2 3	复位信号 2 和 3，用户可以终止该脚本

也可以使用单引号（‘ ’）来代替双引号（“ ”）；其结果是一样的。

### 26.3.1 捕获信号并采取相应的行动

下面的例子一经运行就开始计数直至用户按 <Ctrl-C>（信号 2）。这时该脚本将会显示出当前的循环数字，然后退出。

在本例中 trap 命令的格式为：

```
trap "do_something" signal no:(s)
```

下面就是该脚本：

```
$ pg trap1
#!/bin/sh
#trap1
trap "my_exit" 2
LOOP=0
my_exit()
{
echo "You just hit <CTRL-C>, at number $LOOP"
echo " I will now exit "
exit 1
}

while :
do
  LOOP=`expr $LOOP + 1`
  echo $LOOP
done
```

现在让我们来仔细分析一下该脚本。

```
trap "my_exit" 2
```

在本例中，由于设置了 trap 命令，所以在捕捉到信号 2 以后，双引号内的 my\_exit 函数将被执行。

```
my_exit()
{
    echo "You just hit <CTRL-C>, at number $LOOP"
    echo " I will now exit "
    exit 1
}
```

函数 my\_exit 将在脚本捕捉到信号 2 后被调用；用户将会看到 \$LOOP 变量的内容，即用户按 <Ctrl-C> 时的计数值。在实际中，通常捕捉到信号 2 后所调用的函数是用来完成清除临时文件等任务的。

下面是该脚本的运行结果：

```
$ trap1
1
...
...
211
212
You just hit <CTRL-C>, at number 213
I will now exit
```

### 26.3.2 捕获信号并采取行动的另一个例子

下面就是一个捕获信号后清除临时文件的例子。

下面的脚本在运行时不断使用 df 和 ps 命令向临时文件 HOLD1.\$\$ 和 HOLD2.\$\$ 中写入相应的信息。你应该还记得 \$\$ 表示当前的进程号。当用户按 <CTRL-C> 时，这些临时文件将被清除。

```
$ pg trap2
#!/bin/sh
# trap2
# trap only signal 2....<CTRL-C>
trap "my_exit" 2
HOLD1=/tmp/HOLD1.$$
HOLD2=/tmp/HOLD2.$$

my_exit()
{
    # my_exit
    echo "<CTRL-C> detected..Now cleaning up..wait"
    # delete the temp files
    rm /tmp/*.$$ 2> /dev/null
    exit 1
}

echo "processing..."
# loop forever, do some processing
while :
do
    df >>$HOLD1
```

```
ps xa >>$HOLD2
done
```

上面的脚本在运行时会产生如下的结果：

```
$ trap2
processing....
<CTRL-C> detected..Now cleaning up..wait
```

当收到信号2或3时，尽管一般情况下这都不是误操作，但是为了安全起见，不妨给用户一个选择的机会，这样用户在不小心中按下<CTRL-C>后，仍然可以撤消刚才的动作。

在下面的例子中，在脚本捕捉到信号2后将会向用户提供一个选择，询问用户是否真的要退出。这里使用case语句来决定采取何种操作。

如果用户希望退出，他或她可以选择1，此时当前函数会以状态1退出，而另一个清除进程将会据此启动。如果用户并不希望退出，那么可以选择2或不做任何选择，此时case语句将使用户退回到脚本中原来的地方。在case语句中一定要包含用户输入空字符串的情况。

下面的函数在收到信号后，将会向用户提供选择：

```
my_exit()
{
# my_exit
echo -e "\nReceived interrupt ... "
echo "Do you really wish to exit ???"
echo " 1: Yes"
echo " 2: No"
echo -n " Your choice [1..2] >"
read ANS
case $ANS in
1) # cleanup temp files.. etc..
    exit 1
    ;;
2) # do nothing
3) ;;
esac
}
```

下面是完整的脚本：

```
$ pg trap4
#!/bin/sh
# trap4
# trap signal 1 2 3 and 15
trap "my_exit" 1 2 3 15

LOOP=0

# temp files
HOLD1=/tmp/HOLD1.$$
HOLD2=/tmp/HOLD2.$$
my_exit()
{
# my_exit
echo -e "\nRecieved interrupt..."
echo "Do you wish to really exit ???"
echo " Y: Yes"
echo " N: No"
```



```
echo -n " Your choice [Y..N] >"
read ANS
case $ANS in
Y|y) exit 1;;      # exit the script
N|n) ;;           # return to normal processing
esac
}
```

```
# a while loop here perhaps for reading in fields
echo -n "Enter your name : "
read NAME
echo -n "Enter your age : "
read AGE
```

当上面的脚本运行时，只要在输入任何域时按下 <CTRL-C>，就会得到一个选择：是继续运行还是退出。

```
$ trap4
Enter your name :David Ta
Received interrupt...
Do you really wish to exit ???
1: Yes
2: No
Your choice [1..2] >2
```

```
Enter your age :
```

### 26.3.3 锁住终端

下面的脚本是另一个捕获信号的例子。该脚本名为 lockit，它将使用一个连续不断的 while 循环锁住终端。在该脚本中，trap 命令捕捉信号 2、3 和 15。如果一个用户试图中断该脚本的运行，将会得到一个不成功的提示。

在脚本初次执行时，将会被提示输入一个口令。在解锁终端时没有任何提示，可以直接输入口令并按回车键。该脚本会从终端读入所输入的口令，并与预先设置的口令做比较，如果一致就解锁终端。

如果忘记了自己的口令，那么只好登录到另一个终端上并杀死该进程。在本例中没有对口令的长度加以限制——这完全取决于你。

如果你从另外一个终端上杀死了该进程，当再次回到这个终端时，可能会遇到终端设置问题，例如回车键不起作用。这时可以试着使用下面的命令，这样可以解决大部分问题。

```
$ stty sane
```

下面就是该脚本。

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

# lockit
# trap signals 2 3 and 15
trap "nice_try" 2 3 15
```

```
# get the device we are running on
TTY=`tty`
```

```
nice_try()
{
# nice_try
echo "Nice try, the terminal stays locked"
}

# save stty settings hide characters typed in for the password
SAVEDSTTY=`stty -g`
stty -echo
echo -n "Enter your password to lock $TTY : "
read PASSWORD
clear

while :
do
# read from tty only !!,
read RESPONSE < $TTY
if [ "$RESPONSE" = "$PASSWORD" ]; then
# password matches...unlocking
echo "unlocking..."
break
fi

# show this if the user inputs a wrong password
# or hits return

echo "wrong password and terminal is locked.."
done

# restore stty settings
stty $SAVEDSTTY
```

下面是lockit脚本运行时的输出：

```
$ lockit
Enter your password to lock /dev/tty$1 :
```

接着屏幕就被清除。如果按回车键或其他错误的口令，该脚本将会输出：

```
wrong password and terminal is locked..
Nice try, the terminal stays locked
wrong password and terminal is locked..
Nice try, the terminal stays locked
wrong password and terminal is locked..
```

现在输入正确的口令：

```
unlocking...
$
```

现在又回到命令提示符下了。

#### 26.3.4 忽略信号

在用户登录时，系统将会执行/etc/profile文件，根用户不希望其他普通用户打断这一进程。他通常通过设置trap来屏蔽信号1、2、3和15，然后在用户读当天的消息时重新打开这些信号。最后仍然回到屏蔽这些信号的状态。

在编写脚本时也可以采用类似的办法。在脚本运行的某些关键时刻，比如打开了很多文

件时，不希望该脚本被中断，以免破坏这些文件。通过设置 `trap` 来屏蔽某些信号就可以解决这个问题。在这些关键性的处理过程结束后，再重新打开信号。

忽略信号的一般格式为（信号 9 除外）：

```
trap "signal no:(s)
```

注意，在双引号之间没有任何字符，为了重新回到捕捉信号的状态，可以使用如下的命令：

```
trap "do something" signal no:(s)
```

下面我们来总结一下上述方法。

```
trap "" 1 2 3 15：忽略信号。
```

关键性的处理过程

```
trap "my_exit" 1 2 3 15：重新回到捕捉信号的状态，在捕捉到信号后调用 my_exit 函数。
```

下面就是一个这样的例子，其中的“关键”过程实际上是一个 `while` 循环，但它能够很好地说明这种方法。在第一个循环中，通过设置 `trap` 来屏蔽信号，但是在第二个例子中，又回到捕捉信号的状态。

两个循环都只数到 6，不过在循环中使用了一个 `sleep` 命令，这样就可以有充分的时间来实验中断该循环。

下面就是脚本。

```
$ pg trap_ignore
#!/bin/sh
# trap_ignore
# ignore the signals
trap "" 1 2 3 15

LOOP=0
my_exit()
# my_exit
{
echo "Received interrupt on count $LOOP"
echo "Now exiting..."
exit 1
}

# critical processing, cannot be interrupted...
LOOP=0
while :
do
    LOOP=`expr $LOOP + 1`
    echo "critical processing..$LOOP..you cannot interrupt me"
    sleep 1
    if [ "$LOOP" -eq 6 ]; then
        break
    fi
done

LOOP=0
# critical processing finished, now set trap again but this time allow
interrupts.
trap "my_exit" 1 2 3 15
```

```
while :
do
  LOOP=`expr $LOOP + 1`

  echo "Non-critical processing..$LOOP..interrupt me now if you want"
  sleep 1
  if [ "$LOOP" -eq 6 ]; then
    break
  fi
done
```

在上面的脚本在运行时，如果我们在第一个循环期间按下 <Ctrl-C>，它不会有任何反应，这是因为我们通过设置 trap 屏蔽了信号；而在第二个循环中由于重新回到捕捉信号的状态，按下 <Ctrl-C> 就会调用 my\_exit 函数。

```
$ trap_ignore
critical processing..1..you cannot interrupt me
critical processing..2..you cannot interrupt me
critical processing..3..you cannot interrupt me
critical processing..4..you cannot interrupt me
critical processing..5..you cannot interrupt me
critical processing..6..you cannot interrupt me
Non-critical processing..1..interrupt me now if you want
Non-critical processing..2..interrupt me now if you want
Received interrupt on count 2
Now exiting...
```

当脚本捕获到信号时，通过使用 trap 命令，可以更好地控制脚本的运行。捕获信号并进行处理是一个脚本健壮性的标志。

## 26.4 eval

eval 命令将会首先扫描命令行进行所有的置换，然后再执行该命令。该命令适用于那些一次扫描无法实现其功能的变量。该命令对变量进行两次扫描。这些需要进行两次扫描的变量有时被称为复杂变量。不过我觉得这些变量本身并不复杂。

eval 命令也可以用于回显简单变量，不一定是复杂变量。

```
$ NAME=Honeysuckle
$ eval echo $NAME
Honeysuckle
$ echo $NAME
Honeysuckle
```

解释 eval 命令是怎么回事的最好办法就是看几个例子。

### 26.4.1 执行含有字符串的命令

我们首先创建一个名为 testf 的小文件，在这个小文件中含有一些文本。接着，将 cat testf 赋给变量 MYFILE，现在我们 echo 该变量，看看是否能够执行上述命令。

```
$ pg testf
May Day, May Day
Going Down
```

现在我们将 cat testf 赋给变量 MYFILE。

```
$ MYFILE=cat testf
```

如果我们echo该变量，我们将无法列出 testf 文件中的内容。

```
$ echo $MYFILE
cat testf
```

让我们来试一下 eval 命令，记住 eval 命令将会对该变量进行两次扫描。

```
$ eval $MYFILE
May Day, May Day
Going Down
```

从上面的结果可以看出，使用 eval 命令不但可以置换该变量，还能够执行相应的命令。第一次扫描进行了变量置换，第二次扫描执行了该字符串中所包含的命令 cat testf。

下面是另一个例子。一个名为 CAT\_PASSWD 的变量含有字符串 “cat /etc/passwd | more”。eval 命令可以执行该字符串所对应的命令。

```
$ CAT_PASSWD="cat /etc/passwd | more"
$ echo $CAT_PASSWD
cat /etc/passwd|more
$ eval $CAT_PASSWD
root:HccPbzT5tb00g:0:0:root:/root:/bin/sh
bin:*:1:1:bin:/bin:
daemon:*:2:2:daemon:/sbin:
adm:*:3:4:adm:/var/adm:
...
...
```

eval 命令还可以用来显示出传递给脚本的最后一个参数。现在来看下面的这个例子。

```
$ pg evalit
#!/bin/sh
# evalit
echo " Total number of arguments passed is $# "
echo " The process ID is $$ "
echo " Last argument is " $(eval echo \$$#)
```

在运行上述脚本时，我们会看到如下的结果（你所看到进程号可能会不一样）：

```
$ evalit alpha bravo charlie
Total number of arguments passed is 3
The process ID is 780
Last argument is charlie
```

在上面的脚本中，eval 命令首先把 \$\$# 解析为当前 shell 的参数个数，然后在第二次扫描时得出最后一个参数。

#### 26.4.2 给每个值一个变量名

可以给一个值一个变量名。下面我对此做些解释，假定有一个名为 data 的文件：

```
$ pg data
PC      486
MONITOR svga
NETWORK yes
```

你希望该文件中的第一列成为变量名，第二列成为该变量的值，这样就可以：

```
echo $PC
486
```

怎样才能做到这一点呢？当然是使用 eval 命令。

```
$ pg eval_it
#!/bin/sh
#eval_it
while read NAME TYPE
do
    eval `echo "${NAME}=${TYPE}"`
done < data
echo "You have a $PC pc, with a $MONITOR monitor"
echo "and are you network ? $NETWORK"
```

我们用data文件的第一行来解释上述脚本的执行过程，该脚本读入“PC”和“486”两个词，把它们分别赋给变量NAME和TYPE。Eval命令的第一次扫描把NAME和TYPE分别置换为“PC”和“486”，第二次扫描时将PC作为变量，并将“486”作为变量的值。

下面是运行上述脚本的结果：

```
$ eval_it
You have a 486 pc, with a svga monitor
and are you network ? yes
```

eval命令并不是一个在脚本中很常见的命令，但是如果需要对变量进行两次扫描的话，就要使用eval命令了。

## 26.5 logger命令

系统中含有相当多的日志文件。其中的一个日志文件叫作 messages，它通常位于 /var/adm 或 /var/log 目录下。一个名为 syslog 的配置文件可以用来定义记录在 messages 文件中的消息，这些消息有一定的格式。如果想知道系统中的相应配置，可以查看 /etc/syslog.conf 文件。该文件中包含了用于发送各种不同类型消息的工具及它们的优先级。

这里我们并不想深入探讨 UNIX 和 LINUX 是如何向该文件中记录信息的。我们现在只要知道这些消息有不同的级别，从信息性的消息到关键性的消息。

还可以使用 logger 命令向该文件发送消息。在使用该命令之前，最好查阅联机手册，因为在不同供应商所提供的操作系统上该命令的语法也有所不同。

不过，由于这里只涉及到信息性的消息，因此不必担心下面的命令不安全。

你可能会出于下列的原因向该文件中发送消息：

- 在某一个特定的时间段出现的访问或登录。
- 你的某些执行关键任务的脚本运行失败。
- 监控脚本的报告。

下面是 /var/adm/messages 文件的例子。在系统上所看到的相应文件可能和下面的例子有少许差别。

```
$ tail /var/adm.messages
Jun 16 20:59:03 localhost login[281]: DIALUP AT ttyS1 BY root
Jun 16 20:59:03 localhost login[281]: ROOT LOGIN ON ttyS1
Jun 16 20:59:04 localhost PAM_pwdb[281]: (login) session closed for user
root
Jun 16 21:58:38 localhost named[211]: Cleaned cache of 0 RRs
Jun 16 21:58:39 localhost named[211]: USAGE 929570318 929566719
Jun 16 21:58:39 localhost named[211]: NSTATS 929570318 929566719
```

logger 命令的一般形式为：

```
logger -p -I message
```

其中：

-p：为优先级，这里只涉及到提示用户注意的优先级，这也是缺省值。

-i：在每个消息中记录发送消息的进程号。

### 26.5.1 使用logger命令

可以使用如下命令：

```
$ logger -p notice "This is a test message.Please Ignore $LOGNAME"
```

可能需要等几分钟才能看到该消息被记录到 message文件中。

```
$ tail /var/adm/messages
```

```
...  
...
```

```
Jun 17 10:36:49 acers6 dave: This is a test message.Please Ignore dave
```

如你所见，发送这一消息的用户也被记录了下来。

现在来创建一个小小的脚本，用它来记录当前系统中的用户数。该脚本可以在一天的时段中记录系统的使用率。只要把它放进 crontab文件中，使它每30分钟运行一次即可。

```
$ pg test_logger  
#!/bin/sh  
# test_logger  
logger -p notice "`basename $0`:there are currently `who |wc -l` users on  
the system"
```

运行下面的脚本。

```
$ test_logger
```

现在来看看 message文件的末尾：

```
$ tail /var/adm/messages
```

```
...  
...
```

```
Jun 17 11:02:53 acers6 dave: test_script:there are currently 15 users on  
the system
```

### 26.5.2 在脚本中使用logger命令

向日志文件中发送信息的一个更为合理的用途就是用于脚本非正常退出时。如果希望向日志文件中发送消息，只要在捕获信号的退出函数中包含 logger命令即可。

在下面的清除脚本中，如果该脚本捕获到信号 2、3或15的话，就向该日志文件发送一个消息。

```
$ pg cleanup  
#!/bin/sh  
# cleanup  
# cleanup system logs  
trap "my_exit" 2 3 15  
  
my_exit()  
{  
# my_exit  
logger -p notice "`basename $0`: Was killed whilst cleaning up system  
logs..CHECK OUT ANY DAMAGE"
```

```
exit 1
}

tail -3200c /var/adm/utmp > /tmp/utmp
mv /tmp/utmp /var/adm/utmp
>/var/adm/wtmp
#
tail -10 /var/adm/sulog > /tmp/o_sulog
mv /tmp/o_sulog /var/adm/sulog
...
```

这样只要看一下这个日志文件就可以知道脚本的运行结果是否正常。

```
$ tail /var/adm/messages
...
Jun 17 11:34:28 acers6 dave: cleanup:Was killed whilst cleaning up
systemlogs..      CHECK OUT ANY DAMAGE
```

除了使用logger命令对一些关键性的脚本处理过程做日志外，我还用它来记录使用调制解调器连接系统的用户。下面的一段脚本记录了从串口 tty0和tty02连接到系统中的用户。这部分代码来自于我编写的一个/etc/profile文件。

```
TTY_LINE=`tty`
case $TTY_LINE in
"/dev/tty0")
    TERM=ibm3151
    ;;
"/dev/tty2")
    TERM=vt220
    # checks for allowed users on the modem line
    #
    echo "This is a modem connection"
    # modemf contains login names of valid users
    modemf=/usr/local/etc/modem.users
    if [-s $modemf ]
    then
        user=`cat $modemf | awk '{print $1}' | grep $LOGNAME`
        # if your name is not in the file, you are not coming in
        if [ "$user" != "$LOGNAME" ]
        then
            echo "INVALID USER FOR MODEM CONNECTION"
            echo " DISCONNECTING,,,,,"
            sleep 1
            exit 1
        else
            echo "modem connection allowed"
        fi
    fi
    logger -p notice "modem line connect $TTY_LINE..$LOGNAME"
    ;;
*) TERM=vt220
    stty erase '^h'
    ;;
esac
```



当希望在系统全局的日志文件中记录信息的时候，`logger`命令是一个非常好的工具。

## 26.6 小结

理解信号和对信号的捕获可以使脚本的退出更为完整。通过在系统日志文件中记录信息，你或系统管理员就能够更容易地发现存在的问题。