五 流程控制

2 for语句

1） 使用in关键字循环

根据变量的不同取值，重复执行一组命令操作

格式：

for 变量名 in 取值列表

do

命令序列

done

例子11：循环

#!/bin/bash

for time in morning noon afternoon evening

do

echo $time

done

例子12：

#输入目录名，显示目录下所有内容.

#!/bin/bash

read -p "please input a filename!" -t 30 filename

if [ -z $filename ];then

echo "please input!!!!!!"

exit 1

fi

#如果字符串为空，报错跳出

if [ ! -e $filename ]

then

echo "$filename not cunzai!!"

exit 2

fi

#如果文件不存在，报错跳出

if [ ! -d $filename ]

then

echo "$filename is not driectory"

exit 3

fi

#如果不是目录，报错跳出

file=`ls $filename`

for test in $file

do

echo $test

done

2）数值加加循环

例子13：

#/bin/bash

s=0

for ((i=1;i<=100;i=i+1))

do

s=$(($s+$i))

done

echo $s

3 while循环语句

重复测试指定的条件，只要条件成立则反复执行对应的命令操作

格式：

while 命令或表达式

do

命令列表

done

例子14：

批量添加用户

#!/bin/bash

i=1

while [ $i -le 20 ]

do

useradd stu$i

echo "123456" | passwd --stdin stu$i &> /dev/null

i=`expr $i + 1`

done

例子15：

批量删除用户

#!/bin/bash

aa=`cat /etc/passwd | grep "/bin/bash"|grep -v "root"|cut -d ":" -f 1`

for i in $aa

do

userdel -r $i

done

例子16：

批量添加

#!/bin/bash

aa=10

for ((i=1;i<=$aa;i=i+1))

do

useradd stu$i

echo "123456" | passwd --stdin stu$i &> /dev/null

echo $i

done

4 case多重分支语句

根据变量的不同取值，分别执行不同的命令操作

例子17：

打印选择列表，输出选择

#!/bin/bash

echo -e "shanghai: 1\n"

echo -e "beijing: 2\n"

echo -e "chengdu: 3\n"

read -p "input your choice:" -t 30 choi

case $choi in

"1")

echo "shanghai!!!"

;;

"2")

echo "beijing!!!"

;;

"3")

echo "chengdu!!!"

;;

\*)

echo "qing chongxin shuru!"

;;

esac

六 apache启动脚本分析：

#!/bin/bash

#

# httpd Startup script for the Apache HTTP Server

#

# chkconfig: - 85 15

#自启动设定 -代表自启动级别，85（S85）代表启动序号，15（K15）代表关闭序号。

# description: The Apache HTTP Server is an efficient and extensible \

# server implementing the current HTTP standards.

#服务描述。以上两行用于apache自启动。

# processname: httpd

# config: /etc/httpd/conf/httpd.conf

# config: /etc/sysconfig/httpd

# pidfile: /var/run/httpd/httpd.pid

#

### BEGIN INIT INFO

# Provides: httpd

# Required-Start: $local\_fs $remote\_fs $network $named

# Required-Stop: $local\_fs $remote\_fs $network

# Should-Start: distcache

# Short-Description: start and stop Apache HTTP Server

# Description: The Apache HTTP Server is an extensible server

# implementing the current HTTP standards.

### END INIT INFO

#以上都是注释。

# Source function library.

. /etc/rc.d/init.d/functions

#"."其实就是source，就是调用functions文件。

if [ -f /etc/sysconfig/httpd ]; then

. /etc/sysconfig/httpd

fi

#判断httpd如果是文件，则调用httpd文件。

# Start httpd in the C locale by default.

HTTPD\_LANG=${HTTPD\_LANG-"C"}

#定义变量HTTPD\_LANG的值。并追加变量的值为C，即英文。

# This will prevent initlog from swallowing up a pass-phrase prompt if

# mod\_ssl needs a pass-phrase from the user.

INITLOG\_ARGS=""

# Set HTTPD=/usr/sbin/httpd.worker in /etc/sysconfig/httpd to use a server

# with the thread-based "worker" MPM; BE WARNED that some modules may not

# work correctly with a thread-based MPM; notably PHP will refuse to start.

# Path to the apachectl script, server binary, and short-form for messages.

apachectl=/usr/sbin/apachectl

httpd=${HTTPD-/usr/sbin/httpd}

prog=httpd

pidfile=${PIDFILE-/var/run/httpd/httpd.pid}

lockfile=${LOCKFILE-/var/lock/subsys/httpd}

#定义一系列变量，用于后面的执行。

RETVAL=0

#定义全局命令返回变量。

STOP\_TIMEOUT=${STOP\_TIMEOUT-10}

# The semantics of these two functions differ from the way apachectl does

# things -- attempting to start while running is a failure, and shutdown

# when not running is also a failure. So we just do it the way init scripts

# are expected to behave here.

start() {

echo -n $"Starting $prog: "

LANG=$HTTPD\_LANG daemon --pidfile=${pidfile} $httpd $OPTIONS

RETVAL=$?

echo

[ $RETVAL = 0 ] && touch ${lockfile}

return $RETVAL

}

#定义start函数，用于apache的启动。  
#如果守护进程/usr/sbin/httpd 启动成功（$RETVAL = 0），就建立/var/lock/subsys/httpd文件（touch #${lockfile}）。通过$httpd变量执行/usr/sbin/httpd命令启动apache。通过$pidfile变量调用apache

#的PID。通过变量$OPTIONS定义命令执行时的初始化环境配置，依赖/etc/sysconfig/httpd文件。

# When stopping httpd, a delay (of default 10 second) is required

# before SIGKILLing the httpd parent; this gives enough time for the

# httpd parent to SIGKILL any errant children.

stop() {

echo -n $"Stopping $prog: "

killproc -p ${pidfile} -d ${STOP\_TIMEOUT} $httpd

RETVAL=$?

echo

[ $RETVAL = 0 ] && rm -f ${lockfile} ${pidfile}

}

#定义stop函数，用来关闭apache服务，关闭服务之后会删除pid文件。

reload() {

echo -n $"Reloading $prog: "

if ! LANG=$HTTPD\_LANG $httpd $OPTIONS -t >&/dev/null; then

RETVAL=6

echo $"not reloading due to configuration syntax error"

failure $"not reloading $httpd due to configuration syntax error"

else

# Force LSB behaviour from killproc

LSB=1 killproc -p ${pidfile} $httpd -HUP

RETVAL=$?

if [ $RETVAL -eq 7 ]; then

failure $"httpd shutdown"

fi

fi

echo

}

#定义reload函数，用于apache的重新加载。

#通过/usr/sbin/httpd –t命令判断apache的配置文件。如果配置文件报错，则输出错误提示。如果配

#置文件正确，则重新加载apache。

# See how we were called.

case "$1" in

#判断执行脚本后的第一个参数的值，$1表示执行脚本时的第一个参数。

start)

start

;;

;;

#如果参数值为start，则调用start函数。

stop)

stop

;;

#如果参数值为stop，则调用stop函数。

status)

status -p ${pidfile} $httpd

RETVAL=$?

;;

#如果参数值为status，则执行status –p $httpd命令测试apache状态。

restart)

stop

start

;;

#如果参数值为restart，则先调用stop函数，再调用start函数

condrestart|try-restart)

if status -p ${pidfile} $httpd >&/dev/null; then

stop

start

fi

;;

#如果参数值为condrestart或try-restart，则只有apache服务是已经运行时才先调用stop函数，再调

#用start函数，重启apache。如果apache服务没有运行，则不重启apache。

force-reload|reload)

reload

;;

#如果参数值为force-reload或reload，则调用reload函数。

graceful|help|configtest|fullstatus)

$apachectl $@

RETVAL=$?

;;

#如果参数是graceful或help或configtest或fullstatus，则执行/usr/sbin/apachectl命令，并把参

#数作为命令的参数传入apachectl命令。

\*)

echo $"Usage: $prog {start|stop|restart|condrestart|try-restart|force-reload|reload|status|fullstatus|graceful|help|configtest}"

RETVAL=2

#如果输出的参数不是以上任何参数，则输出错误信息

esac

exit $RETVAL