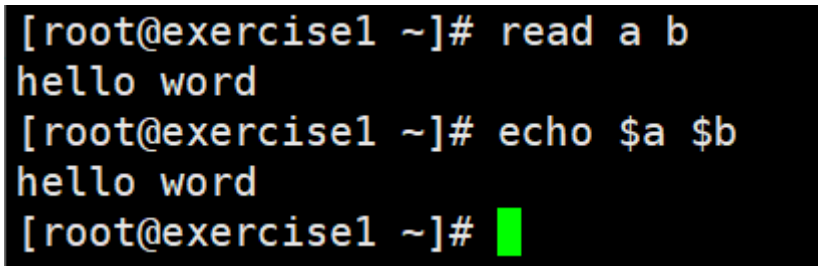
read命令从键盘读取变量的值

从键盘读取变量的值，通常用在shell脚本中与用户进行交互的场合。

该命令可以一次读取多个变量的值，变量和输入的值都需要使用空格隔开。在read命令后面，如果没有指定变量名，读取的数据将被自动赋值给特定的变量REPLY

read从键盘读入数据，赋给变量

```
例子1：

[root@exercise1 ~]# read a b
hello word
[root@exercise1 ~]# echo $a $b
hello word
[root@exercise1 ~]#
```




```
read常用见用法及参数

例子2：从标准输入读取一行并赋值给变量passwd

[root@exercise1 ~]# read passwd
123456
[root@exercise1 ~]#

例子3：读取多个值，从标准输入读取一行，直至遇到第一个空白符或换行符。
        把用户键入的第一个词存到变量first中，把该行的剩余部分保存到变量last中

[root@exercise1 ~]# read firt  last
```

```
aaaa bbbbb
[root@exercise1 ~]#
```

例子3：read -s passwd将你输入的东西隐藏起来，值赋给passwd。这个用户隐藏密码信息

[root@exercise1 ~]# read -s passwd
[root@exercise1 ~]# echo $passwd
123456789
[root@exercise1 ~]#

例子4：输入的时间限制
[root@exercise1 ~]# read -t 2　#超过两秒没有输入，直接退出

例子5：输入的长度限制

[root@exercise1 ~]# read -n 2 e　#最多只接受2个字符
11
[root@exercise1 ~]#

```
例子6：使用-r 参数输入，允许让输入中的内容包括：\识别为普通字符

[root@exercise1 ~]# read -r f
sdf sdf / sdfs \n
[root@exercise1 ~]# echo $f
sdf sdf / sdfs \n
[root@exercise1 ~]# read g
sdf sdf / sdfs \n
[root@exercise1 ~]# echo $g
sdf sdf / sdfs n
[root@exercise1 ~]#
```

例子7：-p用于给出提示符，在下面的例子中我们使用了echo –n“...“来给出提示符

方法一：
[root@exercise1 ~]# read -p "please input:" pass
please input:123456
[root@exercise1 ~]# echo $pass

123456
[root@exercise1 ~]#

方法二：(了解)
[root@exercise1 ~]# echo -n "please input:" ; read pass
please input:123456
[root@exercise1 ~]# echo $pass
123456

综合实例
[root@exercise1 ~]# vim /opt/test-read.sh #写入以下内容

#!/bin/bash
read -p "请输入姓名：" NAME
read -p "请输入年龄：" AGE
read -p "请输入性别：" SEX
cat<<ww
****
你的基本信息如下：
姓名：$NAME年龄：AGE
性别：$SEX
****
ww

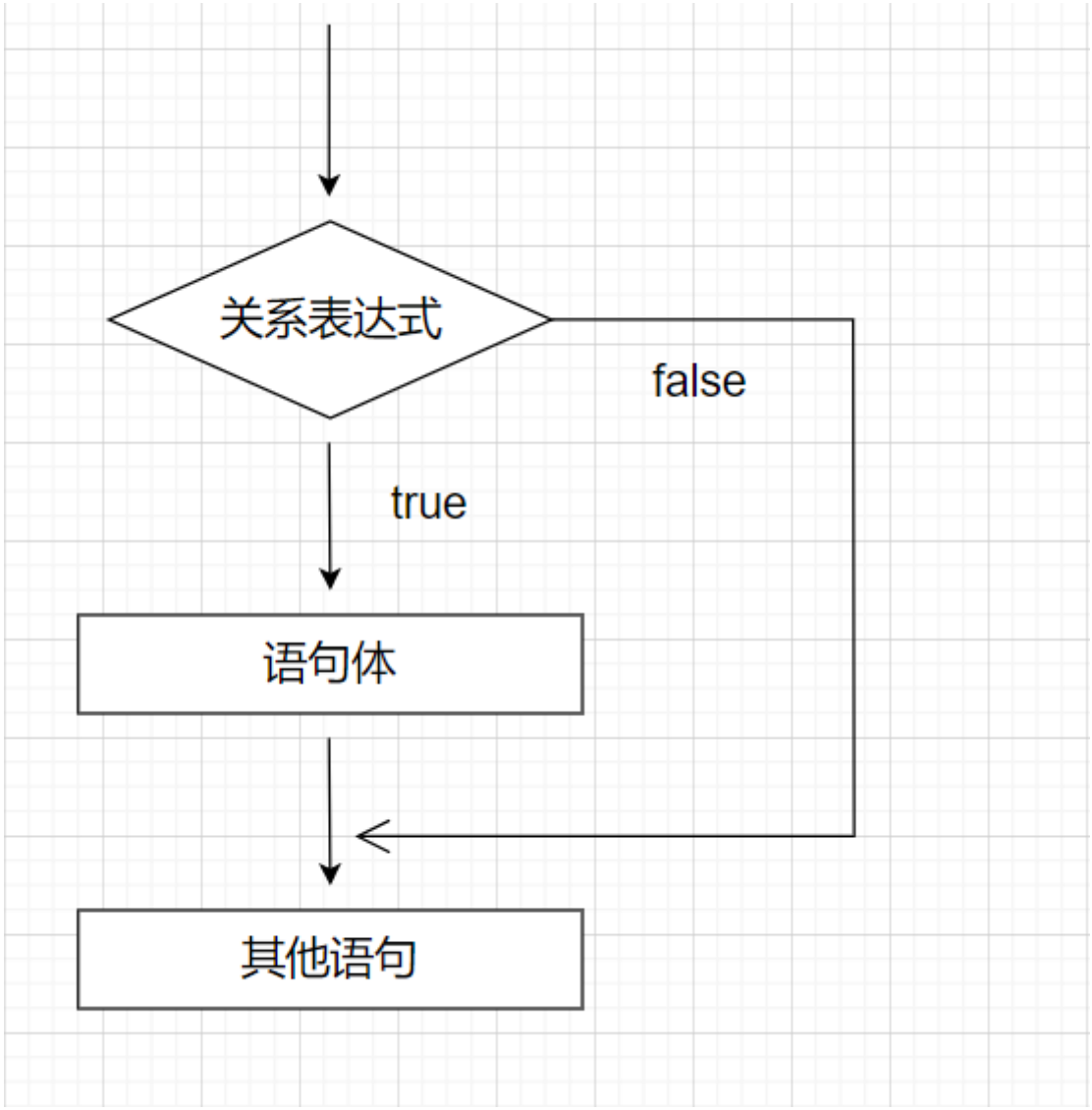
[root@exercise1 opt]# sh test-read.sh
请输入姓名：wwww
请输入年龄：www
请输入性别：ww
****
你的基本信息如下：
姓名：wwww
年龄：www
性别：ww
****

流程控制语句if

语法格式：

if 条件
then
commands
fi

if 语句流程图：



注：根据我们的命令退出码来进行判断(echo $?=0)，如果是0,那么就会执行then后面的命令

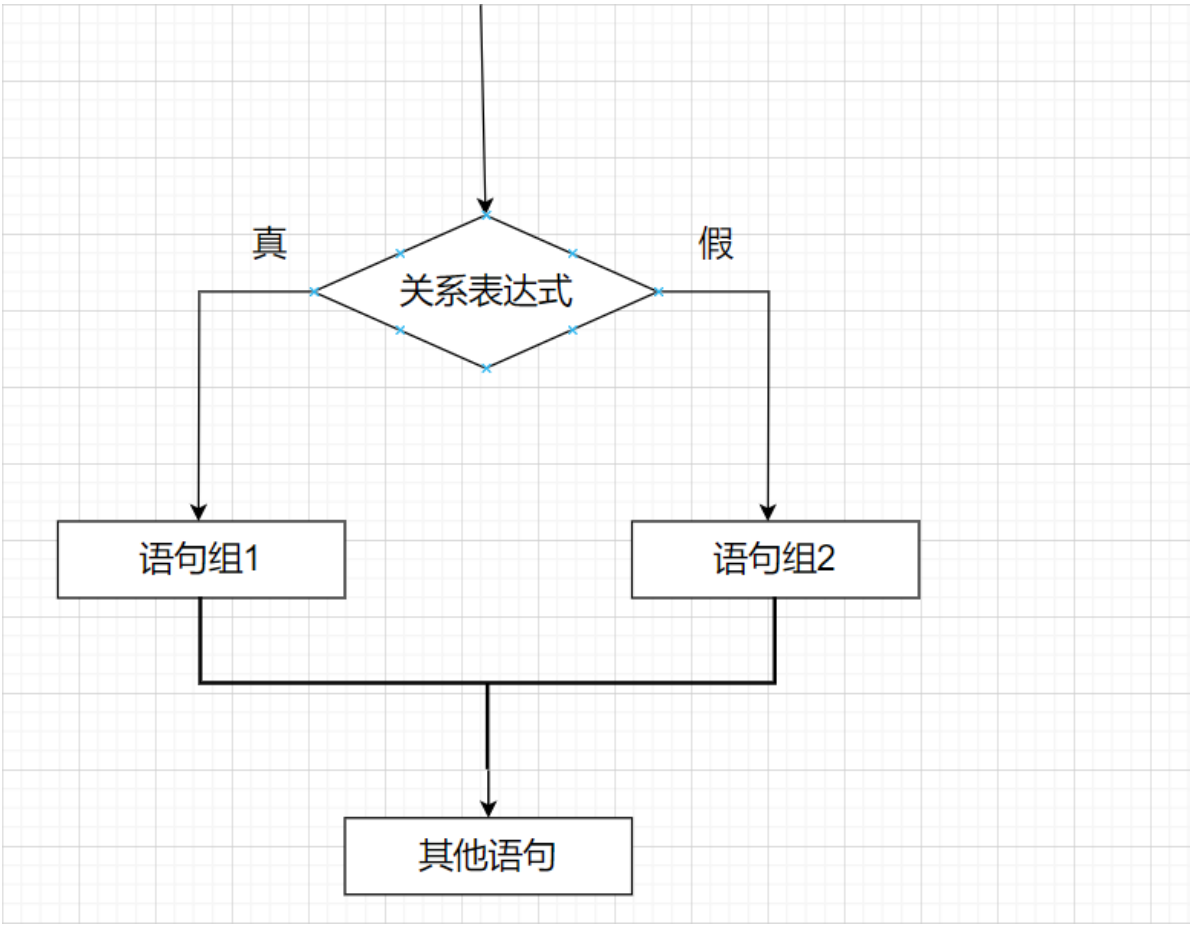
```
例子1：
[root@exercise1 opt]# vim if-1.sh   #编写脚本
#!/bin/bash

if ls /mnt
then
        echo "it's ok"
fi

[root@exercise1 opt]# sh if-1.sh
it's ok
```

双分支if语句

语法格式：

if 条件 ; then
commands1
else
commands2

fi




```
例1：
#!/bin/bash
if  `ls /odkfjpt &>/dev/null`                #此时判断条件为是否可以执行此命令
then
echo "这条命令是正确的"

else

echo "这条命令写错了"


fi
```

```
例2：
#!/bin/bash
ls /odkfjpt &>/dev/null
if  [ $? \> 0 ];then
echo "这条命令写错了"                                #条件不同，执行的命令也不同

else

echo "这条命令是正确的"

fi

```

```
例子1：

[root@exercise1 opt]# vim if-2.sh   #插入以下内容
#!/bin/bash
read  -p "检查当前目录下目录是否存在，请输入目录名：" aaa
if [[ -e /opt/$aaa  ]];
then
        echo   "目录存在"
else
        echo "目录不存在，请输入正确路径"
fi
```

```
例子2：
[root@exercise1 opt]# vim if-3.sh   #插入以下内容
#!/bin/bash

if grep ^root /etc/passwd ;
then
        echo "it's ok"
else
        echo "it's err"
fi

```

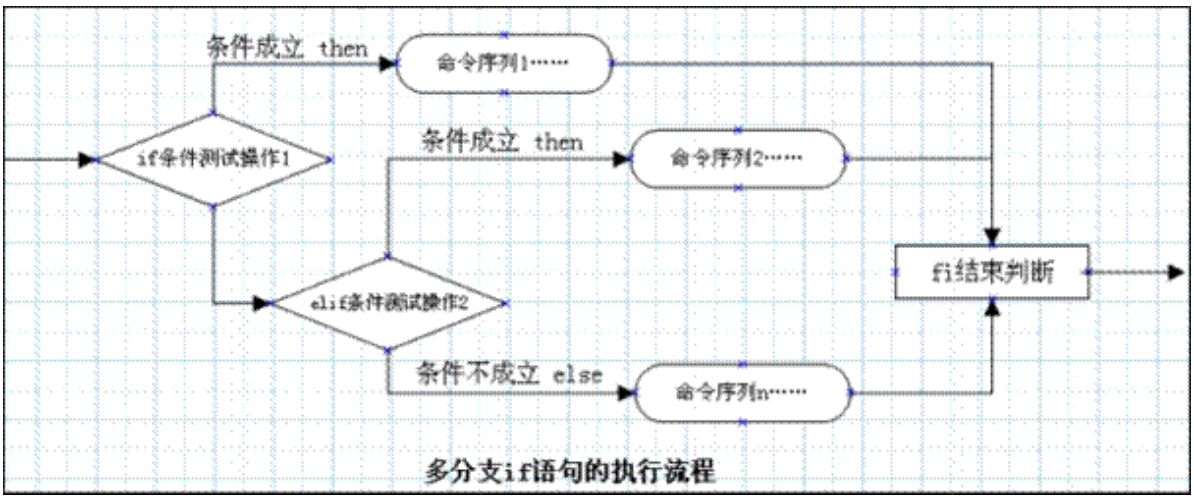
```
[root@exercise1 opt]# sh if-3.sh
root:x:0:0:root:/root:/bin/bash
it's ok
[root@exercise1 opt]#
```

```
例子3：
[root@exercise1 opt]# cat if-4.sh
#!/bin/bash
if grep dsk /etc/passwd ;then
echo "it's ok"
else
echo "it's err"
fi
[root@exercise1 opt]# sh if-4.sh
it's err
[root@exercise1 opt]#
```

多分支if语句

语法结构：

if　　条件测试操作1 ; then
elif　条件测试操作2 ; then
commands1
elif　条件测试操作3 ; then
commands2
.....
else
commands3
fi




```
判断用户在系统中是否存在，是否有家目录

方法一：
[root@exercise1 opt]# vim if-4.sh    #插入以下内容
#!/bin/bash
read -p "请输入用户名:" tu
if grep ^$tu /etc/passwd >/dev/null 2>&1; then
    echo "此用户 $tu 存在"

elif ls -d /home/$tu ; then
    echo "此用户 $tu  不存在"
    echo "$tu 有家目录"

else
    echo "此用户 $tu 不存在"
    echo "$tu 没有家目录"

fi
[root@exercise1 opt]# sh if-4.sh
请输入用户名:root
此用户 root 存在
[root@exercise1 opt]# sh if-4.sh
请输入用户名:abc
ls: 无法访问/home/abc: 没有那个文件或目录
此用户 abc 不存在
abc 没有家目录
[root@exercise1 opt]#

方法二：
[root@exercise1 opt]# vim if-5.sh    #插入以下内容
```

```
#!/bin/bash
read -p "请输入用户名：" hhh
if grep $hhh /etc/passwd
then
    echo "当前系统中存在此用户"
elif ls -d /home/$hhh
then
    echo "$hhh 用户不存在"
    echo "$hhh 有主目录"
else
    echo "系统用户不存在"
    echo "系统不存在用户目录"
fi
[root@exercise1 opt]# sh if-5.sh
请输入用户名：root
root:x:0:0:root:/root:/bin/bash
operator:x:11:0:operator:/root:/sbin/nologin
当前系统中存在此用户
[root@exercise1 opt]# sh if-5.sh
请输入用户名：abc
ls: 无法访问/home/abc: 没有那个文件或目录
系统用户不存在
系统不存在用户目录
```

---

test测试命令

Shell中的test命令用于检查某个条件是否成立，它可以进行数值、字符和文件三个方面的测试

格式：test 测试条件

如果结果是对的，也叫结果为真，用$?=0表示，反之为假，用非0表示
注：“test=[]”，[]里面左右都要有空格

数值比较

参数	说明	示例
-eq	等于则为真	["$a" -eq "$b"]
-ne	不等于则为真	["$a" -ne "$b"]
-gt	大于则为真	["$a" -gt "$b"]
-ge	大于等于则为真	["$a" -ge "$b"]
-lt	小于则为真	["$a" -lt "$b"]
-le	小于等于则为真	["$a" -le "$b"]

```
[root@exercise1 opt]# [ 2 -eq 2 ] && echo "ok" || echo "err"
ok
[root@exercise1 opt]# [ 2 -eq 1 ] && echo "ok" || echo "err"
err
```

[root@exercise1 opt]# vim if-6.sh #插入以下内容

```
#!/bin/bash
if [ "expr $((1+1))" != "expr $((2+3))" ] ; then    #  !意为取反
        echo "两个数字不相等。"
else
        echo "两个数字相等。"
fi
```

[root@exercise1 opt]# sh if-6.sh

两个数字不相等。

例子1：比较大小

```
[root@exercise1 opt]# vim if-7.sh    #插入以下内容
#!/bin/bash
if test 2 -eq 1 ; then
        echo ok
else
        echo err
fi
```

```
if [ 2 -eq 2 ] ; then
        echo ok
else
        echo err
fi
[root@exercise1 opt]# sh if-7.sh
err
ok
```

例子2：比较整数大小

[root@exercise1 opt]# vim if-8.sh #插入以下内容

```
#!/bin/bash
read -p "请输入整数一：" a
read -p "请输入整数二：" b
if expr $a+b &>/dev/null
then
    if [ "$a" -gt "$b" ]
    then
        echo " $a 大于 $b"
    elif [ "$a" -eq "$b" ]
        then
        echo " $a 等于 $b"
    else
        echo " $a 小于 $b"
    fi
else
echo "你输入的不是整数,请重新执行"
fi
```

字符串比较

参数	说明	示例
==	等于则为真	["$a" == "$b"]
=~	左侧字符串是否能够被右侧的PATTERN所匹配，注意: 此表达式一般用于[[ ]]中；扩展的正则表达式	[[$file =~ [0-9]]]
!=	不相等则为真	["$a" != "$b"]
-z 变量	变量的长度为零则为真(应用：检查是否输入密码)	[-z "$a"]
-n 变量	变量的长度不为零则为真(应用：检查是否输入密码)	[-n "$a"]
str1 > str2	str1大于str2为真 #>前一定要加转义符“\”，不然会当重定向来执行	[str1 \> str2]
str1 < str2	str1小于str2为真	[str1 \< str2]

```
例子1：
#!/bin/bash
[root@exercise1 opt]# vim if-9.sh    #插入以下内容
read -p "请输入你的名字：" name

if [ $name == "root" ]
then
        echo "管理员"
else
```

```
        echo "不是管理员"
fi
[root@exercise1 opt]# sh if-9.sh
请输入你的名字：root
管理员
```

#再此例子中只能简单的判断是root用户，如果想更加准确的进行判断还需要截取命令的配合。

例子2：在做字符串大小比较的时候，注意字符串的顺序

1. 大于号和小于号必须转义，要不然SHELL会把它当成重定向符号
2. 大于和小于它们的顺序和sort排序是不一样的
3. 在test比较测试中，它使用的是ASCII顺序，大写字母是小于小写字母的；sort刚好相反

```
扩展：
ASCII（AmericanStandardCodeforInformationInterchange，美国信息交换标准代码）
是基于拉丁字母的一套电脑编码系统，主要用于显示现代英语和其他西欧语言。
它是现今最通用的单字节编码系统，并等同于国际标准ISO/IEC646。
```

ASCII可显示字符

二进制	十进制	十六进制	图形	二进制	十进制	十六进制	图形	二进制	十进制	十六进制	图形
0010 0000	32	20	（空格）(SP)	0100 0000	64	40	@	0110 0000	96	60	`
0010 0001	33	21	!	0100 0001	65	41	A	0110 0001	97	61	a
0010 0010	34	22	"	0100 0010	66	42	B	0110 0010	98	62	b
0010 0011	35	23	#	0100 0011	67	43	C	0110 0011	99	63	c
0010 0100	36	24	$	0100 0100	68	44	D	0110 0100	100	64	d
0010 0101	37	25	%	0100 0101	69	45	E	0110 0101	101	65	e
0010 0110	38	26	&	0100 0110	70	46	F	0110 0110	102	66	f
0010 0111	39	27	'	0100 0111	71	47	G	0110 0111	103	67	g
0010 1000	40	28	(	0100 1000	72	48	H	0110 1000	104	68	h
0010 1001	41	29	)	0100 1001	73	49	I	0110 1001	105	69	i
0010 1010	42	2A	*	0100 1010	74	4A	J	0110 1010	106	6A	j
0010 1011	43	2B	+	0100 1011	75	4B	K	0110 1011	107	6B	k
0010 1100	44	2C	,	0100 1100	76	4C	L	0110 1100	108	6C	l
0010 1101	45	2D	-	0100 1101	77	4D	M	0110 1101	109	6D	m
0010 1110	46	2E	.	0100 1110	78	4E	N	0110 1110	110	6E	n
0010 1111	47	2F	/	0100 1111	79	4F	O	0110 1111	111	6F	o
0011 0000	48	30	0	0101 0000	80	50	P	0111 0000	112	70	p
0011 0001	49	31	1	0101 0001	81	51	Q	0111 0001	113	71	q
0011 0010	50	32	2	0101 0010	82	52	R	0111 0010	114	72	r
0011 0011	51	33	3	0101 0011	83	53	S	0111 0011	115	73	s
0011 0100	52	34	4	0101 0100	84	54	T	0111 0100	116	74	t
0011 0101	53	35	5	0101 0101	85	55	U	0111 0101	117	75	u
0011 0110	54	36	6	0101 0110	86	56	V	0111 0110	118	76	v
0011 0111	55	37	7	0101 0111	87	57	W	0111 0111	119	77	w
0011 1000	56	38	8	0101 1000	88	58	X	0111 1000	120	78	x
0011 1001	57	39	9	0101 1001	89	59	Y	0111 1001	121	79	y
0011 1010	58	3A	:	0101 1010	90	5A	Z	0111 1010	122	7A	z
0011 1011	59	3B	;	0101 1011	91	5B	[	0111 1011	123	7B	{
0011 1100	60	3C	<	0101 1100	92	5C	\	0111 1100	124	7C	|
0011 1101	61	3D	=	0101 1101	93	5D	]	0111 1101	125	7D	}
0011 1110	62	3E	>	0101 1110	94	5E	^	0111 1110	126	7E	~
0011 1111	63	3F	?	0101 1111	95	5F	_				
0111 1111	127	7F	DEL	DEL				删除			

```
例子：
[root@exercise1 opt]# vim if-10.sh
#!/bin/bash
var1=test
var2=Test
if [ $var1 \> $var2 ]
then
        echo "$var1 > $var2"
else
        echo "$var1 < $var2"
fi
[root@exercise1 opt]# sh if-10.sh
test > Test
```

文件类型比较

参数	说明	示例
-e 文件名	如果文件或目录存在则为真	[-e file]
-r 文件名	如果文件存在且可读则为真	[-r file]
-w 文件名	如果文件存在且可写则为真	[-w file]
-x 文件名	如果文件存在且可执行则为真	[-x file]
-s 文件名	如果文件存在且至少有一个字符则为真	[-s file]
-d 文件名	如果文件存在且为目录则为真	[-d file]
-f 文件名	如果文件存在且为普通文件则为真	[-f file]
-c 文件名	如果文件存在且为字符型文件则为真	[-c file]
-b 文件名	如果文件存在且为块特殊文件则为真	[-b file]
file1 -nt fle2	检查file1是否比file2新	[file1 -nt file2]
file1 -ot file2	检查file1是否比file2旧	[file1 -ot file2]

```
[root@exercise1 opt]# [ -f /opt/if-9.sh ] && echo "是文件" || echo "不是文件"
是文件
[root@exercise1 opt]# [ -f /opt/if-9.s ] && echo "是文件" || echo "不是文件"
不是文件

[root@home opt]# touch a.txt
[root@home opt]# [ -s a.txt ] && echo 此文件不为空 ||echo 此文件为空
此文件为空
[root@home opt]# echo 123 >>a.txt
[root@home opt]# [ -s a.txt ] && echo 此文件不为空 ||echo 此文件为空
此文件不为空
```

```
例子1：
[root@exercise1 opt]# vim if-11.sh   #插入以下内容
#!/bin/bash

if [ -e /etc/passwd ] ; then
        echo ok
else
        echo err
fi
[root@exercise1 opt]# sh if-11.sh
ok
```

```
例：
[root@exrcise1 init.d]# action "hehe" /bin/true
-bash: action: 未找到命令
[root@exrcise1 init.d]# [ -f /etc/init.d/functions] &&
source /etc/init.d/functions
[root@exrcise1 init.d]# action "hehe" /bin/true
hehe
  [  确定  ]
[root@exrcise1 init.d]# ^C
[root@exrcise1 init.d]# action "hehe" /bin/false
hehe
  [失败]
```

例子2：

[root@exercise1 opt]# test -e /etc/aaa.txt && echo ok || echo err

err

[root@exercise1 opt]# test -e /etc/passwd && echo ok || echo err

ok

[root@exercise1 opt]# test -e /etc/ && echo ok || echo err

ok

例子3：清空日志目录

```
[root@exercise1 opt]# vim /opt/log.sh   #插入以下内容
#!/bin/bash
if [ $USER != "root" ] ; then          #USER是环境变量自带的,使用环境变量USER即可
        echo "脚本需要 root 用户执行"
        exit 10
else
        echo "脚本符合root用户执行条件"
fi
if [ ! -f /var/log/messages ] ; then  #!相当于“取反”
        echo "文件不存在"
        exit 12
fi
tail -100 /var/log/messages > /var/log/mesg.tmp
> /var/log/messages
mv /var/log/mesg.tmp /var/log/messages
echo "log clean up"

[root@exercise1 opt]# sh /opt/log.sh
脚本符合root用户执行条件
log clean up
```

注：

退出码 exit ,取值范围是 0-255

任务码($?)和重定向的0、1、2不一样，条件判的是任务码

某脚本可能有问题，那么加个exit，运行后，命令行输入echo $?显示任务码可以判断哪里有问题，相当于断点测试

```
例子3：exit退出bash，并返回一个值

[root@exercise1 opt]# ssh   192.168.119.142   #需要ssh设置好两台机子才行
root@192.168.119.142'spassword:123456
Lastlogin:MonMay2820:37:412018frombase .cn
[root@exercise1 opt]#
[root@exercise1 opt]# exit  10
登出
Connectionto192.168.119.142    closed.
[root@exercise1 opt]# echo  $?
10
```

流程控制过程中复杂条件和通配符

```
判断第一种：两个条件都为真或有一个为真就执行
if [ 条件判断一 ] && (||) [ 条件判断二 ]; then
    命令一
elif [ 条件判断三 ] && (||) [ 条件判断四 ]; then
    命令二
else
    执行其它
fi

-------------------------------------------------------------------------

判断第二种
if [ 条件判断一 -a (-o) 条件判断二 -a (-o) 条件判断三 ]; then
elif [ 条件判断三 -a (-o) 条件判断四 ]; then
else 执行其它
fi

-------------------------------------------------------------------------

判断第三种
```

```
if [[ 条件判断一 && (||) 条件判断二 ]]; then
elif [[ 条件判断三 && (||) 条件判断四 ]]; then
else
执行其它
fi
```

例子1：查看系统/etc/profile的创建文件或者目录的umask判断参考，写一个脚本

```
if [ $UID -gt 199 ] && [ "`/usr/bin/id -gn`" = "`/usr/bin/id -un`" ]; then
    umask 002
else
    umask 022
fi
```

```
[root@exercise1 opt]# vim /opt/umask.sh
#插入以下内容

#!/bin/bash

if [ $UID -gt 199 ] && [ "`/usr/bin/id -gn` "="
`/usr/bin/id -un`" ] ; then
        echo "umask 002"
else
        echo "i am root:umask 022"
fi
[root@exercise1 opt]# sh /opt/umask.sh
i am root:umask 022
```

例子2：[[。。。]]和[。。。]的区别

[[...]]运算符是[...]运算符的扩充；[[...]]能够支持*，<、>等符号且不需要转义符

```
例子1：
[root@exercise1 opt]#  if [[ $USER == r* ]] ; then echo "hello,$USER" ; else echo $USER not ; fi
hello,root
注：$USER==r*对比时，r*表示以r开头的任意长度字符串，这样就包括root

当只有一个[]方括号时：
[root@exercise1 opt]#  if [ $USER == r* ] ; then echo "hello,$USER" ; else echo $USER not ; fi
root not
#对比时r*，就表示两个字符串r*

也可以这样写：
[root@exercise1 opt]#  if [[ $USER == [a-z]oot ]] ; then echo "hello,$USER" ; else echo $USER not ;  fi
```

[[。。。]]和[。。。]的区别汇总：

1、所有的字符与逻辑运算符直接用“空格”分开，不能连到一起。

2、在[...]表达式中，常见的>、<需要加转义符\，大小比较

3、进行逻辑运算符&&、||比较时；如果用的[]符号，则用在外面，如[...]&&[...]||[...]如果在[...]里面进行逻辑与或的比较，则用-a、-o进行表示，如[x=y -a x<z   -o x>m]

4、[[...]]运算符只是[...]运算符的扩充；能够支持<、>符号运算不需要转义符；它还是以字符串比较大小。里面支持逻辑运算符||、&&，不再使用-a、-o

5、[[...]]用&&而不是-a表示逻辑“并且”；用||而不是-o表示逻辑“或”

6、[[...]]可以进行算术扩展，而[...]不可以

7、[[...]]能用正则，而[...]不行

8、双括号(())用于数学表达式

9、双方括号号[[]]用于高级字符串处理，比如“模糊匹配”

10、[[...]]对于字符串的比较支持并不好，尤其在使用[[...]]和<,>符号进行比较的时候会出现返回值错误的情况。

---

shell中的通配符

字符	含义	示例
*	匹配0或多个字符	a*b，a与b之间可以有任意长度的任意字符,也可以一个也没有,如aabcb,axyzb,a012b,ab
?	匹配任意一个字符	a?b，a与b之间必须也只能有一个字符,可以是任意字符,如aab,abb,acb,a0b
[list]	匹配list中的任意单一字符	a[xyz]b，a与b之间必须也只能有一个字符,但只能是x或y或z,如:axb,ayb,azb
[!list]	匹配除list中的任意单一字符	a[!0-9]b，a与b之间必须也只能有一个字符,但不能是阿拉伯数字,如axb,aab,a-b
[c1-c2]	匹配c1-c2中的任意单一字符如：[0-9] [a-z]	a[0-9]b，0与9之间必须也只能有一个字符如a0b,a1b...a9b
{string1,string2,...}	匹配sring1或string2(或更多)其一字符串	a{abc,xyz,123}ba与b之间只能是abc或xyz或123这三个字符串之一

```
[root@exercise1 opt]# ls /etc/*.conf
/etc/asound.conf  /etc/GeoIP.conf  /etc/libaudit.conf   /etc/mke2fs.conf    /etc/sudo.conf
/etc/chrony.conf  /etc/host.conf   /etc/libuser.conf    /etc/nsswitch.conf  /etc/sudo-ldap.conf
/etc/dracut.conf  /etc/kdump.conf  /etc/locale.conf     /etc/resolv.conf    /etc/sysctl.conf
/etc/e2fsck.conf  /etc/krb5.conf   /etc/logrotate.conf  /etc/rsyslog.conf   /etc/vconsole.conf
/etc/fuse.conf    /etc/ld.so.conf  /etc/man_db.conf     /etc/sestatus.conf  /etc/yum.conf
[root@exercise1 opt]# ls /etc/???.conf
/etc/yum.conf
[root@exercise1 opt]# touch /opt/a{1,2,3}.txt
[root@exercise1 opt]# ls /opt/a[123].txt
/opt/a1.txt  /opt/a2.txt  /opt/a3.txt
[root@exercise1 opt]# ls /opt/a[1,2,3].txt
/opt/a1.txt  /opt/a2.txt  /opt/a3.txt
[root@exercise1 opt]# ls /opt/a[13].txt
/opt/a1.txt  /opt/a3.txt
```

作业：

1.执行脚本输出以下内容

1.1.当前主机名称

1.2.当前的IP地址

1.3.当前的外网IP

1.4.当前的虚拟平台（hostnamectl ）

1.5.当前系统版本

1.6. 当前内核版本

1.7.当前cpu型号（lscpu）

1.8.当前内存的使用率

1.9.当前磁盘的使用率

安成后只要连接xshell就显示以上内容

2.使用两种传参的方式判断两个数值的大小

3.统计磁盘的使用率，并输出当前的使用率，如果磁盘使用率大于百分之2，则发送邮件到邮箱（echo send mail .....）

如果磁盘使用率小于百分之2，则提示磁盘使用正常

4.统计内存使用率，并输出当前内存使用率，使用率超过百分之二，则发送邮箱报警（echo send mail .....），否则输出提示使用正常

5.通过不同的系统版本号 安装不同的yum网络源

需要有判断网络是否通 如果不通配IP

例：输入两个整数 使用if判断两个整数谁大谁小或等于

```
#!/bin/bash
read -p "请输入两个整数：" num1 num2
expr $num1 + $num2 &>/dev/null
if [ $? -ne 0 ];then
 echo 请输入正确的整数
 exit
fi

if [ $num1 -gt $num2 ];then
 echo "$num1 > $num2"

elif [ $num1 -lt $num2 ];then
 echo "$num1 < $num2"

else
 echo "$num1 = $num2"

fi
```

例：判断是否安装文件成功

```
#!/bin/bash
if [ `rpm -qa wget|wc -l` -eq 0 ];then
   yum  -y install wget &>/opt/wget.log
       if [ $? -eq 0 ];then
        echo "wget安装成功"
       else
        echo "wget安装失败，请检查网络......"
       fi
else
   echo "wget已经安装不需要重复安装"
fi

```

6.实战-11个shell脚本实战

实战1：编写脚本在一个目录创建文件，并输出文件是什么类型的，并且需要规定创建的文件名是只包含英文字母与数字，不符合的不能创建.

实战2:根据学生的成绩判断学生的优劣,低于60分不合格，60-70良好，71-85好，86到100优秀

实战3：每天晚上3:00，备份/etc目录里，使用系统日期做备份文件名。打包前需检测目录存不存在，
打包是否成功需提示，最多只能备份4份

实战4：自定义创建账号，如果用户不输入用户名，则提示必须输入用户名并退出脚本;如果用户不输入密码，则统一使用默认的123456作为默认密码

实战5：依次提示用户输入3个整数，脚本根据数字大小依次排序输出3个数字（用两种方法）

实战6：手动输入IP，脚本能自动检测该主机是处于开机状态还是关机状态

实战7：先手动新添加一块硬盘，用脚本实现自动对磁盘分区，格式化，挂载

```
#!/bin/bash
#对虚拟机的 sdb 磁盘进行分区格式化，使用<<将需要的分区指令导入给程序 fdisk
#n（新建分区），p（创建主分区），1（分区编号为 1），两个空白行（两个回车，相当于将整个磁盘分一个区）
```

实战8：用脚本检测vsftp存在是否存在，如不存在，自动安装

实战9：新添加一块硬盘，使用脚本自动创建逻辑卷

需求1：需要弹出警告提示，提示内容自行补充

需求2：提示用户输入相关参数（磁盘、卷组名称等数据），并测试用户是否输入了这些值，如果没有输入，则脚本退出

实战10：提示用户输入年份后测试判断是否为闰年

提示1：能被4不能被100整除的年是闰年

提示2：能被400整除的年也是闰年

实战11：每当执行date命令时，根据计算机当前时间，返回问候语

提示：0-12点为早晨，12-18点为下午，18-24点为晚上