目 录

1 if-then 语句

if-then 语句的格式如下所示：

```
if command
then
    commands
fi
```

这个语句的意思是，如果成功执行 command 命令，那么将执行 then 后面的所有命令。如果命令的退出状态是 0 以外的其他值，那么 then 后面的命令将不会执行。例子如下：

```
#!/bin/bash
if date
then
    echo "it worked"
fi
```

需要注意的是，command 部分可以有多条命令，这些命令之间用分号隔开，是否执行 commands 部分由 command 的最后一条命令的退出状态决定。

if-then 语句的另一种形式为：

```
if command; then
    commands
fi
```

2 if-then-else 语句

if-then-else 语句的格式如下所示：

```
if command
then
    commands
else
    commands
fi
```

3 嵌套 if 语句

嵌套 if 语句的格式如下所示：

```
if command1
then
```

```
3       commands
4   elif command2
5   then
6       commands
7   fi
```

可以使用多个 elif 语句，格式如下：

```
1   if command1
2   then
3       commands
4   elif command2
5   then
6       commands
7   elif command3
8   then
9       commands
10  elif command4
11  then
12      commands
13  fi
```

4 test 语句

test 命令提供了一种检测 if-then 语句中条件是否为真的方法。如果 test 命令中列出的条件评估值为 true，那么 test 命令的退出状态为 0，否则为其他值。test 命令的格式如下所示：

```
1   test condition
```

在 if-then 语句中使用格式如下：

```
1   if test condition
2   then
3       commands
4   fi
```

可以使用 [] 来声明 test 命令，格式如下：

```
1   if [condition ]
2   then
3       commands
4   fi
```

需要注意的是，在前半个方括号的后面必须有个空格，在后半个方括号的前面也必须有个空格，否则会发生语法错误。

4.1 数值比较

以下是数值比较中的格式：

比较	描述	比较	描述
n1 -eq n2	检查 *n1* 是否等于 *n2*	*n1 -le n2*	检查 *n1* 是否小于或等于 *n2*
n1 -ge n2	检查 *n1* 是否大于或等于 *n2*	*n1 -lt n2*	检查 *n1* 是否小于 *n2*
n1 -gt n2	检查 *n1* 是否大于 *n2*	*n1 -ne n2*	检查 *n1* 是否不等于 *n2*

例子如下所示：

```
#!/bin/bash
val1=10
val2=11

if [ $val1 -gt 5 ]
then
    echo "The test value $val1 is greater than 5"
fi

if [ $val1 -eq $val2 ]
then
    echo "The values are equal"
else
    echo "The values are different"
fi
```

需要注意的是，数值比较仅限于整数，如果使用浮点数将报错。

4.2 字符串比较

字符串比较的格式如下所示：

比较	描述	比较	描述
str1 = str2	检查 *str1* 与 *str2* 是否相同	*str1 > str2*	检查 *str1* 是否大于 *str2*
str1 != str2	检查 *str1* 与 *str2* 是否不同	*-n str1*	检查 *str1* 的长度是否大于 0
str1 < str2	检查 *str1* 是否小于 *str2*	*-z str1*	检查 *str1* 的长度是否为 0

需要注意的是，使用大于号或小于号的时候，需要在它们前面加上反义符号，否则会被当作重定位符号。例子如下所示：

```
#!/bin/bash
val1=baseball
val2=hockey

if [ $val1 \> $val2 ]
then
    echo "$val1 is greater than $val2"
else
```

```
9         echo "$val1 is less than $val2"
10    fi
```

需要注意的是，test 命令使用 ASCII 码的大小对字符串进行排序。

4.3 文本比较

test 命令可以测试 linux 文件系统上的文件状态和路径，如下所示：

-d file		检查 file 是否存在并且是一个目录
-e file		检查 file 是否存在
-f file		检查 file 是否存在并且是一个文件
-r file		检查 file 是否存在并且可读
-s file		检查 file 是否存在并且不为空
-w file		检查 file 是否存在并且可写
-x file		检查 file 是否存在并且可执行
-O file		检查 file 是否存在并且被当前用户拥有
-G file		检查 file 是否存在并且默认组是否为当前用户组
file1 file2	-nt	检查 file1 是否比 file2 新
file1 file2	-ot	检查 file1 是否比 file2 旧

需要注意的是，如果要在-nt 或-ot 比较中使用文件，必须保证两个文件存在，否则返回值为 false。

5 复合条件检查

两个条件的与如下所示：

```
1    [ condition1 ] && [ condition2 ]
```

两个条件的或如下所示：

```
1    [ condition1 ] || [ condition2 ]
```

6 if-then 的高级特征

6.1 使用双圆括号

双圆括号的格式如下所示：

```
(( expression ))
```

双圆括号中允许在比较中包含高级数学公式，还可以使用一些常用的数学符号，如下图所示：

符号	描述	符号	描述
val++	后增量	<<	逐位左移
val--	后减量	>>	逐位右移
++val	前增量	&	逐位布尔逻辑与
--val	前减量	|	逐位布尔逻辑或
!	逻辑否定	&&	逻辑与
~	逐位取反	||	逻辑或
**	取幂		

例子如下所示：

```
#!/bin/bash
val1=10

if (( $val1**2 > 90 ))
then
    (( val2 = $val1**2 ))
    echo "The square of $val1 is $val2"
fi
```

需要注意的是，在双圆括号中，< 和 > 都不会被认为是重定向符号。反正在双圆括号中可以使用在 c 语言中常用的数学符号。

6.2 使用双方括号

双方括号的格式如下所示：

```
[[ expression ]]
```

双方括号提供了模式匹配功能，可以使用正则表达式，例子如下所示：

```
#!/bin/bash
# r*就是正则表达式
if [[ $USER == r* ]]
then
    echo "Hello $USER"
else
```

```
7         echo "Sorry, I don't know you"
8     fi
```

7 case 命令

case 命令的格式如下所示：

```
1     case variable in
2     pattern1 | pattern2)
3         commands1;;
4     pattern3)
5         commands2;;
6     *)
7     default commands;;
8     esac
```

可以在一行中列出多个模式，使用竖条操作符将每个模式分开。星号可以匹配任何的模式。例子如下：

```
1     #!/bin/bash
2     case $USER in
3     rich | barbara)
4         echo "Welcome, $USER"
5         echo "Please enjoy your visit";;
6     testing)
7         echo "Special testing account";;
8     jessica)
9         echo "Don't forget to log off when you are done";;
10    *)
11        echo "Sorry, you are not allowed here"
12    esac
```

8 for 命令

for 命令的格式如下所示：

```
1     for var in list
2     do
3         commands
4     done
```

参数 list 用于提供一系列用于迭代的值，例子如下所示：

```
1     #!/bin/bash
2     for test in Alabama Alaska Arizona Arknansas California Colorado
3     do
4         echo "The next state is $test"
```

```
done
```

list 也可以是一个变量，如下例所示：

```
#!/bin/bash
list="Alabama Alaska Arizona Arknansas California Colorado"
list=$list" Connecticut"

for state in $list
do
    echo "Have you ever visited $state?"
done
```

list 还可以是命令的输出，如下例所示：

```
#!/bin/bash
file="states"

for state in `cat $file`
do
    echo "Visit beautiful $state"
done
```

list 参数还可以使用通配符，使用 for 语句遍历文件的例子如下：

```
#!/bin/bash

for file in /home/rich/test/*
do
    # 加双引号是为了避免文件名有空格
    if [ -d "$file" ]
    then
        echo "$file is a directory"
    elif [ -f "$file" ]
    then
        echo "$file is a file"
    fi
done
```

9 C 式的 for 命令

bash 中的 C 式 for 循环的格式如下所示：

```
for (( variable assignment; condition; iteration process ))
```

这里的 for 循环和 C 语言完全一样，例子如下所示：

```
#!/bin/bash
for (( i=1; i <= 10; i++ ))
do
```

```
4         echo "The next number is $i"
5     done
```

和 C 语言中的 for 循环一样，bash 中的 C 式循环也可以使用多个变量，例子如下所示：

```
1     #!/bin/bash
2
3     for (( a=1, b=10; a <= 10; a++, b-- ))
4     do
5         echo "$a - $b"
6     done
```

10 while 命令

while 命令的条件判断和 if-then 语句一样，也可以使用 test 命令来辅助条件判断，格式如下所示：

```
1     #!/bin/bash
2     while test command
3     do
4         commands
5     done
```

使用 while 循环的例子如下所示：

```
1     #!/bin/bash
2     var1=10
3
4     while [ $var1 -gt 0 ]
5     do
6         echo $var1
7         var1=$[ $var1 - 1 ]
8     done
```

while 循环可以使用多条命令，但是只以最后一条为准，如下例所示：

```
1     #!/bin/bash
2     var1=10
3
4     while echo $var1
5           [ $var1 -ge 0 ]
6     do
7         echo "This is inside the loop"
8         var1=$[ $var1-1 ]
9     done
```

11 until 命令

until 命令格式如下：

```
until test command
do
    other commands
done
```

使用例子如下：

```
#!/bin/bash
var1=100

until echo $var1
      [ $var1 -eq 0 ]
do
    echo "Inside the loop: $var1"
    var1=$[ $var1-25 ]
done
```

12 文件数据的循环

通过改变字段分隔符 IFS 这个环境变量，可以通过修改 IFS 来修改字段分隔符。
如果想只识别换行符，IFS 值为：

```
# 注意换行符前面要带一个$
IFS=$'\n'
```

如果想识别冒号，IFS 值为：

```
IFS=:
```

例子如下：

```
#!/bin/bash
IFS.OLD=$IFS
IFS=$'\n'
for entry in `cat /etc/passwd`
do
    echo "Values in $entry -"
    IFS=:
    for value in entry
    do
        echo " $value"
    done
done
```

13 控制循环

13.1 break 命令

使用 break 命令可以退出任何类型的循环。不同于 C 语言的 break 命令，bash 的 break 命令可以跳出多级循环，格式如下所示：

```
# n是要跳出的循环级别
break n
# 如果不带参数，就是跳出所在的循环
break
```

例子如下：

```
#!/bin/bash

for (( a=1; a < 4; a++ ))
do
    echo "Outer loop: $a"
    for (( b=1; b < 100; b++ ))
    do
        if [ $b -gt 4]
        then
            break 2
        fi
    done
done
```

13.2 continue 命令

continue 命令用于提前停止当前循环，和 break 命令一样，continue 命令也可以指定要提前停止的循环级别，格式如下所示：

```
# n是要提前停止的循环级别
continue n
# 不带参数是，就是提前停止当前循环
continue
```

例子如下：

```
#!/bin/bash

for (( a=1; a <= 5; a++ ))
do
    echo "Iteration $a:"
    for (( b=1; b < 3; b++ ))
    do
        if [ $a -gt 2 ] && [ $a -lt 4 ]
        then
```

```
10                continue 2
11            fi
12            var3=$[ $a * $b ]
13            echo "The result of $a * $b is $var3"
14        done
15    done
```

14 处理循环的输出

可以在 done 命令后面添加重定位输出符号或管道符号来向某个文件循环输出结果，例子如下所示：

```
1     for file in /home/rich/*
2     do
3         if [ -d "$file" ]
4         then
5             echo "$file is a directory"
6         elif
7             echo "$file is a file"
8         fi
9     done > output.txt
```

使用管道的例子如下所示：

```
1     #!/bin/bash
2     for state in "North Dakota Connecticut Illinois Alabama Tennessee"
3     do
4         echo "$state is the next place to go"
5     done | sort
```