CentOS 丛书目录 — 系统管理 — 网络服务 — 应用部署

条件测试

内容提要

1. 学会使用 test 和 [] 书写条件测试
2. 学会使用 [[]] 书写条件测试
3. 区分 [] 和 [[]] 的书写规则
4. 区分 [[]] 和 (()) 的书写规则

命令执行顺序

bash 允许在一个命令行上可以执行多条命令，没个命令之间可以用如下的符号间隔：

- **;** ： 用 ; 间隔的各命令按顺序依次执行
- **&&** ： 前后命令的执行存在"逻辑与"关系，只有&&前面的命令执行成功后，它后面的命令才被执行
- **||** ： 前后命令的执行存在"逻辑或"关系，只有||前面的命令执行失败后，它后面的命令才被执行

优先级为：

- ; 的优先级最低
- ||和&&具有相同的优先级
- 同优先级，按从左到右的结合原则执行命令行
- 使用( )可以组合命令行中的命令，改变执行顺序

使用举例：

```
$ date ; pwd ; ls
# 顺序执行date、pwd和ls命令。
$ mail jjh < message && rm message
# 若文件message被mail发送出去，就把它删除，否则不删除。
$ write jjh < report || mail jjh < report
# 若对方拒绝对话，就将信息发送到他的信箱里。
$ date ; cat file |wc
# 只有cat命令的信息通过管道送给wc命令。
$ (date; cat file) |wc
# date和cat命令的信息都通过管道送给wc命令。
```

测试语句

在 bash 的各种流程控制结构中通常要进行各种测试，再根据测试结果执行不同的操作。测试语句语法如下：

```
格式1： test <测试表达式>
格式2： [ <测试表达式> ]
格式3： [[ <测试表达式> ]]
```

说明：

- 格式1 和 格式2 是等价的
- 格式3是扩展的 test 命令

- 在[[ ]]中可以使用通配符进行模式匹配
- &&, ||, <, 和> 操作符能够正常存在于[[ ]]中，但不能在 [] 中出现
- 要对整数进行关系运算也可以使用Shell的算术运算符 (()) 进行测试

文件测试操作符

在书写测试表达式时，可以使用如下的文件测试操作符：

文件测试表达式	说明
-e file	文件是否存在
-f file	是否为普通文件
-d file	是否为目录文件
-L file	是否为符号链接文件
-b file	是否为块设备文件
-c file	是否为字符设备文件
-s file	文件长度不为0（非空文件）
-r file	是否为只读文件
-w file	是否为可写文件
-x file	是否为可执行文件
-u file	是否为设置了set-user-id (suid)标记的文件
-g file	是否为设置了set-group-id(sgid)标记的文件
-k file	是否为设置了save-text-mode标记（粘贴位）的文件
-O file	测试者是否是文件的属主
-G file	测试者是否是文件的同组人
file1 -nt file2	file1 是否比 file2 新
file1 -ot file2	file1 是否比 file2 旧
file1 -ef file2	file1 是否与 file2 共用相同的 i-node (硬链接)

字符串测试操作符

在书写测试表达式时，还可以使用如下的字符串测试操作符：

文件测试操作符	说明
-z string	测试字符串是否为空串
-n string	测试字符串是否为非空串
string1 = string2	测试两个字符串是否相同（也可以使用 == 代替 =）
string1 != string2	测试两个字符串是否不相同

整数二元比较操作符

在书写测试表达式时，还可以使用如下的整数二元比较操作符：

在**[]**中使用的比较符	在**(())**中使用的比较符	说明
-eq	==	相等
-ne	!=	不等
-gt	>	大于
-ge	>=	大于等于

-lt	<	小于
-le	<=	小于等于

使用逻辑运算符

还可以使用如下的操作符连接测试表达式实现复杂的条件测试：

在**[]**中使用的逻辑连接符	在**[[]]**中使用的逻辑连接符	说明
-a	&&	实现"与"逻辑
-o	||	实现"或"逻辑
!	!	实现"非"逻辑

测试语句举例

下面举一些条件测试的例子。为了清晰地显示测试的结果，使用了打印 真/假 的输出。

```
[ -e "$file1" ] && echo true || echo false
```

与下面的 if 语句

```
if [ -e "$file1" ]; then echo true; else echo false; fi
```

等效。

文件测试举例

```
$  file1=/etc/passwd;file2=/etc/shadow
$  [ -e "$file1" ] && echo true || echo false
true  # 由于在正常的系统中肯定存在文件 /etc/passwd
$  [ -d "$file1" ] && echo true || echo false
false # 由于 /etc/passwd 是文件而并非目录
$  [ -s "$file1" ] && echo true || echo false
true  # 由于 /etc/passwd 不是空文件
$  [ -x "$file1" ] && echo true || echo false
false # 由于 /etc/passwd 无执行权限
$  [ -k "/tmp"  ] && echo true || echo false
true  # 由于 /tmp 设置了粘着位
$  [ -u "/usr/bin/passwd" ] && echo true || echo false
true  # 由于 /usr/bin/passwd 设置了 SUID
$  [ -O "$file1" ] && echo true || echo false
false # 由于测试者不是 /etc/passwd 文件的属主
$  [ "$file2" -nt "$file1" ] && echo true || echo false
true   # 由于 /etc/shadow 比 /etc/passwd 新
$  [ -f "$file1" -a -O "$file1" ] && echo true || echo false
false # 由于 /etc/passwd 虽然是普通文件但测试者不是文件的属主，与值为假
$  [ -f "$file1" -o -O "$file1" ] && echo true || echo false
true  # 由于 /etc/passwd 虽然是普通文件但测试者不是文件的属主，或值为真
$  [[ -f "$file1" && -O "$file1" ]] && echo true || echo false
false  # 在 [[]] 中 用 && 代替 [] 中的 -a
$  [[ -f "$file1" || -O "$file1" ]] && echo true || echo false
true  # 在 [[]] 中 用 || 代替 [] 中的 -o
```

字符串测试举例

```
$  [ -n "$file" ] && echo true || echo false
false # 由于 file 变量尚未定义，值为空
$  [ -n "$file1" ] && echo true || echo false
true  # 由于上面定义了 file1 的值为 /etc/passwd，非空
$  [ "$file1" ] && echo true || echo false
true  # 与上一条语句等价
$  [ ! -z "$file1" -a "$file2" ] && echo true || echo false
true  # 由于 file1 和 file2 均为非空，与后值为真
$  [[ ! -z "$file1" && "$file2" ]] && echo true || echo false
true  # 在 [[]] 中 用 && 代替 [] 中的 -a
$  str1=centos;str2=ubuntu;str3=centos   # 再定义三个字符串变量
$  [ "$str1" = "$str2" ] && echo true || echo false
false  # 由于 centos 和 ubuntu 不同
$  [ "$str1" != "$str2" ] && echo true || echo false
true  # 由于 centos 和 ubuntu 不同
$  [ "$str1" == "$str3" ] && echo true || echo false
true  # 由于 centos 和 centos 相同
$  [[ $str1 > $str2 ]] && echo true || echo false
false  # 由于 c 的 ASCII 码比 u 的 ASCII 码小
$  [ $str1 \> $str2 ] && echo true || echo false
false  # 在 [] 中做字符串大小的比较，必须用 \> 和 \<
```

整数测试举例

```
$  [ "$int1" -eq 0  ] && echo true || echo false
false # 由于变量 int1 尚未定义，其值为空，不为 0
$  int1=10;int2=3   # 定义两个变量
$  [ $int1 -eq $int2 ] && echo true || echo false
false # 由于 10 不等于 3
$  (($int1==$int2)) && echo true || echo false
false # 与上一条语句等价
$  [ $int1 -ne $int2 ] && echo true || echo false
true  # 由于 10 不等于 3
$  (($int1!=$int2)) && echo true || echo false
true  # 与上一条语句等价
$  [ $int1 -gt $int2 ] && echo true || echo false
true  # 由于 10 大于 3
$  ((int1>int2)) && echo true || echo false
true # 由于 10 大于 3
$  (($int1>$int2)) && echo true || echo false
true # 与上一条语句等价
$  (( $int1>$int2 )) && echo true || echo false
true # 与上一条语句等价
$  [[ $int1>$int2 ]] && echo true || echo false
false # 注意与上句的区别，此语法是做字符串比较，由于 1 的 ASCII 码比 3 的小
```

1. 为了避免混淆，当涉及数值的关系或逻辑运算时应该尽量使用(())
2. 在 (()) 中引用变量的值时，加 $ 或不加 $ 均可
3. 与在 [] 和 [[]] 中不同，在 (()) 中不必使用左右的空格

- 显示源文件

- 登录