Linux——shell脚本基础2：条件测试及控制流

[1.条件测试 1](#_Toc323506997)

[A.文件条件测试 2](#_Toc323506998)

[B.字符串测试 3](#_Toc323506999)

[C.数值测试 3](#_Toc323507000)

[2.控制流 4](#_Toc323507001)

[A.if-else 4](#_Toc323507002)

[B.case语句 6](#_Toc323507003)

[C.for循环 7](#_Toc323507004)

[D.until循环 9](#_Toc323507005)

[E.while循环 10](#_Toc323507006)

[F.补充break&continue 11](#_Toc323507007)

### 1.条件测试

|  |
| --- |
| Test |
| 命令用于测试字符串、文件状态和数字，也同控制流相结合  文件状态判断；  字符串比较；  数字比较；  逻辑判断； |

|  |
| --- |
| 语法 |
| 两种格式: test condition  或 [ condition ] 注意两边加了空格  中括号（［］）、参数之间必须有一个空格 |

|  |  |
| --- | --- |
| 逻辑运算符 | |
| -a | 逻辑与 |
| -o | 逻辑或 |
| ! | 逻辑否 |
| &&, || |  |
| 示例： | |
| 1. 测试两文件是否均可读   $[ -w result.txt –a –w scores.txt ]  $echo $?   1. 测试两文件中其中一个是否可执行   $[ -x dream –o –x dream2 ]   1. 判断是否可写可执行   $[ -w dream –a –x dream ]  $echo $?   1. 判断文件是非可执行文件   $[ ! –x dream ]   1. [条件判断表达式1]&& [条件判断表达式2]|| [条件判断表达式3] | |

#### A.文件条件测试

|  |  |
| --- | --- |
| 文件条件测试 | |
| 类型 | |
| -d | 目录，是否存在且是目录 |
| -f | 普通文件（Regular file）,是否存在且是文件 |
| -L | 符号链接 |
| -r | Readable（文件、目录可读） |
| -b | 块专用文件 |
| -c | 字符专用文件 |
| -e | 文件存在 |
| -g | 如果文件的set-group-id位被设置则结果为真 |
| -s | 文件长度大于0，非空 |
| -z | 文件长度=0 |
| -L | 符号链接 |
| 权限 | |
| -r | 可读 |
| -w | Writable（文件、目录可写） |
| -x | Executable（文件可执行、目录可浏览） |
| -u | 文件有suid位设置 |
| -s | 文件存在且非空白文件 |
| 两个文件比较 | |
| -nt | new er than 判断file1是否比file2新 |
| -ot | older than |
| -ef | 是否为同一文件 |
| 示例 | |
| Set-uid位授予程序其拥有者的访问权限而不是其使用者的访问权限  Set-gid位授予了程序其所在组的访问权限  示例   1. 测试文件是否可写   $test –w dream  $echo $?  或者  $[ -w dream ]   1. 测试目录是否存在   $[ -d appsbin ]   1. 测试文件权限是否设置了suid   $[ -u dream]  if test –f fred.c if [ -f fred.c ]  then  …..  fi | |

#### B.字符串测试

|  |  |
| --- | --- |
| 字符串测试 | |
| 字符串测试是错误捕获很重要的一部分，特别：测试用户输入或比较变量时尤为重要 | |
| 五种格式: | |
| test “str”  test str\_operator “str”  test “str1” str\_operator “str2”  [ string\_operator str1 ]  [ string string\_operator string2 ] | |
| String\_operator 字符串比较 | |
| = | 两字符串相等 |
| != | 两字符串不等 |
| -z | 空串 [zero] |
| -n | 非空串 [nozero] |
| 示例： | |
| 1. 测试环境变量是否为空   $[ -z $EDITOR ]  $echo $?   1. 测试是否为某字符串   $[ $EDITOR = “vi” ]  $echo $?   1. 测试   $TYPE=”/dev/rmt0”  $TYPE2=”/dev/rmt1”  $[ “$TYPE” = “$TYPE2” ]  $echo $? | |

#### C.数值测试

|  |  |
| --- | --- |
| 数值测试 | |
| 格式 | |
| “number” numberic\_operator “number”  或 [ “number” numberic\_operator “number”] | |
| Numberic\_operator 算术比较 | |
| -eq | 数值相等（equal） |
| -ne | 不等（not equal） |
| -gt | A>B（greater than） |
| -lt | A<B（less than） |
| -le | A<=B（less、equal） |
| -ge | A>=B（greater、equal） |
| 示例： | |
| 1. 比较相等   $NUMBER=130  $[ $NUMBER –eq 130 ] #$[ “$NUMBER” –eq “130” ]  $echo $?   1. 比较两变量   $SOURCE=13  $DEST=15  $[ “$SOURCE” –gt “$DEST”]  $echo $?   1. 直接比较数值   $[ 990 –le 995 –a 123 –gt 33 ] | |

### 2.控制流

#### A.if-else

|  |
| --- |
| If then else语句 |
| 格式 |
| if 条件1  then  命令1  elif 条件2  then  命令2  else  命令3  fi |
| if 条件 if 条件;then  then 命令 等价 命令  fi fi |
| 示例 |
| 1. 简单的if语句   $if [ “10” –lt “12” ]  > then  > echo “10 is less than 12”  >fi  2. 直接上命令  $if grep “dave” data.file > /dev/null 2>&1  >then  > echo “Dave in”  > else  > echo “Dave not in”  >fi  3. 测试文件拷贝是否成功  $if cp myfile myfile.bak  > then  > echo “Good copy”  > else  > echo “basename:$0:could not copy”  >fi  4. 当前目录测试  $DIRECTORY=`pwd` //反引号  $if [ “$DIRECTORY” != “/” ]  >then  > echo “Not root”  >exit 1  >fi  5. 文件权限测试  $LOGFILE=test.txt  $if [ ! –w “LOGFILE” ]  >then  > echo “you cannot write to $LOGFILE” >&2  >fi  6.变量设置测试  $if [ -z $EDITOR ]  >then  > echo “Your EDITOR environment is not set”  >else  > echo “Using $EDITOR as the default editor”  >fi  7.可以多个if 嵌套，但注意层次  $if [ -d $DIR]  >then  > cd $DIR  > if [ $? = 0 ]  > then  > do something  > fi  >fi  8.多个检测值  $if [ “$ANS” = “y” ] || [ “$ANS” = “Y” ]  注意：  if [ “$myvar” == “yes” ]; then  # do something ...  fi  检查变量  当不确定变量是否存在时，加双引号通常是安全的做法。 |

#### B.case语句

|  |
| --- |
| Case语句 |
| 格式: |
| case variable in  pattern [| pattern] …..)  命令….  ;;  模式2)  命令…..  ;;  \*)  …….  ;;  esac  注意，每个模式行都以双分号结尾  把最精确的匹配放在最前面 |
| 示例： |
| 1. 简单的case语句   read ANS  case $ANS in  1)  echo “1”  ;;  2)  echo “2”  ;;  \*)  echo “not right”  exit 1  ;;  esac   1. 匹配模式使用   read ANS  case $ANS in  v100|v101)  echo “1”  ;;  V102)  echo “2”  ;;  \*)  echo “not right”  exit 1  ;;  esac   1. 提示输入y/n   read ANS  case $ANS in  y|Y|yes|Yes)  echo “yes is selected”  ;;  n|N)  echo “no is selected”  ;;  \*)  echo “`basename $0` :Unknown response”  exit 1  ;;  esac |

#### C.for循环

|  |
| --- |
| for循环 |
| 格式 |
| for 变量名 in 列表  do  命令1  命令2  done |
| 示例： |
| 1. 简单的for循环   for loop in 1 2 3 4 5  do  echo $loop  done   1. 打印字符串列表   for loop in “orange red blue grey”  do  echo $loop  done   1. 打印当前命令下的所有文件   for loop in `ls`  do  echo $loop  done   1. 对for循环使用参数   脚本中，for params in “$@” 等价于 for params  for params  do  echo “You supplied $params in $@”  done   1. 多文件转换   for files in `ls LPSO\*`  do  cat $files | tr “[a-z]” “[A-Z]” >$files  done   1. 循环计数   counter=0  for files in \*  do  counter = `expr $counter + 1`  done   1. for循环的嵌套   for 变量1 in 列表1  do  for 变量2 in 列表2  do  命令  done  done   1. bash中的for循环写法   #!/bin/bash  for i in 1 2 3 4 5 6 7 8 9 10; do  # do something  done  for i in `seq 1 10`; do  # do something  done  for i in {1..10}; do  # do something  done  示例：bash的 for 循环写法  for ((i = 1; i <= 10; i++)); do  # do something  done  C语言自诞生那天起就只有一种写法  for (i = 0; i < 10; i++) {  # do something  }   1. eg   for ((初始值；限制值；限制步长))  do  程序段  done |

#### D.until循环

|  |
| --- |
| until循环 |
| 执行一系列命令直到条件为真时停止  Until循环与while循环处理方式上刚好相反  如果循环至少要执行一次，使用while，如果可能根本不需要执行，使用until |
| 格式: |
| until condition  do  命令…..  done  测试发生在循环末尾，所以至少执行一次 |
| 示例： |
| 1. 简单的until测试   IS\_ROOT=`who | grep root`  until [ “$IS\_ROOT” ]  do  sleep 5  done   1. 监视文件被删除   LOCK\_FILE=”temp”  until [ ! –f $LOCK\_FILE ]  do  sleep 1  done  echo “file deleted”   1. 监视磁盘空间   LOCK\_OUT=`df | grep /logs | awk ‘{print $5}’| sed ‘s/%//g’ `  echo $LOCK\_OUT  until [ “$LOCK\_OUT” –gt “90” ]  do  echo “File system.Logs is nearly full”  exit  done   1. eg   until [condition]&&[condition]||[condition]  do  程序段落  done |

#### E.while循环

|  |
| --- |
| while循环 |
| 用于不断执行一系列命令，也用于从输入文件中读取数据 |
| 格式: |
| while condition  do  命令1  命令2  done |
| 示例： |
| 简单的while命令  COUNTER=0  while [ $COUNTER –lt 5 ]  do  COUNTER=`expr $COUNTER + 1`  echo $COUNTER  done   1. 循环读文件   while read LINE  do  echo $LINE done < dream   1. 读文件不同字段   while read DEPT ITEMS do  …..  done < total.txt   1. 每次读一对记录   while read rec1  do  read rec2  echo “$rec1”  echo “$rec2”  done <record.txt   1. 忽略注释行   while read LINE  do  case $LINE in  \#\*) ;;  \*)  echo $LINE  ;;  esac  done   1. 无限循环   while [ 1 ]  do  #dosomething  done  6.  while [condition]&&[condition]||[condition]  do  程序段落  done |

#### F.补充break&continue

|  |
| --- |
| 补充;break和continue控制循环 |
| Break |
| 允许跳出循环，通常在进行一些列处理后退出循环或case语句  若多重循环，可指定跳出的循环个数，如跳出两重循环 break 2  命令不执行当前循环体内break下面的语句  从当前循环退出.  在控制条件未满足之前，跳出for,while,until循环，可以为break提供一个额外参数来表明需要跳出的循环层数  默认只跳出一层循环 |
| 示例： |
| 1. 跳出case   case $ANS in  1|2|3)  Do something  break;  ;;  \*)  ;;  esac |
| continue |
| 不会跳出循环，只是跳过此循环步  命令是程序在本循体内忽略下面的语句,从循环头开始执行.  跳到下一次循环继续执行 |