条件判断 函数

循环

网道（WangDoc.com），互联网文档计划

Bash 提供三种循环语法 `for` 、 `while` 和 `until` 。

目录 [隐藏]

1. while 循环

`while` 循环有一个判断条件，只要符合条件，就不断循环执行指定的语句。

```
while condition; do
  commands
done
```

上面代码中，只要满足条件 `condition` ，就会执行命令 `commands` 。然后，再次判断是否满足条件 `condition` ，只

要满足，就会一直执行下去。只有不满足条件，才会退出循环。

循环条件 `condition` 可以使用 `test` 命令，跟 `if` 结构的判断条件写法一致。

```
#!/bin/bash

number=0
while [ "$number" -lt 10 ]; do
  echo "Number = $number"
  number=$((number + 1))
done
```

上面例子中，只要变量 `$number` 小于10，就会不断加1，直到 `$number` 等于10，然后退出循环。

关键字 `do` 可以跟 `while` 不在同一行，这时两者之间不需要使用分号分隔。

```
while true
do
  echo 'Hi, while looping ...';
done
```

上面的例子会无限循环，可以按下 Ctrl + c 停止。

`while` 循环写成一行，也是可以的。

```
$ while true; do echo 'Hi, while looping ...'; don
```

`while` 的条件部分也可以是执行一个命令。

```
$ while echo 'ECHO'; do echo 'Hi, while looping ..
```

上面例子中，判断条件是 `echo 'ECHO'` 。由于这个命令总是执行成功，所以上面命令会产生无限循环。

链接

本文源码

代码仓库

反馈

`while` 的条件部分可以执行任意数量的命令，但是执行结果的真伪只看最后一个命令的执行结果。

```
$ while true; false; do echo 'Hi, looping ...'; do
```

上面代码运行后，不会有任何输出，因为 `while` 的最后一个命令是 `false` 。

2. until 循环

`until` 循环与 `while` 循环恰好相反，只要不符合判断条件（判断条件失败），就不断循环执行指定的语句。一旦符合判断条件，就退出循环。

```
until condition; do
  commands
done
```

关键字 `do` 可以与 `until` 不写在同一行，这时两者之间不需要分号分隔。

```
until condition
do
  commands
done
```

下面是一个例子。

```
$ until false; do echo 'Hi, until looping ...'; do
Hi, until looping ...
Hi, until looping ...
Hi, until looping ...
^C
```

上面代码中， `until` 的部分一直为 `false` ，导致命令无限运行，必须按下 Ctrl + c 终止。

```
#!/bin/bash

number=0
until [ "$number" -ge 10 ]; do
  echo "Number = $number"
  number=$((number + 1))
done
```

上面例子中，只要变量 `number` 小于10，就会不断加1，直到 `number` 大于等于10，就退出循环。

`until` 的条件部分也可以是一个命令，表示在这个命令执行成功之前，不断重复尝试。

```
until cp $1 $2; do
  echo 'Attempt to copy failed. waiting...'
  sleep 5
done
```

上面例子表示，只要 `cp $1 $2` 这个命令执行不成功，就5秒钟后再尝试一次，直到成功为止。

`until` 循环都可以转为 `while` 循环，只要把条件设为否定即可。上面这个例子可以改写如下。

```
while ! cp $1 $2; do
  echo 'Attempt to copy failed. waiting...'
  sleep 5
done
```

一般来说， `until` 用得比较少，完全可以统一都使用 `while` 。

3. for...in 循环

`for...in` 循环用于遍历列表的每一项。

```
for variable in list
do
```

```
  commands
done
```

上面语法中， `for` 循环会依次从 `list` 列表中取出一项，作为变量 `variable` ，然后在循环体中进行处理。

关键词 `do` 可以跟 `for` 写在同一行，两者使用分号分隔。

```
for variable in list; do
  commands
done
```

下面是一个例子。

```
#!/bin/bash

for i in word1 word2 word3; do
  echo $i
done
```

上面例子中， `word1 word2 word3` 是一个包含三个单词的列表，变量 `i` 依次等于 `word1` 、 `word2` 、 `word3` ，命令 `echo $i` 则会相应地执行三次。

列表可以由通配符产生。

```
for i in *.png; do
  ls -l $i
done
```

上面例子中， `*.png` 会替换成当前目录中所有 PNG 图片文件，变量 `i` 会依次等于每一个文件。

列表也可以通过子命令产生。

```
#!/bin/bash

count=0
for i in $(cat ~/.bash_profile); do
  count=$((count + 1))
```

```
  echo "Word $count ($i) contains $(echo -n $i | w
done
```

上面例子中， `cat ~/.bash_profile` 命令会输出 `~/.bash_profile` 文件的内容，然后通过遍历每一个词，计算该文件一共包含多少个词，以及每个词有多少个字符。

`in list` 的部分可以省略，这时 `list` 默认等于脚本的所有参数 `$@` 。但是，为了可读性，最好还是不要省略，参考下面的例子。

```
for filename; do
  echo "$filename"
done

# 等同于

for filename in "$@" ; do
  echo "$filename"
done
```

在函数体中也是一样的， `for...in` 循环省略 `in list` 的部分，则 `list` 默认等于函数的所有参数。

4. for 循环

`for` 循环还支持 C 语言的循环语法。

```
for (( expression1; expression2; expression3 )); d
  commands
done
```

上面代码中， `expression1` 用来初始化循环条件， `expression2` 用来决定循环结束的条件， `expression3` 在每次循环迭代的末尾执行，用于更新值。

注意，循环条件放在双重圆括号之中。另外，圆括号之中使用变量，不必加上美元符号 `$` 。

它等同于下面的 `while` 循环。

```
(( expression1 ))
while (( expression2 )); do
  commands
  (( expression3 ))
done
```

下面是一个例子。

```
for (( i=0; i<5; i=i+1 )); do
  echo $i
done
```

上面代码中，初始化变量 `i` 的值为0，循环执行的条件是 `i` 小于5。每次循环迭代结束时， `i` 的值加1。

`for` 条件部分的三个语句，都可以省略。

```
for ((;;))
do
  read var
  if [ "$var" = "." ]; then
    break
  fi
done
```

上面脚本会反复读取命令行输入，直到用户输入了一个点（ `.` ）为止，才会跳出循环。

5. break，continue

Bash 提供了两个内部命令 `break` 和 `continue` ，用来在循环内部跳出循环。

`break` 命令立即终止循环，程序继续执行循环块之后的语句，即不再执行剩下的循环。

```
#!/bin/bash

for number in 1 2 3 4 5 6
do
  echo "number is $number"
  if [ "$number" = "3" ]; then
    break
  fi
done
```

上面例子只会打印3行结果。一旦变量 `$number` 等于3，就会跳出循环，不再继续执行。

`continue` 命令立即终止本轮循环，开始执行下一轮循环。

```
#!/bin/bash

while read -p "What file do you want to test?" fil
do
  if [ ! -e "$filename" ]; then
    echo "The file does not exist."
    continue
  fi

  echo "You entered a valid file.."
done
```

上面例子中，只要用户输入的文件不存在， `continue` 命令就会生效，直接进入下一轮循环（让用户重新输入文件名），不再执行后面的打印语句。

6. select 结构

`select` 结构主要用来生成简单的菜单。它的语法与 `for...in` 循环基本一致。

```
select name
[in list]
do
```

```
  commands
done
```

Bash 会对 `select` 依次进行下面的处理。

1. `select` 生成一个菜单，内容是列表 `list` 的每一项，并且每一项前面还有一个数字编号。
2. Bash 提示用户选择一项，输入它的编号。
3. 用户输入以后，Bash 会将该项的内容存在变量 `name` ，该项的编号存入环境变量 `REPLY` 。如果用户没有输入，就按回车键，Bash 会重新输出菜单，让用户选择。
4. 执行命令体 `commands` 。
5. 执行结束后，回到第一步，重复这个过程。

下面是一个例子。

```
#!/bin/bash
# select.sh

select brand in Samsung Sony iphone symphony Walto
do
  echo "You have chosen $brand"
done
```

执行上面的脚本，Bash 会输出一个品牌的列表，让用户选择。

```
$ ./select.sh
1) Samsung
2) Sony
3) iphone
4) symphony
5) Walton
#?
```

如果用户没有输入编号，直接按回车键。Bash 就会重新输出一遍这个菜单，直到用户按下 `Ctrl + c` ，退出执行。

`select` 可以与 `case` 结合，针对不同项，执行不同的命令。

```
#!/bin/bash

echo "Which Operating System do you like?"

select os in Ubuntu LinuxMint Windows8 Windows10 W
do
  case $os in
    "Ubuntu"|"LinuxMint")
      echo "I also use $os."
    ;;
    "Windows8" | "Windows10" | "WindowsXP")
      echo "Why don't you try Linux?"
    ;;
    *)
      echo "Invalid entry."
      break
    ;;
  esac
done
```

上面例子中， `case` 针对用户选择的不同项，执行不同的命令。

7. 参考链接

- Bash Select Command, Fahmida Yesmin

条件判断 函数

联系：contact@wangdoc.com